Elastic Beanstalk

Developer Guide

API Version 2010-12-01

Elastic Beanstalk: Developer Guide

Table of Contents

What Is Elastic Beanstalk and Why Do I Need It?

Amazon Web Services (AWS) comprises dozens of services, each of which exposes an area of functionality. While the variety of services offers flexibility for how you want to manage your AWS infrastructure, it can be challenging to figure out which services to use and how to provision them.

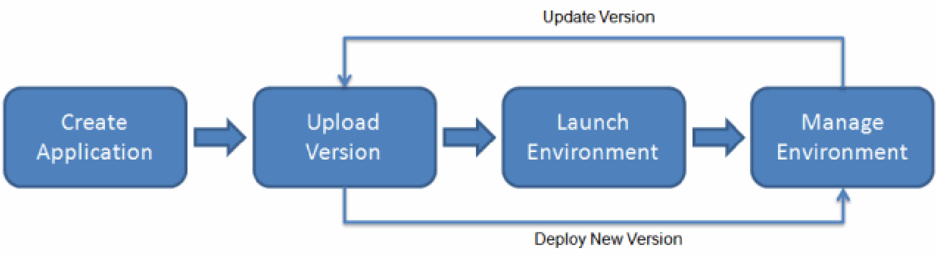
With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS cloud without worrying about the infrastructure that runs those applications. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring. Elastic Beanstalk uses highly reliable and scalable services that are available in the AWS Free Usage Tier such as:

- Amazon Elastic Compute Cloud
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon CloudWatch
- Elastic Load Balancing
- Auto Scaling
- Amazon RDS
- Amazon DynamoDB
- Amazon CloudFront
- Amazon ElastiCache

To learn more about the AWS Free Usage Tier, and how to deploy a sample web application in it using AWS Elastic Beanstalk, go to Getting Started with AWS: Deploying a Web Application.

You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface.

To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.



After you create and deploy your application, information about the application—including metrics, events, and environment status—is available through the AWS Management Console, APIs, or Command Line Interfaces, including the unified AWS CLI. For step-by-step instructions on how to create, deploy, and manage your application using the AWS Management Console, go to Getting Started Using Elastic Beanstalk (p. 4). To learn more about an Elastic Beanstalk application and its components, see Elastic Beanstalk Components (p. 15).

Elastic Beanstalk provides developers and systems administrators an easy, fast way to deploy and manage their applications without having to worry about AWS infrastructure. If you already know the AWS resources you want to use and how they work, you might prefer AWS CloudFormation to create your AWS resources by creating a template. You can then use this template to launch new AWS resources in the exact same way without having to recustomize your AWS resources. Once your resources are deployed, you can modify and update the AWS resources in a controlled and predictable way, providing the same sort of version control over your AWS infrastructure that you exercise over your software. For more information about AWS CloudFormation, go to AWS CloudFormation Getting Started Guide.

Storage

Elastic Beanstalk does not restrict your choice of persistent storage and database service options. For more information on AWS storage options, go to Storage Options in the AWS Cloud.

Pricing

There is no additional charge for Elastic Beanstalk. You pay only for the underlying AWS resources that your application consumes. For details about pricing, see the Elastic Beanstalk service detail page.

Community

Customers have built a wide variety of products, services, and applications on top of AWS. Whether you are searching for ideas about what to build, looking for examples, or just want to explore, you can find many solutions at the AWS Customer App Catalog. You can browse by audience, services, and technology. We also invite you to share applications you build with the community. Developer resources produced by the AWS community are at http://aws.amazon.com/resources/.

Where to Go Next

This guide contains conceptual information about the Elastic Beanstalk web service, as well as information about how to use the service to deploy web applications. Separate sections describe how to use the AWS Management console, command line interface (CLI) tools, and API to deploy and manage your Elastic Beanstalk environments. This guide also documents how Elastic Beanstalk is integrated with other services provided by Amazon Web Services.

We recommend that you first read Getting Started Using Elastic Beanstalk (p. 4) to learn how to start using Elastic Beanstalk. Getting Started steps you through creating, viewing, and updating your Elastic Beanstalk application, as well as editing and terminating your Elastic Beanstalk environment. Getting Started also describes different ways you can access Elastic Beanstalk. We also recommend that you familiarize yourself with Elastic Beanstalk concepts and terminology by reading How Does Elastic Beanstalk Work? (p. 15).

Getting Started Using Elastic Beanstalk

Topics

Getting started with Elastic Beanstalk is simple, and the AWS Management Console makes it easy for you to create, edit, and manage your Docker, Go, Java, PHP, .NET, Node.js, Python, and Ruby applications in a matter of minutes. The following walkthrough steps you through how to use the console to get started. You can also access Elastic Beanstalk using the AWS Toolkit for Eclipse, AWS Toolkit for Microsoft Visual Studio, AWS SDKs, APIs, and CLIs; for more information about these tools, see AWS Developer Tools. The remainder of this section provides information about each of these and where to go next.

Walkthrough

The following tasks will help you get started with Elastic Beanstalk to create, view, deploy, and update your application as well as edit and terminate your environment. You'll use the AWS Management Console, a point-and-click web-based interface, to complete these tasks.

Step 1: Sign up for the Service

If you're not already an AWS customer, you'll need to sign up. Signing up allows you to access Elastic Beanstalk and other AWS services that you will need, such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), and Amazon Simple Notification Service (Amazon SNS).

To sign up for an AWS account

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Follow the on-screen instructions.

> **Note**
> If you have never registered for Amazon EC2, part of the sign-up procedure for Elastic Beanstalk will include receiving an automated telephone call and entering a PIN using the telephone keypad.

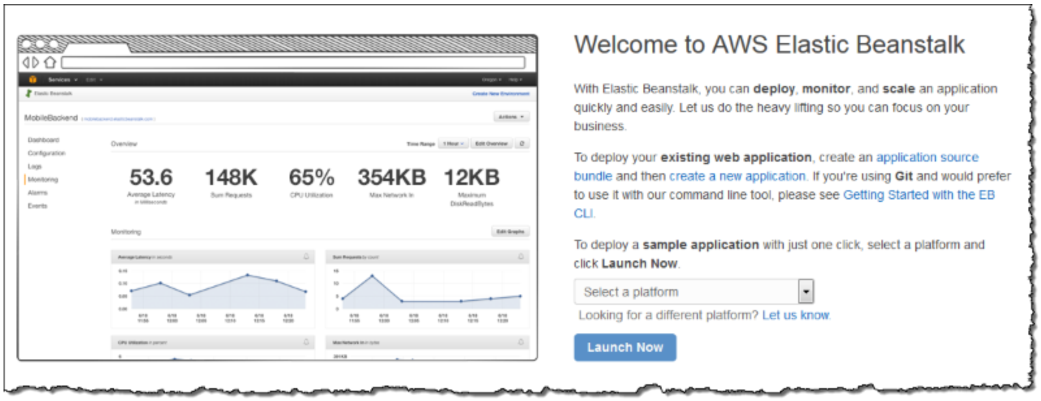
Step 2: Create an Application

Next, you will create and deploy a sample application. For this step, you use a sample application that is already prepared. If any Elastic Beanstalk applications already exist in the region in which you want to create and deploy an application, you must follow different procedures to create a new application. For more information, see Create an Application (p. 34).

> **Important**
> Elastic Beanstalk is free, but the AWS resources that it provides will be live (and not running in a sandbox). You will incur the standard usage fees for these resources until you terminate them in the last task in this tutorial. The total charges will be minimal (typically less than a dollar). For information on how you might minimize any charges, go to http://aws.amazon.com/free/.

To create a sample application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Select a platform and then click **Launch Now**.



Welcome to AWS Elastic Beanstalk

With Elastic Beanstalk, you can **deploy**, **monitor**, and **scale** an application quickly and easily. Let us do the heavy lifting so you can focus on your business.

To deploy your **existing web application**, create an application source bundle and then create a new application. If you're using **Git** and would prefer to use it with our command line tool, please see Getting Started with the EB CLI.

To deploy a **sample application** with just one click, select a platform and click **Launch Now**.

Select a platform ▾

Looking for a different platform? Let us know.

Launch Now

To run a sample application on AWS resources, Elastic Beanstalk takes the following actions, which can take several minutes to complete:

- Creates an Elastic Beanstalk application named **My First Elastic Beanstalk Application**.
- Launches an environment named **Default-Environment** that provisions the AWS resources to host the sample application.
- Creates a new application version named **Sample Application**, which refers to the default Elastic Beanstalk sample application file.
- Deploys the **Sample Application** application into the **Default-Environment**.

Step 3: View Information About Your Environment

After you create the Elastic Beanstalk application, you can view information about the application you deployed and its provisioned resources by going to the environment dashboard in the AWS Management Console. The dashboard shows the health of your application's environment, the running version, and the environment configuration.

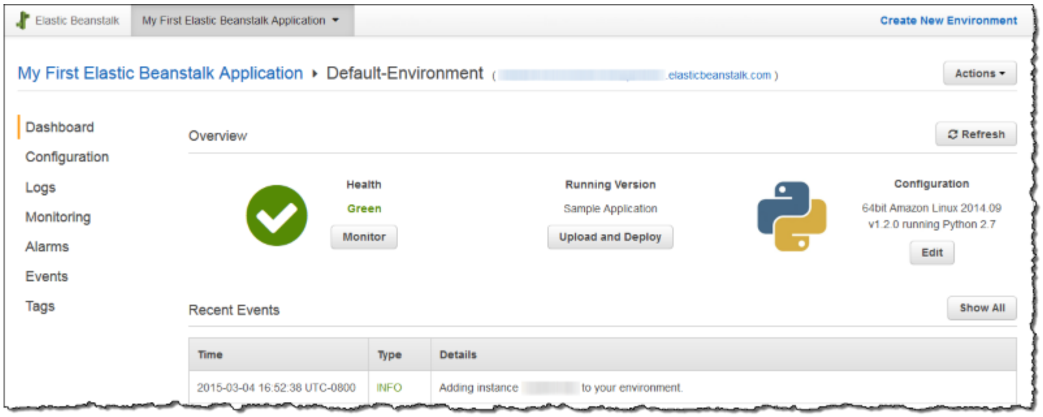
While Elastic Beanstalk creates your AWS resources and launches your application, the environment will be in a `Launching` (gray) state. Status messages about launch events are displayed in the environment's dashboard.

To see the environment dashboard for the My First Elastic Beanstalk Application application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk applications page, click **Default-Environment** in the **My First Elastic Beanstalk Application** application.

   From the dashboard you can view the status of the environment, the running application version, the platform, and a list of recent events.

   > **Note**
   > If the environment health is gray, the environment is still in the process of being launched.



Time	Type	Details
2015-03-04 16:52:38 UTC-0800	INFO	Adding instance [redacted] to your environment.

You can also view additional details about the environment by going to other pages from the dashboard:

- The **Configuration** page shows the resources provisioned for this environment, such as Amazon EC2 instances that host your application. This page also lets you configure some of the provisioned resources.
- The **Logs** page lets you view a snapshot of the last 100 lines of logs or review all logs for all your servers.
- The **Monitoring** page shows the statistics for the environment, such as average latency and CPU utilization. This page also lets you create alarms for the metrics that you are monitoring.
- The **Alarms** page shows the CloudWatch alarms you've created for this environment.
- The **Events** page shows any informational or error messages from services that this environment is using.
- The **Tags** page shows the metadata that you assigned in the form of tags to this environment. Each tag is represented on the page as a key-value pair. The page includes tags that Elastic Beanstalk automatically creates for the environment name and environment ID. For more information about tags, see Tag an Environment (p. 97).

Step 4: Deploy New Version

You can update your deployed application, even while it is part of a running environment. For a Java application, you can also use the AWS Toolkit for Eclipse to update your deployed application; for instructions, see Edit the Application and Redeploy (p. 583). For a PHP application, it is easy to update your application using a Git deployment via eb; for instructions, see Deploying Elastic Beanstalk Applications in PHP (p. 706). For a .NET application, you can use the AWS Toolkit for Visual Studio to update your deployed application; for instructions, see Edit the Application and Redeploy (p. 635).

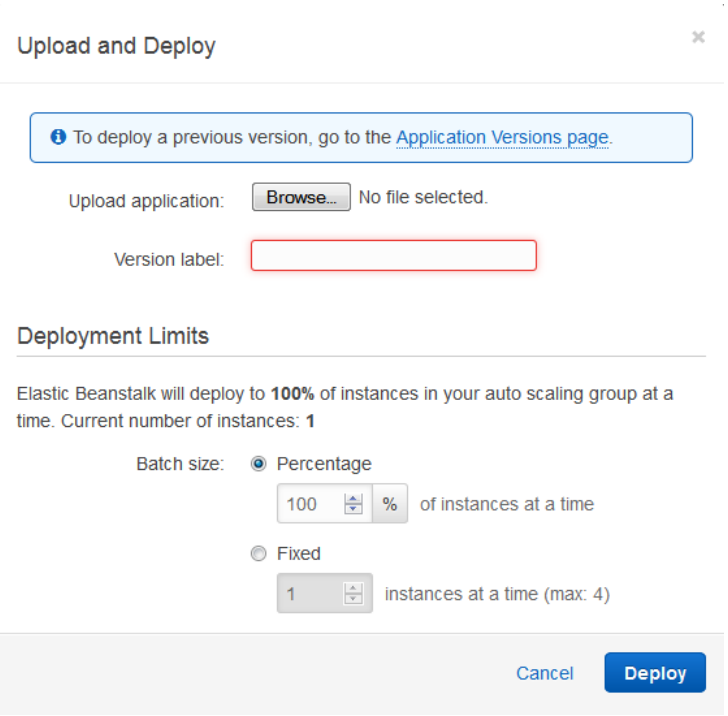
The application version you are running now is labeled **Sample Application**.

To update your application version

1. Download one of the following sample applications that match the configuration for your environment:

   - **Java with Tomcat**– elasticbeanstalk-sampleapp.war
   - **.NET on Windows Server with IIS**– FirstSample.zip
   - **Node.js**– nodejs-sample.zip
   - **PHP**– php-newsample-app.zip
   - **Python**– python-sample-20150402.zip
   - **Ruby (Passenger Standalone)**– ruby-sample.zip
   - **Ruby (Puma)**– ruby2PumaSampleApp.zip
   - **Single Container Docker** – docker-sample-v3.zip
   - **Preconfigured Docker (Glassfish)**– glassfish-sample.war
   - **Preconfigured Docker (Python 3.x)**– python3-sample.zip
   - **Preconfigured Docker (Go)**–golang-sample.zip

2. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
3. From the Elastic Beanstalk applications page, click **My First Elastic Beanstalk Application** and then click **Default-Environment**.
4. In the **Overview** section, click **Upload and Deploy** and then enter details about the application version.



Upload and Deploy

To deploy a previous version, go to the Application Versions page.

Upload application: Browse... No file selected.

Version label:

Deployment Limits

Elastic Beanstalk will deploy to **100%** of instances in your auto scaling group at a time. Current number of instances: **1**

Batch size: ● Percentage

100 % of instances at a time

○ Fixed

1 instances at a time (max: 4)

   - Use **Upload application** to locate and specify the application version (WAR or ZIP file) that you want to upload.
   - For **Version label**, enter a name for the application version that you are uploading, such as **`Sample Application Second Version`**.

5. Click **Deploy**.

Elastic Beanstalk now deploys your file to your Amazon EC2 instances. You can view the status of your deployment on the environment's dashboard. The **Environment Health** status turns gray while the application version is updated. When the deployment is complete, Elastic Beanstalk performs an application health check. The status returns to green when the application responds to the health check. The environment dashboard will show the new **Running Version** as **Sample Application Second Version** (or whatever you provided as the **Version label**.

Your new application version is also uploaded and added to the table of application versions. To view the table of application versions, click **My First Elastic Beanstalk Application** and then click **Application Versions**.

Step 5: Change Configuration

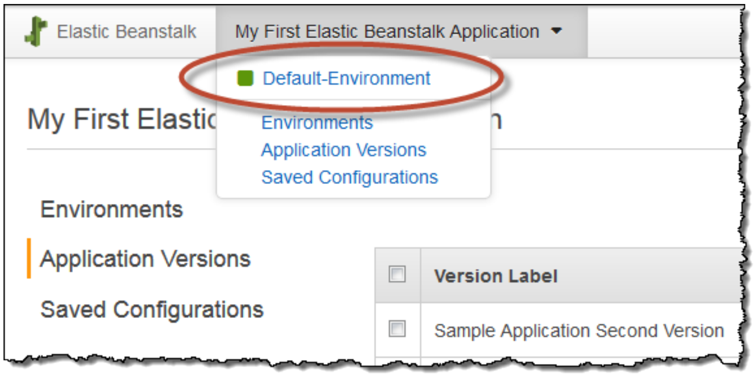
You can customize your environment to better suit your application. For example, if you have a compute-intensive application, you can change the type of Amazon EC2 instance that is running your application.

Some configuration changes are simple and happen quickly. Some changes require Elastic Beanstalk to delete and recreate AWS resources, which can take several minutes. Elastic Beanstalk will warn you about possible application downtime when changing configuration settings.

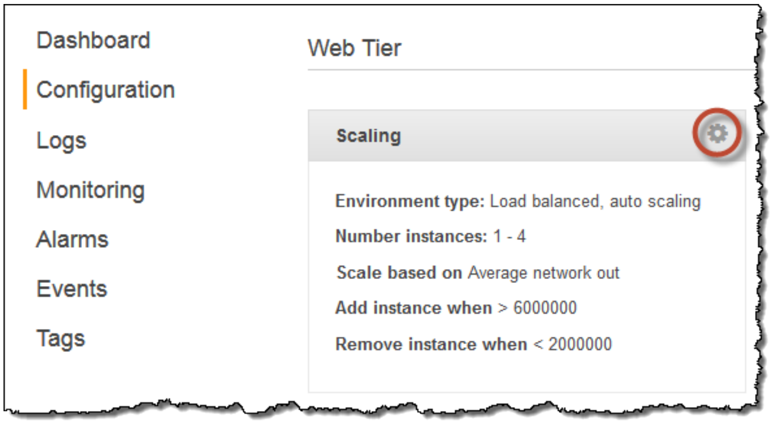
In this task, you change the minimum instance settings for your Auto Scaling group from one to two and then verify that the change occurred. After the new instance gets created, it will become associated with your load balancer.

To change your environment configuration

1. Go back to the environment dashboard by clicking **My First Elastic Beanstalk Application** and then **Default-Environment**.



2. In the navigation pane, click **Configuration**.
3. In the **Scaling** settings, click the gear icon (⚙).



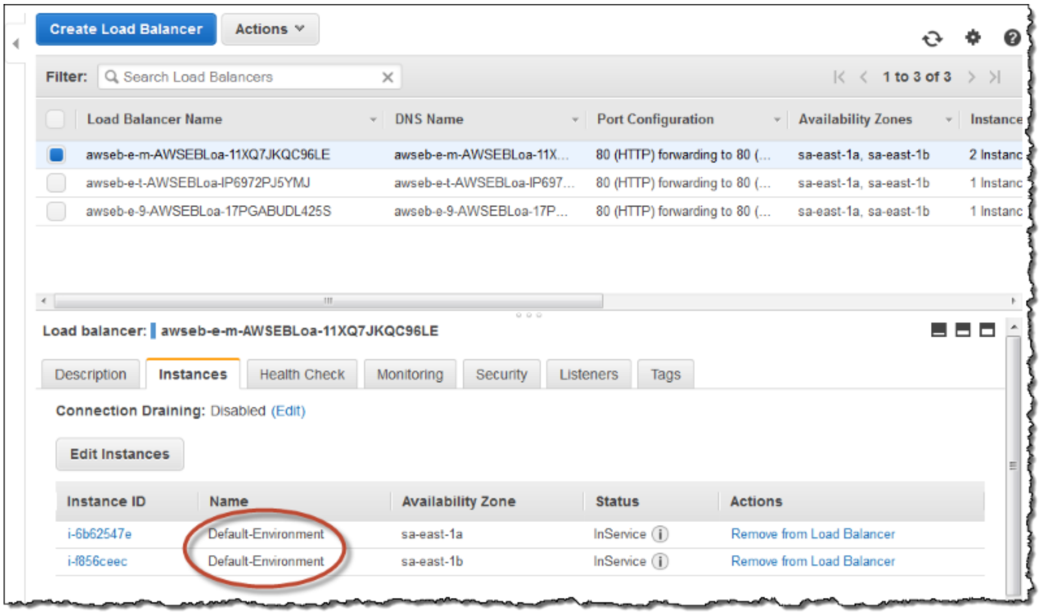
Environment type: Load balanced, auto scaling
Number instances: 1 - 4
Scale based on Average network out
Add instance when > 6000000
Remove instance when < 2000000

4. In the **Auto Scaling** section, change **Minimum Instance Count** from **1** to **2**. This increases the minimum number of Auto Scaling instances deployed in Amazon EC2.
5. At the bottom of the page, click **Save**.

The environment update might take a few minutes. When the environment is ready, you can go to the next task to verify your changes.

To verify changes to load balancers

1. In the navigation pane, click **Events**.
   You will see the event **Successfully deployed new configuration to environment** in the events list. This confirms that the Auto Scaling minimum instance count has been set to 2. A second instance is launched automatically.
2. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
3. In the navigation pane, under **NETWORK & SECURITY**, click **Load Balancers**.
4. Repeat the next two steps until you identify the load balancer with the desired instance name.
5. Click a load balancer in the list of load balancers.
6. Click the **Instances** tab in the **Load Balancer: *<load balancer name>*** pane, and then look at the **Name** in the **Instances** table.



Load Balancer Name	DNS Name	Port Configuration	Availability Zones	Instance
awseb-e-m-AWSEBLoa-11XQ7JKQC96LE	awseb-e-m-AWSEBLoa-11X...	80 (HTTP) forwarding to 80 (...	sa-east-1a, sa-east-1b	2 Instanc
awseb-e-t-AWSEBLoa-IP6972PJ5YMJ	awseb-e-t-AWSEBLoa-IP697...	80 (HTTP) forwarding to 80 (...	sa-east-1a, sa-east-1b	1 Instanc
awseb-e-9-AWSEBLoa-17PGABUDL425S	awseb-e-9-AWSEBLoa-17P...	80 (HTTP) forwarding to 80 (...	sa-east-1a, sa-east-1b	1 Instanc

Load balancer: awseb-e-m-AWSEBLoa-11XQ7JKQC96LE

Connection Draining: Disabled (Edit)

Instance ID	Name	Availability Zone	Status	Actions
i-6b62547e	Default-Environment	sa-east-1a	InService	Remove from Load Balancer
i-f856ceec	Default-Environment	sa-east-1b	InService	Remove from Load Balancer

The information shows that two instances are associated with this load balancer, corresponding to the increase in Auto Scaling instances.

Step 6: Clean Up

Congratulations! You have successfully deployed a sample application to the cloud, uploaded a new version, and modified its configuration to add a second Auto Scaling instance. To make sure you are not charged for any services you don't need, delete any unwanted applications and environments from Elastic Beanstalk and AWS services.

To completely delete the application

1. Delete all application versions.

   a. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
   b. From the Elastic Beanstalk applications page, click **Default-Environment** in the **My First Elastic Beanstalk Application** application.
   c. Click **Upload and Deploy**.
   d. When prompted for an application, click **Application Versions page**.
   e. On the **Application Versions** page, select all application versions that you want to delete, and then click **Delete**.
   f. Confirm the versions that you are deleting, and then click **Delete**.
   g. Click **Done**.

2. Terminate the environment.

   a. Go back to the environment dashboard by clicking **My First Elastic Beanstalk Application** and then **Default-Environment**.
   b. Click **Actions** and then click **Terminate Environment**.
   c. Confirm that you are terminating **`Default-Environment`** and then click **Terminate**.

3. Delete the `My First Elastic Beanstalk Application` Elastic Beanstalk application.

   a. Click **Elastic Beanstalk** at the upper left to return to the main dashboard.
   b. From the Elastic Beanstalk applications page, click **Actions** for the **My First Elastic Beanstalk Application** application and then click **Delete Application**.
   c. Confirm that you want to delete this Elastic Beanstalk application by clicking **Delete**.

Accessing Elastic Beanstalk

Topics

You can create an application using one of several different Elastic Beanstalk interfaces: the Elastic Beanstalk console in the AWS Management Console, Git deployment using AWS DevTools, the Elastic Beanstalk command line interface, the AWS Toolkits for Eclipse and Visual Studio, or programmatically through the AWS SDKs for Java, PHP, .NET, Node.js, Python, Ruby, or the Elastic Beanstalk web service API.

The simplest and quickest method for creating an application is to use the Elastic Beanstalk console. This does not require any additional software or tools. The console is a web browser interface that you can use to create and manage your Elastic Beanstalk applications.

The following sections contain overviews of each of the available Elastic Beanstalk interfaces. In order to use the Elastic Beanstalk features, you need to have an AWS account and be signed up for Elastic Beanstalk. For instructions on how to sign up for Elastic Beanstalk, see Step 1: Sign up for the Service (p. 4).

AWS Management Console

The AWS Management Console enables you to manage applications through Elastic Beanstalk from a single web browser interface. The console provides access to all of your deployed applications and gives you the ability to manage and monitor your applications and environments. From the console you can:

- Create and delete applications
- Add and delete application versions
- Create and delete environments
- Identify the running version within an environment

- View operational metrics
- View application and environment logs

The AWS Management Console is available at http://console.aws.amazon.com/elasticbeanstalk.

For more information about getting started with Elastic Beanstalk using the AWS Management Console, go to the Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

The EB CLI

The EB CLI is a command line tool for creating and managing environments. See Tools (p. 384) for details.

AWS SDK for Java

The AWS SDK for Java provides a Java API you can use to build applications that use AWS infrastructure services. With the AWS SDK for Java, you can get started in minutes with a single, downloadable package that includes the AWS Java library, code samples, and documentation.

The AWS SDK for Java requires J2SE Development Kit 5.0 or later. You can download the latest Java software from http://developers.sun.com/downloads/. The SDK also requires Apache Commons (Codec, HTTPClient, and Logging), and Saxon-HE third-party packages, which are included in the third-party directory of the SDK.

For more information on using the AWS SDK for Java, go to the AWS SDK for Java.

AWS Toolkit for Eclipse

With the AWS Toolkit for Eclipse plug-in, you can create new AWS Java web projects that are preconfigured with the AWS SDK for Java, and then deploy the web applications to Elastic Beanstalk. The Elastic Beanstalk plug-in builds on top of the Eclipse Web Tools Platform (WTP). The toolkit provides a Travel Log sample web application template that demonstrates the use of Amazon S3 and Amazon SNS.

To ensure you have all the WTP dependencies, we recommend that you start with the Java EE distribution of Eclipse, which you can download from http://eclipse.org/downloads/.

For more information on using the Elastic Beanstalk plug-in for Eclipse, go to the AWS Toolkit for Eclipse web page. To get started creating your Elastic Beanstalk application using Eclipse, see Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576).

AWS SDK for .NET

The AWS SDK for .NET enables you to build applications that use AWS infrastructure services. With the AWS SDK for .NET, you can get started in minutes with a single, downloadable package that includes the AWS .NET library, code samples, and documentation.

The AWS SDK for .NET requires .NET Framework 2.0 or later. It will work with any of the following Visual Studio editions:

- Microsoft Visual Studio Professional Edition 2010 and 2012 (recommended)
- Microsoft Visual C# 2008 Express Edition
- Microsoft Visual Web Developer 2008 Express Edition

For more information on using the AWS SDK for .NET, go to the AWS SDK for .NET web page.

AWS Toolkit for Visual Studio

With the AWS Toolkit for Visual Studio plug-in, you can deploy an existing .NET application to Elastic Beanstalk. You can also create new projects using the AWS templates that are preconfigured with the AWS SDK for .NET. For prerequisite and installation information, go to AWS Toolkit for Visual Studio. To get started creating your Elastic Beanstalk application using Visual Studio, see Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio (p. 612).

AWS SDK for Node.js

The AWS SDK for Node.js enables you to build applications on top of AWS infrastructure services. With the AWS SDK for Node.js, you can get started in minutes with a single, downloadable package that includes the AWS Node.js library, code samples, and documentation.

For more information on using the AWS SDK for Node.js, go to the AWS SDK for Node.js (Developer Preview) web page.

AWS SDK for PHP

The AWS SDK for PHP enables you to build applications on top of AWS infrastructure services. With the AWS SDK for PHP, you can get started in minutes with a single, downloadable package that includes the AWS PHP library, code samples, and documentation.

The AWS SDK for PHP requires PHP 5.2 or later.

For more information on using the AWS SDK for PHP, go to the AWS SDK for PHP web page.

Boto (AWS SDK for Python)

With Boto, you can get started in minutes with a single, downloadable package complete with the AWS Python library, code samples, and documentation. You can build Python applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides Python developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in Python for how to use the libraries to build applications. For information about Boto, sample code, documentation, tools, and additional resources, go to http://aws.amazon.com/python/.

AWS SDK for Ruby

You can get started in minutes with a single, downloadable package complete with the AWS Ruby library, code samples, and documentation. You can build Ruby applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides Ruby developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in Ruby for how to use the libraries to build applications. For information about the SDK, sample code, documentation, tools, and additional resources, go to http://aws.amazon.com/ruby/.

Elastic Beanstalk API

Elastic Beanstalk provides a comprehensive API that enables you to programmatically access Elastic Beanstalk functionality. For more information, go to the Elastic Beanstalk API Reference.

Endpoints

For information about this product's regions and endpoints, go to Regions and Endpoints in the *Amazon Web Services General Reference*.

Where to Go Next

Now that you have learned about Elastic Beanstalk and how to access it, we recommend that you read How Does Elastic Beanstalk Work? (p. 15) This topic provides information about the Elastic Beanstalk components, the architecture, and important design considerations for your Elastic Beanstalk application.

How Does Elastic Beanstalk Work?

Topics

Now that you have a better understanding of what Elastic Beanstalk is, let's take a peek under the hood and see how Elastic Beanstalk works. The following sections discuss the Elastic Beanstalk components, the architecture, and important design considerations for your Elastic Beanstalk application.

Elastic Beanstalk Components

The components that comprise Elastic Beanstalk work together to enable you to easily deploy and manage your application in the cloud. This section discusses those components.

Application

An Elastic Beanstalk *application* is a logical collection of Elastic Beanstalk components, including *environments*, *versions*, and *environment configurations*. In Elastic Beanstalk an application is conceptually similar to a folder.

Application Version

In Elastic Beanstalk, an *application version* refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon Simple Storage Service (Amazon S3) object that contains the deployable code such as a Java WAR file. An application version is part of an application. Applications can have many versions and each application version is unique. In a running environment, you can deploy any application version you already uploaded to the application or you can upload and immediately deploy a new application version. You might upload multiple application versions to test differences between one version of your web application and another.

Environment

An *environment* is a version that is deployed onto AWS resources. Each environment runs only a single application version at a time, however you can run the same version or different versions in many environments at the same time. When you create an environment, Elastic Beanstalk provisions the resources needed to run the application version you specified. For more information about the environment and the resources that are created, see Architectural Overview (p. 16).

Environment Configuration

An *environment configuration* identifies a collection of parameters and settings that define how an environment and its associated resources behave. When you update an environment's configuration settings, Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources (depending on the type of change).

Configuration Template

A *configuration template* is a starting point for creating unique environment configurations. Configuration templates can be created or modified by using the Elastic Beanstalk command line utilities or API.

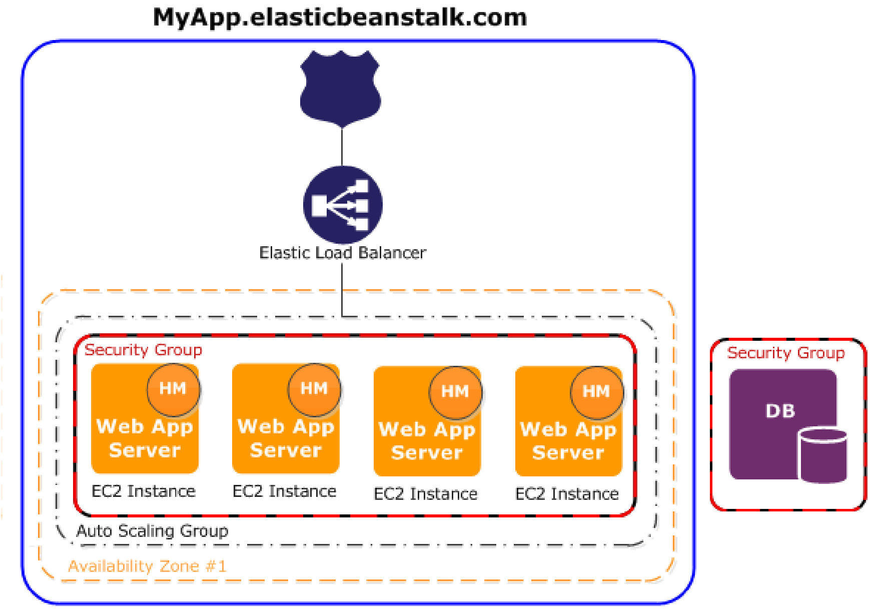
Architectural Overview

When you launch an Elastic Beanstalk environment, you choose an environment tier, platform, and environment type. The environment tier that you choose determines whether Elastic Beanstalk provisions resources to support a web application that handles HTTP(S) requests or a web application that handles background-processing tasks. An environment tier whose web application processes web requests is known as a *web server tier*. An environment tier whose web application runs background jobs is known as a *worker tier*. This topic describes the components, resources, and architecture for each type of environment tier.

> **Note**
> One environment cannot support two different environment tiers because each requires its own set of resources; a worker environment tier and a web server environment tier each require an Auto Scaling group, but Elastic Beanstalk supports only one Auto Scaling group per environment.

Web Server Environment Tiers

This following diagram illustrates an example Elastic Beanstalk architecture for a web server environment tier and shows how the components in that type of environment tier work together. The remainder of this section discusses all the components in more detail.



The environment is the heart of the application. In the diagram, the environment is delineated by the solid blue line. When you create an environment, Elastic Beanstalk provisions the resources required to run your application. AWS resources created for an environment include one elastic load balancer (ELB in the diagram), an Auto Scaling group, and one or more Amazon EC2 instances.

Every environment has a CNAME (URL) that points to a load balancer. The environment has a URL such as MyApp.elasticbeanstalk.com. This URL is aliased in Amazon Route 53 to an Elastic Load Balancing URL—something like abcdef-123456.us-west-2.elb.amazonaws.com—by using a CNAME record. Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. It provides secure and reliable routing to your infrastructure. Your domain name that you registered with your DNS provider will forward requests to the CNAME. The load balancer sits in front of the Amazon EC2 instances, which are part of an Auto Scaling group. (The Auto Scaling group is delineated in the diagram by a broken black line.) Auto Scaling automatically starts additional Amazon EC2 instances to accommodate increasing load on your application. If the load on your application decreases, Auto Scaling stops instances, but always leaves at least one instance running. For information about how to map your root domain to your Elastic Load Balancer, see Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295).

The software stack running on the Amazon EC2 instances is dependent on the *container type*. A container type defines the infrastructure topology and software stack to be used for that environment. For example, an Elastic Beanstalk environment with an Apache Tomcat container uses the Amazon Linux operating system, Apache web server, and Apache Tomcat software. For a list of supported container types, see Supported Platforms (p. 24). Each Amazon EC2 server instance that runs your application uses one of these container types. In addition, a software component called the *host manager (HM)* runs on each Amazon EC2 server instance. (In the diagram, the HM is an orange circle in each EC2 instance.) The host manager is responsible for:

- Deploying the application
- Aggregating events and metrics for retrieval via the console, the API, or the command line
- Generating instance-level events
- Monitoring the application log files for critical errors
- Monitoring the application server

- Patching instance components
- Rotating your application's log files and publishing them to Amazon S3

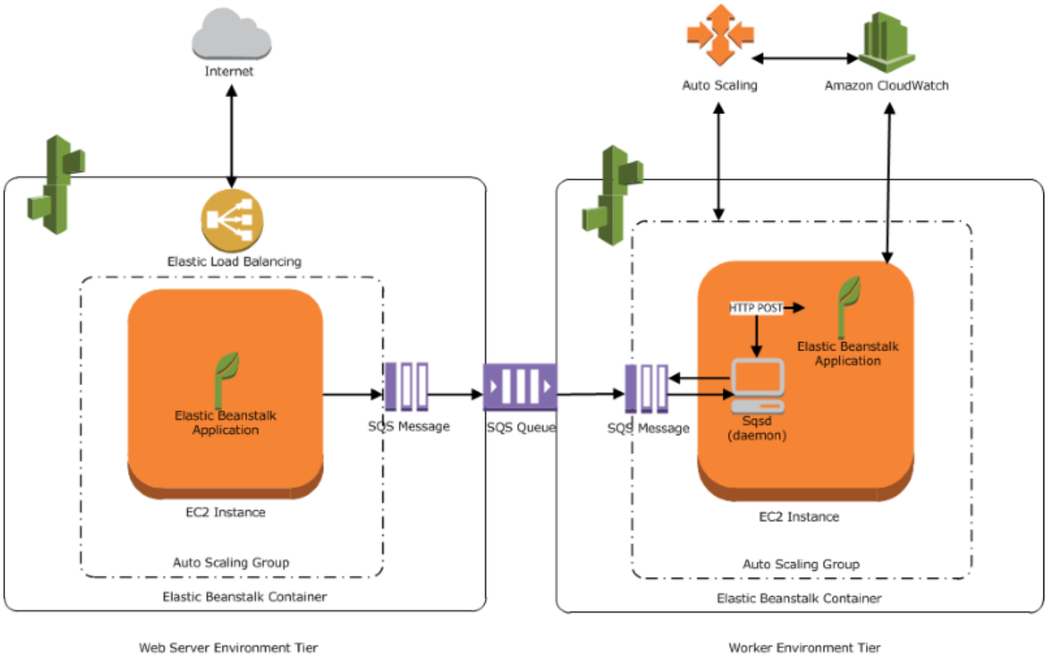
The host manager reports metrics, errors and events, and server instance status, which are available via the AWS Management Console, APIs, and CLIs.

The Amazon EC2 instances shown in the diagram are part of one security group. A security group defines the firewall rules for your instances. By default, Elastic Beanstalk defines a security group, which allows everyone to connect using port 80 (HTTP). You can define more than one security group. For instance, you can define a security group for your database server. For more information about Amazon EC2 security groups and how to configure them for your Elastic Beanstalk application, see the Amazon EC2 Security Groups (p. 201).

Worker Environment Tiers

AWS resources created for a worker environment tier include an Auto Scaling group, one or more Amazon EC2 instances, and an IAM role. For the worker environment tier, Elastic Beanstalk also creates and provisions an Amazon SQS queue if you don't already have one. When you launch a worker environment tier, Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each EC2 instance in the Auto Scaling group. The daemon is responsible for pulling requests from an Amazon SQS queue and then sending the data to the web application running in the worker environment tier that will process those messages. If you have multiple instances in your worker environment tier, each instance has its own daemon, but they all read from the same Amazon SQS queue.

The following diagram shows the different components and their interactions across environments and AWS services.



Amazon CloudWatch is used for alarms and health monitoring. For more information, go to Basic Health Reporting (p. 247).

For details about how the worker environment tier works, see How a Worker Environment Works (p. 69).

Service Roles, Instance Profiles, and User Policies

When you create a new environment, AWS Elastic Beanstalk prompts you to provide two AWS Identity and Access Management (IAM) roles, a service role and an instance profile. The Elastic Beanstalk service role grants the service access to use other AWS services on your behalf. The Elastic Beanstalk instance profile grants the instances in your environment permission to call AWS services for features including enhanced health reporting (all Linux-based platforms) and container management (multicontainer Docker configuration).

In addition to the two roles that you assign to your environment, you can also create user policies and apply them to IAM users and groups in your account to allow those users and groups to use Elastic Beanstalk without granting them access to other AWS services.

For most environments, the service role and instance profile that the AWS Management Console prompts you to create with 1-click role creation when you launch your environment have all of the permissions that you need. Likewise, the managed policies (p. 326) provided by Elastic Beanstalk for full access and read-only access contain all of the user permissions required for daily use. See the IAM user guide for in-depth coverage of AWS permissions.

Topics

Elastic Beanstalk Service Role

A service role is the IAM role that Elastic Beanstalk assumes when calling other services on your behalf. Elastic Beanstalk uses the service role that you specify when creating an Elastic Beanstalk environment when it calls Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing, and Auto Scaling APIs to gather information about the health of its AWS resources.

The following statement contains all of the permissions that Elastic Beanstalk needs to monitor environment health:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeInstanceHealth",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:GetConsoleOutput",
        "ec2:AssociateAddress",
        "ec2:DescribeAddresses",
        "ec2:DescribeSecurityGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeScalingActivities",
    "autoscaling:DescribeNotificationConfigurations"
   ],
   "Resource": [
    "*"
   ]
  }
 ]
}
```

This policy allows also includes Amazon SQS actions to allow Elastic Beanstalk to monitor queue activity for worker environments.

When you create an environment with the Elastic Beanstalk console, Elastic Beanstalk prompts you to create a service role named `aws-elasticbeanstalk-service-role`with the permissions listed in the preceding statement and a trust policy that allows Elastic Beanstalk to assume the service role. If you create the service role yourself with IAM, you also must add the trust policy.

The following trust policy, when attached to a role with the permissions listed in the preceding statement, allows Elastic Beanstalk to assume the service role:

```
{
  "Version": "2012-10-17",
  "Statement": [
   {
    "Sid": "",
    "Effect": "Allow",
    "Principal": {
     "Service": "elasticbeanstalk.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
     "StringEquals": {
      "sts:ExternalId": "elasticbeanstalk"
     }
    }
   }
  ]
}
```

If you use the Elastic Beanstalk API or EB CLI (p. 384) to create an environment, specify a service role with the `ServiceRole` configuration option in the `aws:elasticbeanstalk:environment` namespace. See Using Enhanced Health Reporting with the AWS Elastic Beanstalk API (p. 268) for details.

Elastic Beanstalk Instance Profile

An instance profile is an IAM role that is applied to instances launched in your Elastic Beanstalk environment. When creating an Elastic Beanstalk environment, you specify the instance profile that is used when your instances:

- Write logs to Amazon Simple Storage Service
- In multicontainer Docker environments, coordinate container deployments with Amazon EC2 Container Service
- In worker environments, read from an Amazon Simple Queue Service (Amazon SQS) queue
- In worker environments, perform leader election with Amazon DynamoDB
- In worker environments, publish instance health metrics to Amazon CloudWatch

The following policy contains statements that allow instances in your environment to perform each of the above actions. You can add all of these statements to a single role and use it for all of your environments, or separate permissions into roles for web server, worker tier, and multicontainer Docker environments.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Sid": "QueueAccess",
        "Action": [
            "sqs:ChangeMessageVisibility",
            "sqs:DeleteMessage",
            "sqs:ReceiveMessage",
            "sqs:SendMessage"
        ],
        "Effect": "Allow",
        "Resource": "*"
    }, {
        "Sid": "MetricsAccess",
        "Action": [
            "cloudwatch:PutMetricData"
        ],
        "Effect": "Allow",
        "Resource": "*"
    }, {
        "Sid": "BucketAccess",
        "Action": [
            "s3:Get*",
            "s3:List*",
            "s3:PutObject"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:s3:::elasticbeanstalk-*-{{accountid}}/*",
            "arn:aws:s3:::elasticbeanstalk-*-{{accountid}}-*/*"
        ]
    }, {
        "Sid": "DynamoPeriodicTasks",
        "Action": [
            "dynamodb:BatchGetItem",
            "dynamodb:BatchWriteItem",
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Query",
            "dynamodb:Scan",
            "dynamodb:UpdateItem"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:dynamodb:*:{{accountid}}:table/*-stack-AWSEBWorkerCronLead
erRegistry*"
        ]
    }, {
        "Sid": "ECSAccess",
        "Effect": "Allow",
        "Action": [
            "ecs:StartTask",
            "ecs:StopTask",
            "ecs:RegisterContainerInstance",
            "ecs:DeregisterContainerInstance",
            "ecs:DiscoverPollEndpoint",
            "ecs:Submit*",
            "ecs:Poll"
        ],
        "Resource": "*"
    }
  ]
}
```

Elastic Beanstalk User Policy

Elastic Beanstalk requires permissions not only for its own API actions, but for several other AWS services as well. Elastic Beanstalk uses user permissions to launch all of the resources in an environment, including EC2 instances, an Elastic Load Balancing load balancer, and an Auto Scaling group. Elastic Beanstalk also uses user permissions to save logs and templates to Amazon S3, send notifications to Amazon SNS, assign instance profiles, and publish metrics to CloudWatch. Elastic Beanstalk requires AWS CloudFormation permissions to orchestrate resource deployments and updates. It also requires Amazon RDS permissions to create databases when needed, and Amazon SQS permissions to create queues for worker environments.

The following policy allows access to the actions used to create and manage Elastic Beanstalk environments. This policy is available in the IAM console as a managed policy named `AWSElasticBeanstalkFullAccess`. You can apply the managed policy to an IAM user or group to grant permission to use Elastic Beanstalk, or create your own policy that excludes permissions that are not needed by your users.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "iam:PassRole",
        "iam:ListRoles",
        "iam:ListInstanceProfiles"
      ],
      "Resource": "*"
    }
  ]
}
```

The above permissions allow a user to assign an existing service role to an environment, but do not allow the user to create a role. You can allow an IAM user to manage roles by adding the following permissions to the above policy, but be aware that if you grant a user permission to create roles, that user can create a role that has unrestricted permissions.

```
iam:CreateRole,
iam:PutRolePolicy,
iam:ListServerCertificates,
iam:CreateInstanceProfile,
iam:AddRoleToInstanceProfile
```

To avoid the need to give users permission to create roles, create a new environment (p. 50) and generate the default service role when prompted.

Elastic Beanstalk also provides a read-only managed policy named `AWSElasticBeanstalkReadOnlyAccess`. This policy allows a user to view, but not modify or create, Elastic Beanstalk environments.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*",
        "rds:Describe*",
        "sqs:Get*",
        "sqs:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

Supported Platforms

Elastic Beanstalk has web server platforms that support applications developed for Docker, Java, .NET, Node.js, PHP, Python, and Ruby with multiple configurations available for most platforms. Worker environments are supported for all platforms except .NET.

Elastic Beanstalk provisions the resources needed to run your application including one or more Amazon EC2 instances. The software stack running on the Amazon EC2 instances depends on the configuration. In a configuration name, the version number refers to the configuration version.

The solution stack name listed beneath the configuration name can be used to launch an environment with the EB CLI (p. 384), Elastic Beanstalk API, or AWS CLI (p. 523). Solution stack names can also be retrieved from the service with the ListAvailableSolutionStacks API (aws elasticbeanstalk list-available-solution-stacks in the AWS CLI). This operation returns all of the solution stacks that you can use to create an environment, including the latest stacks (listed on this page), and any previous versions that you have used in the past.

All current Linux based platform configurations run on Amazon Linux 2015.03 (64-bit). Detailed release notes are available for recent releases at aws.amazon.com/releasenotes.

Docker

You can use Docker containers to deploy applications to Elastic Beanstalk. They are included in the list of predefined configurations (or solution stacks) that you can choose when you create an environment. Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Docker Version	Web Server
Single Container Docker 1.6 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Docker 1.6.2*	2015.03	1.6.2	Nginx 1.6.2
Multicontainer Docker 1.6 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Multi-container Docker 1.6.2 (Generic)*	2015.03	1.6.2	none

For information on previous configurations, see Docker Platform History (p. 802).

Preconfigured Docker

You can deploy applications to Elastic Beanstalk with Docker containers that are preconfigured to support applications that were developed using a specific language platform. They are included in the list of predefined configurations (or solution stacks) as **Docker - Preconfigured** configurations that you can choose when you create an environment. Elastic Beanstalk supports the following preconfigured Docker configurations:

Configuration and *Solution Stack Name*	AMI	Platform	Container OS	[illegible]
Glassfish 4.1 (Docker) version 2.0.0 *64bit Debian jessie v2.0.0 running GlassFish 4.1 Java 8 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Glassfish 4.0 (Docker) version 2.0.0 *64bit Debian jessie v2.0.0 running GlassFish 4.0 Java 7 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.4 (Docker) version 2.0.0 *64bit Debian jessie v2.0.0 running Go 1.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.3 (Docker) version 2.0.0 *64bit Debian jessie v2.0.0 running Go 1.3 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Python 3.4 with uWSGI 2 (Docker) version 2.0.0 *64bit Debian jessie v2.0.0 running Python 3.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]

For information on previous configurations, see Preconfigured Docker Platform History (p. 807).

Java with Tomcat

You can get started with Java using the AWS Toolkit for Eclipse. The toolkit is a downloadable package that includes the AWS libraries, project templates, code samples, and documentation. The AWS SDK for

Java supports developing applications using either Java 5 or Java 6. Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Java 8 with Tomcat 8 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Tomcat 8 Java 8*	2015.03	Java 1.8.0_51	Tomcat 8.0.20	Apache 2.2.29
Java 7 with Tomcat 7 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Tomcat 7 Java 7*	2015.03	Java 1.7.0_85	Tomcat 7.0.62	Apache 2.2.29
Java 6 with Tomcat 7 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Tomcat 7 Java 6*	2015.03	Java 1.6.0_35	Tomcat 7.0.62	Apache 2.2.29

For information on previous configurations, see Java Platform History (p. 816).

.NET on Windows Server with IIS

You can get started in minutes using the AWS Toolkit for Visual Studio. The toolkit includes the AWS libraries, project templates, code samples, and documentation. The AWS SDK for .NET supports the development of applications using .NET Framework 2.0 or later.

> **Note**
> This platform does not support worker environments or enhanced health reporting.

To learn how to get started deploying a .NET application using the AWS Toolkit for Visual Studio, see Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio (p. 612). Elastic Beanstalk supports the following configurations

Configuration and *Solution Stack Name*	Framework	Web Server
Windows Server 2012 R2[1] with IIS 8.5 *64bit Windows Server 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012 R2[1] Server Core with IIS 8.5 *64bit Windows Server Core 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012[1] with IIS 8 *64bit Windows Server 2012 running IIS 8*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8

Configuration and *Solution Stack Name*	Framework	Web Server
Windows Server 2008 R2[1] with IIS 7.5 *64bit Windows Server 2008 R2 running IIS 7.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin Summary for August 2015

For information on previous configurations, see .NET on Windows Server with IIS Platform History (p. 825).

Node.js

You can get started in minutes using the AWS Management Console or the eb command line interface. You can deploy applications by simply zipping them up and uploading them through the console. To learn how to upload an application using the AWS Management Console, see Create an Application (p. 34). To learn how to get started deploying a Node.js application to Elastic Beanstalk using eb and Git, see Deploying Node.js Applications to AWS Elastic Beanstalk (p. 666). Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Platform	Web Server	Git
Node.js version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Node.js*	2015.03	Node.js 0.12.6 Also supports 0.10.39, 0.10.38, 0.10.31, 0.8.28	Nginx 1.6.2 or Apache 2.4.12	Git 2.1.0

For information on previous configurations, see Node.js Platform History (p. 827).

> **Note**
> When support for the version of Node.js that you are using is removed from the platform configuration, you must change the version setting prior to doing a platform upgrade (p. 90). This may occur when a security vulnerability is identified for one or more versions of Node.js When this occurs, attempting to upgrade to a new version of the platform that does not support the configured NodeVersion (p. 123) will fail. To avoid needing to create a new environment, change the *NodeVersion* configuration option to a version that is supported by both the old configuration version and the new one, and then perform the platform upgrade.

PHP

You can get started in minutes using the AWS Management Console or the eb command line interface. You can deploy applications by simply zipping them up and uploading them through the console. To learn how to upload an application using the AWS Management Console, see Create an Application (p. 34). To learn how to get started deploying a PHP application to Elastic Beanstalk using eb and Git, see Deploying Elastic Beanstalk Applications in PHP (p. 706). Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Language	Composer	b eW -vreS r e
PHP 5.6 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running PHP 5.6*	2015.03	PHP 5.6.9	1.0.0-alpha10	ehcapA 21.4.2
PHP 5.5 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running PHP 5.5*	2015.03	PHP 5.5.25	1.0.0-alpha10	ehcapA 21.4.2
PHP 5.4 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running PHP 5.4*	2015.03	PHP 5.4.41	1.0.0-alpha10	ehcapA 21.4.2

For information on previous configurations, see PHP Platform History (p. 837).

Python

Elastic Beanstalk supports Python applications running on Apache and WSGI. This includes support for many popular frameworks such as Django and Flask. You can get started in minutes using the AWS Management Console or the eb command line interface. You can deploy applications by simply zipping them up and uploading them through the console. To learn how to upload an application using the AWS Management Console, see Create an Application (p. 34). To learn how to get started deploying a Python application to Elastic Beanstalk using eb and Git, see Working with Python (p. 733). Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Language	Web Server
Python 3.4 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Python 3.4*	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
Python 2.7 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Python 2.7*	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5
Python 2.6 version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Python*	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

For information on previous configurations, see Python Platform History (p. 845).

Ruby

Elastic Beanstalk runs Ruby applications including support for many popular frameworks such as Rails and Sinatra. You can deploy applications in minutes using the eb command line interface and Git or the AWS Management Console. To learn how to get started deploying a Ruby application to Elastic Beanstalk using eb and Git, see Deploying Elastic Beanstalk Applications in Ruby Using EB CLI and Git (p. 771). To learn how to upload an application using the AWS Management Console, see Create an Application (p. 34). Elastic Beanstalk supports the following configurations:

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Ruby 2.2 with Puma version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.2 (Puma)*	2015.03	Ruby 2.2.2	Puma 2.10.2	Nginx 1.6.2
Ruby 2.2 with Passenger version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.2 (Passenger Standalone)*	2015.03	Ruby 2.2.2	Passenger 4.0.59	Nginx 1.6.2
Ruby 2.1 with Puma version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.1 (Puma)*	2015.03	Ruby 2.1.5-p273	Puma 2.10.2	Nginx 1.6.2
Ruby 2.1 with Passenger version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.1 (Passenger Standalone)*	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59	Nginx 1.6.2
Ruby 2.0 with Puma version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.0 (Puma)*	2015.03	Ruby 2.0.0-p598	Puma 2.10.2	Nginx 1.6.2
Ruby 2.0 with Passenger version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 2.0 (Passenger Standalone)*	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59	Nginx 1.6.2
Ruby 1.9 with Passenger version 2.0.0 *64bit Amazon Linux 2015.03 v2.0.0 running Ruby 1.9.3*	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59	Nginx 1.6.2

For information on previous configurations, see Ruby Platform History (p. 850).

Design Considerations

Because applications deployed using Elastic Beanstalk run on Amazon cloud resources, you should keep several things in mind when designing your application: *scalability*, *security*, *persistent storage*, *fault tolerance*, *content delivery*, *software updates and patching*, and *connectivity*. For a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security and economics, go to AWS Cloud Computing Whitepapers.

Scalability

When you're operating in a physical hardware environment, as opposed to a cloud environment, you can approach scalability two ways—you can scale up (vertical scaling) or scale out (horizontal scaling). The scale-up approach requires an investment in powerful hardware as the demands on the business increase, whereas the scale-out approach requires following a distributed model of investment, so hardware and application acquisitions are more targeted, data sets are federated, and design is service-oriented. The scale-up approach could become very expensive, and there's still the risk that demand could outgrow capacity. Although the scale-out approach is usually more effective, it requires predicting the demand at regular intervals and deploying infrastructure in chunks to meet demand. This approach often leads to unused capacity and requires careful monitoring.

By moving to the cloud you can bring the use of your infrastructure into close alignment with demand by leveraging the elasticity of the cloud. Elasticity is the streamlining of resource acquisition and release, so that your infrastructure can rapidly scale in and scale out as demand fluctuates. To implement elasticity, configure your Auto Scaling settings to scale up or down based on metrics from the resources in your environment (utilization of the servers or network I/O, for instance). You can use Auto Scaling to automatically add compute capacity when usage rises and remove it when usage drops. Publish system metrics (CPU, memory, disk I/O, network I/O) to Amazon CloudWatch and configure alarms to trigger Auto Scaling actions or send notifications. For more instructions on configuring Auto Scaling, see Configuring Auto Scaling with Elastic Beanstalk (p. 177).

Elastic Beanstalk applications should also be as *stateless* as possible, using loosely coupled, fault-tolerant components that can be scaled out as needed. For more information about designing scalable application architectures for AWS, read the Architecting for the Cloud: Best Practices whitepaper.

Security

Security on AWS is a shared responsibility. AWS protects the physical resources in your environment and ensure that the cloud is a safe place for you to run applications. You are responsible for the security of data coming in and out of your Elastic Beanstalk environment and the security of your application.

Configure SSL to protect information from your clients. You will need a certificate from an external certification authority such as VeriSign or Entrust. The public key included in the certificate authenticates your server to the browser and serves as the basis for creating the shared session key used to encrypt the data in both directions. For instructions on creating, uploading, and assigning an SSL certificate to your environment, see Configuring HTTPS for your Elastic Beanstalk Environment (p. 228).

When you configure an SSL certificate for your environment, data is encrypted between the client and your environment's Elastic Load Balancing load balancer. By default, encryption is terminated at the load balancer and traffic between the load balancer and Amazon EC2 instances is unencrypted.

Persistent Storage

Elastic Beanstalk applications run on Amazon EC2 instances that have no persistent local storage. When the Amazon EC2 instances terminate, the local file system is not saved, and new Amazon EC2 instances start with a default file system. You should design your application to store data in a persistent data source. Amazon Web Services offers a number of persistent storage options that you can leverage for your application, including:

- Amazon Simple Storage Service (Amazon S3). For more information about Amazon S3, go to the documentation.
- Amazon Elastic Block Store (Amazon EBS). For more information, go to the documentation and see also the article Feature Guide: Elastic Block Store.
- Amazon DynamoDB. For more information, go to the documentation. For an example using Amazon DynamoDB with Elastic Beanstalk, see Example: DynamoDB, CloudWatch, and SNS (p. 138).
- Amazon Relational Database Service (Amazon RDS). For more information, go to the documentation and see also Amazon RDS for C# Developers.

Fault Tolerance

As a rule of thumb, you should be a pessimist when designing architecture for the cloud. Always design, implement, and deploy for automated recovery from failure. Use multiple Availability Zones for your Amazon EC2 instances and for Amazon RDS. Availability Zones are conceptually like logical data centers. Use Amazon CloudWatch to get more visibility into the health of your Elastic Beanstalk application and take appropriate actions in case of hardware failure or performance degradation. Configure your Auto Scaling settings to maintain your fleet of Amazon EC2 instances at a fixed size so that unhealthy Amazon EC2 instances are replaced by new ones. If you are using Amazon RDS, then set the retention period for backups, so that Amazon RDS can perform automated backups.

Content Delivery

When users connect to your website, their requests may be routed through a number of individual networks. As a result users may experience poor performance due to high latency. Amazon CloudFront can help ameliorate latency issues by distributing your web content (such as images, video, and so on) across a network of edge locations around the world. End users are routed to the nearest edge location, so content is delivered with the best possible performance. CloudFront works seamlessly with Amazon S3, which durably stores the original, definitive versions of your files. For more information about Amazon CloudFront, see http://aws.amazon.com/cloudfront.

Software Updates and Patching

Elastic Beanstalk periodically updates its platform configurations with new software and patches. Elastic Beanstalk does not upgrade running environments to new configuration versions automatically, but you can initiate a platform upgrade (p. 90) to update your running environment in place. Platform upgardes use rolling updates (p. 169) to keep your application available by applying changes in batches.

Connectivity

Elastic Beanstalk needs to be able to connect to the instances in your environment to complete deployments. When you deploy an Elastic Beanstalk application inside an Amazon VPC, the configuration required to enable connectivity depends on the type of Amazon VPC environment you create:

- For single-instance environments, no additional configuration is required because Elastic Beanstalk assigns each Amazon EC2 instance a public Elastic IP address that enables the instance to communicate directly with the Internet.
- For load-balancing, autoscaling environments in an Amazon VPC with both public and private subnets, you must do the following:
  - Create a load balancer in the public subnet to route inbound traffic from the Internet to the Amazon EC2 instances.
  - Create a network address translation (NAT) instance to route outbound traffic from the Amazon EC2 instances to the Internet.
  - Create inbound and outbound routing rules for the Amazon EC2 instances inside the private subnet.
  - Configure the default Amazon VPC security group to allow traffic from the Amazon EC2 instances to the NAT instance.
- For a load-balancing, autoscaling environment in an Amazon VPC that has one public subnet, no additional configuration is required because the Amazon EC2 instances are configured with a public IP address that enables the instances to communicate with the Internet.

For more information about using Elastic Beanstalk with Amazon VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).

Where to Go Next

Now that you have learned some Elastic Beanstalk basics, you are ready to start creating and deploying your applications. If you are a developer, go to one of the following sections:

- **Java** — Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576)
- **.NET** — Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio (p. 612)
- **Node.js** — Deploying Node.js Applications to AWS Elastic Beanstalk (p. 666)
- **PHP** — Deploying Elastic Beanstalk Applications in PHP (p. 706)
- **Python** — Working with Python (p. 733)
- **Ruby** — Deploying Elastic Beanstalk Applications in Ruby Using EB CLI and Git (p. 771)

If you want to manage and configure your applications and environments using the AWS Management Console, command line interface, or APIs, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

Tutorials

Language and framework specific tutorials are spread throughought the AWS Elastic Beanstalk Developer Guide. New and updated tutorials are added to this list as they are published. The most recent updates are shown first.

These tutorials are targeted at intermediate users and may not contain instructions for basic steps such as signing up for AWS. If this is your first time using AWS or Elastic Beanstalk, check out the Getting Started walkthrough (p. 4) to get your first Elastic Beanstalk environment up and running from scratch.

- **Python and Flask** - Deploying a Flask Application (p. 743)
- **Python and Django** - Deploying a Django Application (p. 750)
- **Docker, PHP and Nginx** - Multicontainer Docker Environments with the AWS Management Console (p. 551)
- **Ruby on Rails** - Deploying a Rails Application to Elastic Beanstalk (p. 771)
- **Ruby and Sinatra** - Deploying a Sinatra Application to AWS Elastic Beanstalk (p. 779)
- **.NET Framework (IIS and ASP.NET)** - Tutorial: How to Deploy a .NET Sample Application Using AWS Elastic Beanstalk (p. 613)

Managing and Configuring Applications and Environments Using the Console, CLI, and APIs

Topics

This topic discusses some of the most important features of Elastic Beanstalk in detail, including usage examples using the AWS Management Console, CLI, and the APIs. For more information about the CLI, see EB Command Line Interface (p. 384). For more information about the API, go to AWS Elastic Beanstalk API Reference.

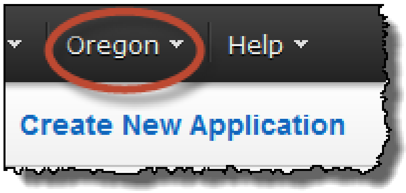
Create an Application

You can use the AWS Management Console, the command line interface (CLI), or the API to create a new Elastic Beanstalk application and deploy the application version to a new environment.

AWS Management Console

To create a new application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select a region in which to create the Elastic Beanstalk application.



3. On the Elastic Beanstalk application navigation bar, click **Create New Application**.
4. Enter the name of the application and, optionally, a description. Then click **Next**.

That's it. The console will add your new application to the navigation bar at the top of the screen. Click on the name of an application in the navigation bar to view environments, application versions, and saved configurations associated with it.

After creating a new application, the console prompts you to create an environment for it. For detailed information about all of the options available, see Launching a New AWS Elastic Beanstalk Environment (p. 55). If you would just like to launch a sample application in your environment quickly, try the abridged instructions in Launching an Environment with a Sample Application in the AWS Management Console (p. 50).

Command Line

To create a new application with the AWS CLI

- Use the `create-application` command:

```
$ aws elasticbeanstalk create-application --application-name my-application
{
    "Application": {
        "ApplicationName": "my-application",
        "ConfigurationTemplates": [],
        "DateUpdated": "2015-07-07T21:47:32.191Z",
        "DateCreated": "2015-07-07T21:47:32.191Z"
    }
}
```

API

To create a new application

1. Call `CreateApplication` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`Description`* = `description`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&Description=description
&Operation=CreateApplicationVersion
&AuthParams
```

2. Call `CreateApplicationVersion` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`VersionLabel`* = `Version1`
   - *`Description`* = `description`
   - *`SourceBundle.S3Bucket`* = `<your S3 bucket name>`
   - *`SourceBundle.S3Key`* = `mynewjavawebapp-v1.war`

   ### Example

   ```
   https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
   &VersionLabel=Version1
   &Description=description
   &SourceBundle.S3Bucket=<your S3 bucket name>
   &SourceBundle.S3Key=mynewjavawebapp-v1.war
   &Operation=CreateApplicationVersion
   &AuthParams
   ```

3. Call `CheckDNSAvailability` with the following parameters:

   - *`CNAMEPrefix`* = `mysampleapplication`

   ### Example

   ```
   https://elasticbeanstalk.us-west-2.amazon.com/?CNAMEPrefix=mysampleapplication
   &Operation=CheckDNSAvailability
   &AuthParams
   ```

4. Call `CreateEnvironment` with one of the following sets of parameters:

   a. For a web server environment tier:

      - *`ApplicationName`* = `SampleApp`
      - *`VersionLabel`* = `Version1`
      - *`EnvironmentName`* = `mynewappenv`
      - *`SolutionStackName`* = `"32bit Amazon Linux running Tomcat 7"`
      - *`CNAMEPrefix`* = `mysampleapplication`
      - *`Description`* = `description`
      - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:launchconfiguration`
      - *`OptionSettings.member.1.OptionName`* = `IamInstanceProfile`
      - *`OptionSettings.member.1.Value`* = `ElasticBeanstalkProfile`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version1
&EnvironmentName=mynewappenv
&SolutionStackName=32bit%20Amazon%20Linux%20running%20Tomcat%207
&CNAMEPrefix=mysampleapplication
&Description=description
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=ElasticBeanstalkProfile
&AuthParams
```

b. For a worker environment tier:

- *EnvironmentName* = SampleAppEnv2
- *VersionLabel* = Version2
- *Description* = description
- *SolutionStackName* = "32bit Amazon Linux running Tomcat 7"
- *ApplicationName* = SampleApp
- *Tier* = Worker
- *OptionSettings.member.1.Namespace* = aws:autoscaling:launchconfiguration
- *OptionSettings.member.1.OptionName* = IamInstanceProfile
- *OptionSettings.member.1.Value* = ElasticBeanstalkProfile
- *OptionSettings.member.2.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.2.OptionName* = WorkerQueueURL
- *OptionSettings.member.2.Value* = sqsd.elasticbeanstalk.us-west-2.amazon.com
- *OptionSettings.member.3.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.3.OptionName* = HttpPath
- *OptionSettings.member.3.Value* = /
- *OptionSettings.member.4.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.4.OptionName* = MimeType
- *OptionSettings.member.4.Value* = application/json
- *OptionSettings.member.5.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.5.OptionName* = HttpConnections
- *OptionSettings.member.5.Value* = 75
- *OptionSettings.member.6.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.6.OptionName* = ConnectTimeout
- *OptionSettings.member.6.Value* = 10
- *OptionSettings.member.7.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.7.OptionName* = InactivityTimeout
- *OptionSettings.member.7.Value* = 10

- *OptionSettings.member.8.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.8.OptionName* = VisibilityTimeout
- *OptionSettings.member.8.Value* = 60
- *OptionSettings.member.9.Namespace* = aws:elasticbeanstalk:sqsd
- *OptionSettings.member.9.OptionName* = RetentionPeriod
- *OptionSettings.member.9.Value* = 345600

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&SolutionStackName=32bit%20Amazon%20Linux%20running%20Tomcat%207
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=ElasticBeanstalkProfile
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-west-2.amazon.com
&OptionSettings.member.3.Namespace=aws%3elasticbeanstalk%3sqsd
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.5.OptionName=HttpConnections
&OptionSettings.member.5.Value=75
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.6.OptionName=ConnectTimeout
&OptionSettings.member.6.Value=10
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.7.OptionName=InactivityTimeout
&OptionSettings.member.7.Value=10
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.8.OptionName=VisibilityTimeout
&OptionSettings.member.8.Value=60
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.9.OptionName=RetentionPeriod
&OptionSettings.member.9.Value=345600
&AuthParams
```

5. Call `DescribeEnvironments` with the following parameter:
   - *EnvironmentName* = mynewappenv

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=mynewappenv
&Operation=DescribeEnvironments
&AuthParams
```

Create an Application Version

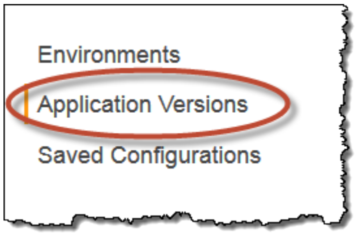
You can create different versions for an application. Each application version consists of a unique file (WAR file or ZIP file), as well as contextual information about the version. This topic describes how to create a new version of an existing Elastic Beanstalk application and deploy it to an existing environment. You may want to do this if, for instance, you have updated your application and want to re-deploy it to your testing environment. For information on how to create new application versions using the AWS Toolkit for Eclipse, see Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576). For more information on how to create new application versions for PHP, see Deploying Elastic Beanstalk Applications in PHP (p. 706). For more information on how to create new application versions using the AWS Toolkit for Visual Studio, see Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio (p. 612).

> **Note**
> For information on creating a new application, see Create an Application (p. 34).

AWS Management Console

To create a new application version

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the application that you want to work with.
3. From the Elastic Beanstalk console applications page, click the name of the application to which you want to add a new application version.
4. In the navigation pane, click **Application Versions**.



5. Click **Upload**.

   - Enter a label for this version in the **Version label** field.
   - (Optional) Enter a brief description for this version in the **Description** field.
   - Click **Browse** to specify the location of the application version (.war or .zip file).

     > **Note**
     > Elastic Beanstalk supports only a single .war file for a Java application version and only a single .zip file for other applications. The file size limit is 512 MB.

   - Click **Upload**.

The file you specified is associated with your application. You can deploy the application version to a new or existing environment. For more information, see Launching a New AWS Elastic Beanstalk Environment (p. 55) or Deploying Applications to AWS Elastic Beanstalk Environments (p. 78)

Command Line

To create a new application version with the AWS CLI

```
$ aws elasticbeanstalk create-application-version --application-name my-applic
ation --version-label v1
{
    "ApplicationVersion": {
        "ApplicationName": "my-application",
        "VersionLabel": "v1",
        "SourceBundle": {
            "S3Bucket": "elasticbeanstalk-us-west-2",
            "S3Key": "GenericSampleApplication"
        },
        "DateUpdated": "2015-07-07T21:56:42.426Z",
        "DateCreated": "2015-07-07T21:56:42.426Z"
    }
}
```

This command creates an application version with a sample application. To create an application version with your own code, specify the location of a source bundle in S3 with the `--source-bundle` option.

To create a new application version from a source bundle stored in S3

```
$ aws elasticbeanstalk create-application-version --application-name my-applic
ation --version-label v2 --source-bundle S3Bucket=my-bucket,S3Key=php-proxy-
sample.zip
{
    "ApplicationVersion": {
        "ApplicationName": "my-application",
        "VersionLabel": "v2",
        "SourceBundle": {
            "S3Bucket": "my-bucket",
            "S3Key": "php-proxy-sample.zip"
        },
        "DateUpdated": "2015-07-07T22:03:49.013Z",
        "DateCreated": "2015-07-07T22:03:49.013Z"
    }
}
```

API

To create a new application version

1. Call `CreateApplicationVersion` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`VersionLabel`* = `Version2`
   - *`Description`* = `description`

- *SourceBundle.S3Bucket* = <your bucket name>
- *SourceBundle.S3Key* = <your application file name>
- *AutoCreateApplication* = true

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&Description=description
&SourceBundle.S3Bucket=amazonaws.com
&SourceBundle.S3Key=sample.war
&AutoCreateApplication=true
&Operation=CreateApplicationVersion
&AuthParams
```

2. Call UpdateEnvironment with the following parameters:

   - *EnvironmentName* = SampleAppEnv
   - *VersionLabel* = Version2
   - *Description* = description
   - *TemplateName* = MyConfigTemplate

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=mysampleappenv
&TemplateName=myconfigtemplate
&Description=description
&VersionLabel=Version2
&Operation=UpdateEnvironment
&AuthParams
```

3. Call DescribeEnvironments with the following parameter:

   - *EnvironmentName* = SampleAppEnv

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&Operation=DescribeEnvironments
&AuthParams
```

Delete an Application Version

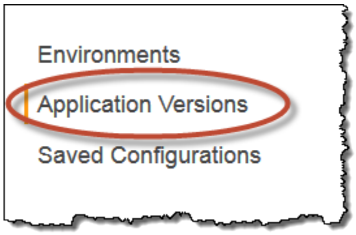
You can create different versions of a web application for an Elastic Beanstalk application. Each application version consists of a unique file (WAR file or ZIP file), as well as contextual information about the version. This topic describes how to delete an application version from an Elastic Beanstalk application. You might want to do this if, for example, you previously uploaded multiple application versions to test differences between one version of your web application and another, but you no longer need all versions. Or, you might want to do this because you reached the default limit of 500 application versions per AWS account.

> **Note**
> Elastic Beanstalk archives application versions to allow quick rollback. Deleting an application version does not delete the environment to which the application version is deployed, does not cause any downtime, or otherwise affect running environments.

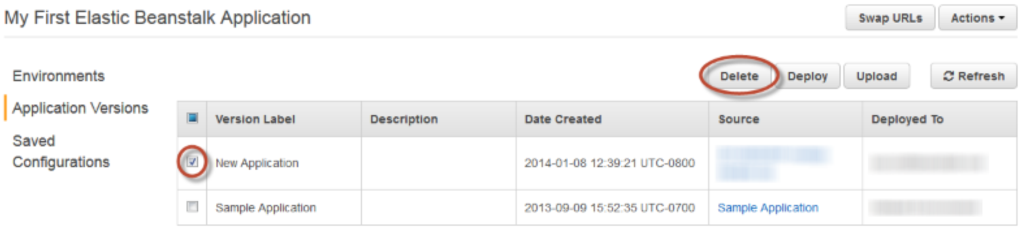
AWS Management Console

To delete an application version

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the application that you want to work with.
3. From the Elastic Beanstalk console applications page, click the name of the application from which you want to delete an application version.
4. In the navigation pane, click **Application Versions**.



5. In the list of application versions, select the check box next to the application version that you want to delete, and then click **Delete**.



■	Version Label	Description	Date Created	Source	Deployed To
☑	New Application		2014-01-08 12:39:21 UTC-0800	[redacted]	[redacted]
□	Sample Application		2013-09-09 15:52:35 UTC-0700	Sample Application	[redacted]

6. On the **Delete Application Versions** page, verify that the version label displayed represents the application version that you want to delete.
7. (Optional) To remove the application source bundle for this application version from your Amazon S3 bucket, select the **Delete versions from Amazon S3** check box.
8. When you are finished, click **Delete**, and then click **Done**.

Command Line

To delete an application version with the AWS CLI

```
$ aws elasticbeanstalk delete-application-version --application-name my-applic
ation --version-label v1
```

This command deletes the application version but leaves the source bundle used to create it in S3. If you would like to delete the source bundle as well as the application version, use the `--delete-source-bundle` option.

To delete an application version and the source bundle used to create it

```
$ aws elasticbeanstalk delete-application-version --application-name my-applic
ation --version-label v2 --delete-source-bundle
```

> **Note**
> No output is generated by the command.

API

To delete an application version

- Call `DeleteApplicationVersion` with the following parameters:
  - *`ApplicationName`* = `SampleApp`
  - *`VersionLabel`* = `Version2`
  - *`DeleteSourceBundle`* = `false`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&DeleteSourceBundle=false
&AuthParams
```

Create an Application Source Bundle

When you use the AWS Elastic Beanstalk console to deploy a new application or an application version, you'll need to upload a source bundle. Your source bundle must meet the following requirements:

- Consist of a single ZIP file or WAR file (you can include multiple `WAR` files inside your `ZIP` file)
- Not exceed 512 MB
- Not include a parent folder or top-level directory (subdirectories are fine)

If you want to deploy a worker application that processes periodic background tasks, your application source bundle must also include a `cron.yaml` file. For more information, see Periodic Tasks[2] (p. 72).

This section explains how to create a source bundle manually.

> **Note**
> If you're creating a source bundle with the eb command-line tool, the AWS Toolkit for Eclipse, or the AWS Toolkit for Visual Studio, the ZIP or WAR file will automatically be structured correctly. For more information, go to EB Command Line Interface (p. 384), Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576), and AWS Toolkit for Visual Studio (p. 662).

Zipping Files in Mac OS X Finder or Windows Explorer

When you create a `ZIP` file in Mac OS X Finder or Windows Explorer, make sure you zip the files and subfolders themselves, rather than zipping the parent folder.

> **Note**
> The graphical user interface (GUI) on Mac OS X and Linux-based operating systems does not display files and folders with names that begin with a period (.). Use the command line instead of the GUI to compress your application if the `ZIP` file must include a hidden folder, such as `.ebextensions`. For command line procedures to create a `ZIP` file on Mac OS X or a Linux-based operating system, see Creating a Source Bundle from the Command Line (p. 46).

Example

Suppose you have a Python project folder labeled `myapp`, which includes the following files and subfolders:

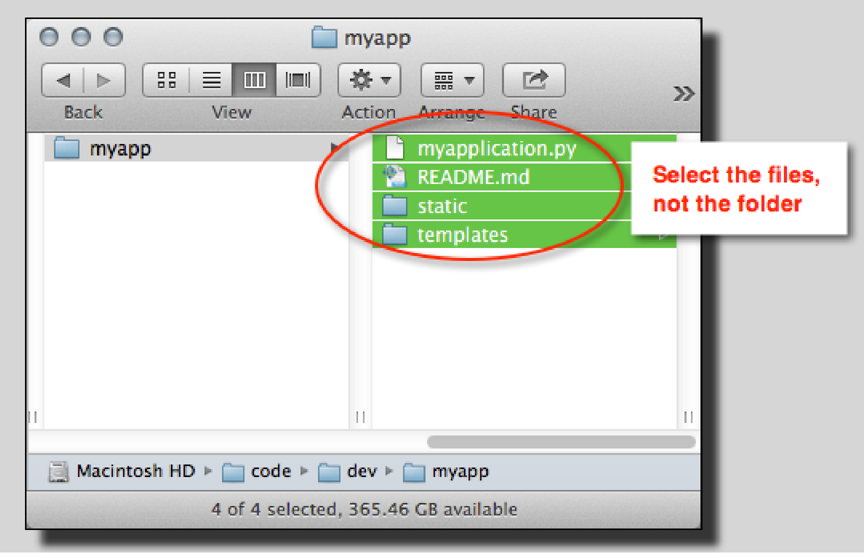
```
myapplication.py
README.md
static/
static/css
static/css/styles.css
static/img
static/img/favicon.ico
static/img/logo.png
templates/
templates/base.html
templates/index.html
```

As noted in the list of requirements above, your source bundle must be compressed without a parent folder, so that its decompressed structure does not include an extra top-level directory. In this example, no `myapp` folder should be created when the files are decompressed (or, at the command line, no `myapp` segment should be added to the file paths).

This sample file structure is used throughout this topic to illustrate how to zip files.

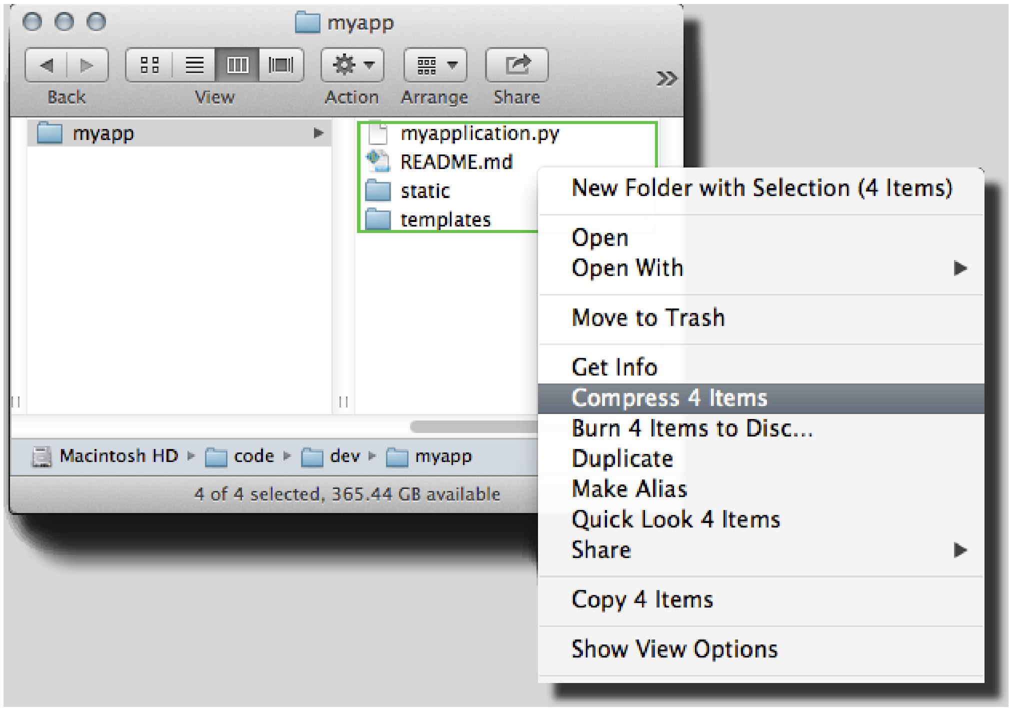
To zip files in Mac OS X Finder

1. Open your top-level project folder and select all the files and subfolders within it. Do not select the top-level folder itself.



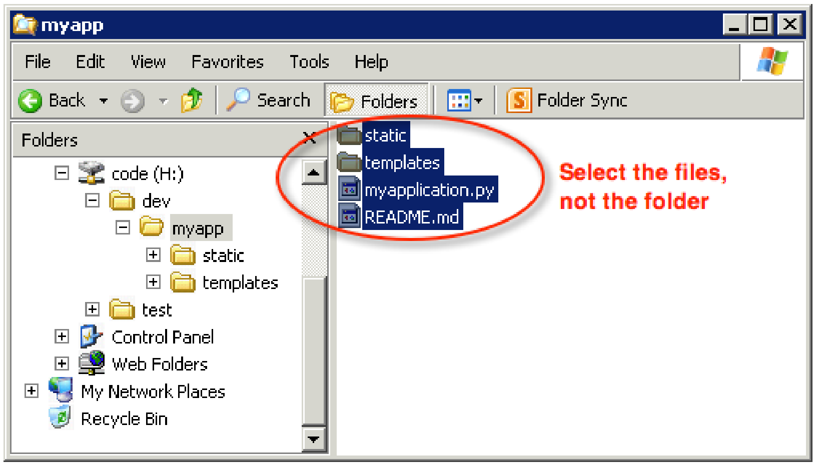
Select the files, not the folder

2. Right-click the selected files, and then click **Compress *X* items**, where *X* is the number of files and subfolders you've selected.



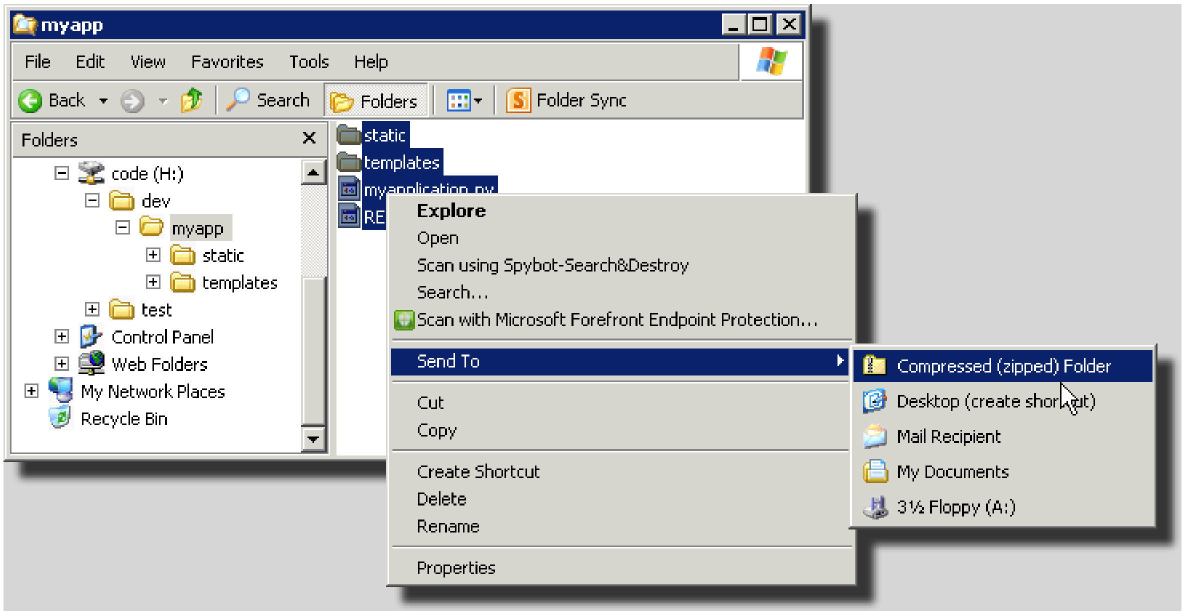
To zip files in Windows Explorer

1. Open your top-level project folder and select all the files and subfolders within it. Do not select the top-level folder itself.



Select the files, not the folder

2. Right-click the selected files, click **Send to**, and then click **Compressed (zipped) folder**.

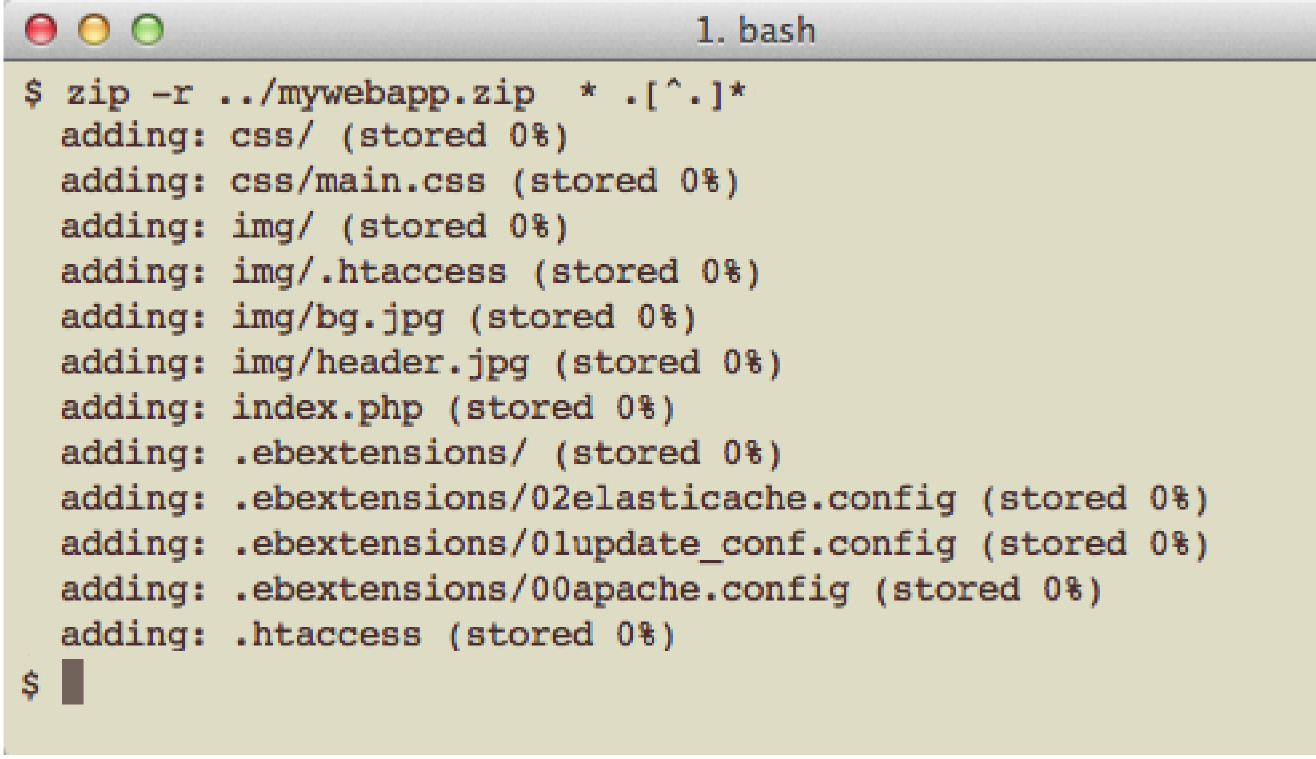


Creating a Source Bundle from the Command Line

When you create a source bundle using a `zip` command or `jar` command (Mac OS X or Linux), you may want to work in the same directory as your source files, rather than in a parent folder or other higher-level directory. This will help ensure that the file paths in the compressed archive do not include an extra parent folder. Mac OS X and Linux-based operating systems hide files and folders with names that begin with a period (.). The following command ensures that your source bundle includes folders that begin with a period, such as the `.ebextensions` folder.

Example

```
$ zip ../myapp.zip -r * .[^.]*
$ jar -cvf mywebapp.war *
```



```
$ zip -r ../mywebapp.zip  * .[^.]*
  adding: css/ (stored 0%)
  adding: css/main.css (stored 0%)
  adding: img/ (stored 0%)
  adding: img/.htaccess (stored 0%)
  adding: img/bg.jpg (stored 0%)
  adding: img/header.jpg (stored 0%)
  adding: index.php (stored 0%)
  adding: .ebextensions/ (stored 0%)
  adding: .ebextensions/02elasticache.config (stored 0%)
  adding: .ebextensions/01update_conf.config (stored 0%)
  adding: .ebextensions/00apache.config (stored 0%)
  adding: .htaccess (stored 0%)
$ 
```

Creating a Source Bundle with Git

If you're using Git to manage your application source code, you can use the `archive` command to create your source bundle.

To bundle your most recent Git commit

- To create a ZIP archive of your most recent Git commit on the current branch, type the following command or issue it via a Git client, replacing *<myapp>* with your preferred archive name:

  ```
  $ git archive --format=zip HEAD > <myapp>.zip
  ```

  For more details, go to the git-archive manual page.

Testing Your Source Bundle

You may want to test your source bundle locally before you upload it to Elastic Beanstalk. Because Elastic Beanstalk essentially uses the command line to extract the files, it's best to do your tests from the command line rather than with a GUI tool.

To test the file extraction in Mac OS X or Linux

1. Open a terminal window (Mac OS X) or connect to the Linux server. Navigate to the directory that contains your source bundle.
2. Using the `unzip` or `jar xf` command, decompress the archive.
3. Ensure that the decompressed files appear in the same folder as the archive itself, rather than in a new top-level folder or directory.

Note
If you use Mac OS X Finder to decompress the archive, a new top-level folder will be created, no matter how you structured the archive itself. For best results, use the command line.

To test the file extraction in Windows

1. Download or install a program that allows you to extract compressed files via the command line. For example, you can download the free unzip.exe program from http://stahlforce.com/dev/index.php?tool=zipunzip.
2. If necessary, copy the executable file to the directory that contains your source bundle. If you've installed a system-wide tool, you can skip this step.
3. Using the appropriate command, decompress the archive. If you downloaded unzip.exe using the link in step 1, the command is `unzip` *`<archive-name>`*.
4. Ensure that the decompressed files appear in the same folder as the archive itself, rather than in a new top-level folder or directory.

Bundling Multiple WAR Files for Java with Tomcat Environments

You can deploy multiple web applications to each instance in a Tomcat environment by bundling multiple WAR files into a single source bundle (p. 43). Simplify deployments and reduce operating costs by running multiple applications in a single environment, instead of running a separate environment for each application. This strategy is effective for lightweight applications that don't require a lot of resources.

Application Source Bundle Structure for Multiple WAR Files

To create an application source bundle that contains multiple WAR files, organize the WAR files using the following structure:

```
MyApplication.zip
    .ebextensions
    foo.war
    bar.war
    ROOT.war
```

When you deploy a source bundle containing multiple WAR files to an AWS Elastic Beanstalk (Elastic Beanstalk)environment, each application is accessible from a different path off of the root domain name. The above example includes three applications, `foo`, `bar`, and `ROOT`. `ROOT.war` is a special filename that tells Elastic Beanstalk to run that application at the root domain, so the three applications are available at `http://MyApplication.elasticbeanstalk.com/foo`, `http://MyApplication.elasticbeanstalk.com/bar`, and `http://MyApplication.elasticbeanstalk.com`.

The source bundle must include only an `.ebextensions` folder and WAR files. The `.ebextensions` folder is optional and can contain configuration files that customize the resources deployed to your environment. See Elastic Beanstalk Environment Configuration (p. 99) for information on using configuration files.

For information about creating source bundles, see Create an Application Source Bundle (p. 43).

To upload an application source bundle containing multiple WAR files

1. Launch AWS Elastic Beanstalk in the AWS Management Console.
2. Choose **Create New Application**.
3. Type an **Application name**, and then choose **Next**.
4. Choose **Create web server**.
5. When prompted for an IAM instance profile with the appropriate permissions, choose the default instance profile (`aws-elasticbeanstalk-ec2-role`), or create one, if prompted. Choose **Next**.
6. On the **Environment Type** page, in **Predefined configuration**, choose **Tomcat**.
7. In the **Predefined configuration** area, choose **Change platform version**.
8. Choose the Tomcat solution stack that matches the version of Java you are using, and then choose **Next**.
9. On the **Application Version** page, for **Source**, choose **Upload your own**, and then choose **Browse**.
10. Choose your application source bundle, and then choose **Open**.
11. Choose **Next** to accept the default settings until you reach the **Review** page.
12. Choose **Launch**.

Filter Applications in Your Environment

You can filter the list of all Elastic Beanstalk applications deployed in an environment. You may want to do this if, for example, you deployed a large number of applications in one environment. The AWS Management Console can help you quickly find an Elastic Beanstalk application in an environment. You can search for applications within only one environment at a time.

To filter a list of applications in your environment

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work in.
3. On the Elastic Beanstalk **All Applications** page, click **Filter by Application Name**.
4. Type part or all of the name of the application you want to find.

As you type, Elastic Beanstalk dynamically displays a list of applications with names that include the text in your search query.

Managing Environments

By using the Elastic Beanstalk console, you can save or change the provisioning and configuration of the AWS resources your application environments use. For information about managing your application environments using the AWS Toolkit for Eclipse, see Managing Elastic Beanstalk Application Environments (p. 599). For information about managing your application environments using the AWS Toolkit for Visual Studio, see Managing Your Elastic Beanstalk Application Environments (p. 651). For information about launching a new environment with a saved configuration, see Launching a New AWS Elastic Beanstalk Environment (p. 55).

Elastic Beanstalk environments are managed environments. If you directly use another service to change settings that Elastic Beanstalk can manage on your behalf, Elastic Beanstalk does not track the changes in the same way it would if you made the changes using Elastic Beanstalk. For example, consider a scenario where you use Amazon EC2 to remap the Elastic IP address of an instance in an environment that you launched using Elastic Beanstalk to another instance. When you delete the environment, Elastic Beanstalk will delete the IP address as though it were still part of the environment. We strongly recommend that you use the procedures in this section to administer your environments rather than perform unmanaged configuration changes.

Topics

Launching an Environment with a Sample Application in the AWS Management Console

The **Create New Environment** wizard in the AWS Management Console guides you through the creation of an environment step by step, with a bevy of options for configuring the resources that Elastic Beanstalk deploys on your behalf. If you are just getting started, you can use the default values for many of these options without issue.

> **Note**
> Creating an environment requires the permissions in the Elastic Beanstalk full access managed policy. See Elastic Beanstalk User Policy (p. 22) for details.

Follow this procedure to launch a new environment running the default application or a sample application provided in a source bundle (ZIP or WAR file). Once the environment is up and running, you can check out the various configuration options available and upload your own code at your own pace.

To launch an environment with a sample application

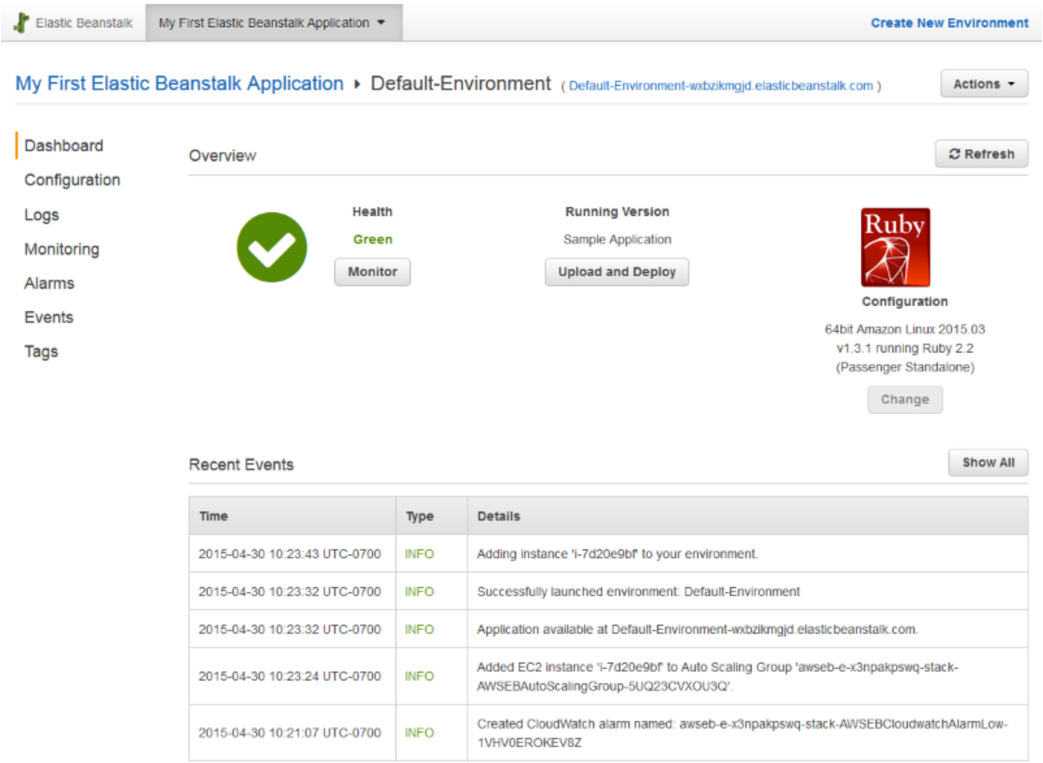
1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Select an existing application or create a new one (p. 34).
3. Click **Create a New Environment** in the upper right corner.
4. Click **Create web server**.
5. For **Permissions**, click **Next** to use the default instance profile or create a new one.
6. Click the drop down menu under **Predefined configuration** and select the platform configuration that matches the language or framework that you want to work in and click **Next**.
7. For **Application Version**, select **Sample Application** under **Existing application version**, or click **Upload your own** and upload an application source bundle. Click **Next**.
8. Enter an **Environment name** and URL prefix for your environment and click **Next**.
9. If the application requires a database, select **Create an RDS DB Instance**.
10. Click **Next**.
11. For **Configuration Details**, set **Instance type** to `t2.micro` and click **Next**.
12. Click **Next** to skip tag creation.
13. Click **Next** on the **Permissions** page. If you don't have the default instance profile and service role, Elastic Beanstalk prompts you to create them. Accept the prompt to create the default roles and assign them to your environment.

    > **Note**
    > Creating roles requires additional permissions. See Service Roles, Instance Profiles, and User Policies (p. 19) for details.
14. If you added a database to the environment, enter a username and password and click **Next**.
15. Click **Launch**

While Elastic Beanstalk creates your environment, you are redirected to the Environment Management Console (p. 51). Once the environment health turns green, click on the URL next to the environment name to view the running application.

Environment Management Console

The AWS Management Console provides a management page for each of your AWS Elastic Beanstalk environments. From this page, you can manage your environment's configuration and perform common actions including restarting the web servers running in your environment, cloning the environment, or rebuilding it from scratch.



Time	Type	Details
2015-04-30 10:23:43 UTC-0700	INFO	Adding instance 'i-7d20e9bf' to your environment.
2015-04-30 10:23:32 UTC-0700	INFO	Successfully launched environment: Default-Environment
2015-04-30 10:23:32 UTC-0700	INFO	Application available at Default-Environment-wxbzikmgjd.elasticbeanstalk.com.
2015-04-30 10:23:24 UTC-0700	INFO	Added EC2 instance 'i-7d20e9bf' to Auto Scaling Group 'awseb-e-x3npakpswq-stack-AWSEBAutoScalingGroup-5UQ23CVXOU3Q'.
2015-04-30 10:21:07 UTC-0700	INFO	Created CloudWatch alarm named: awseb-e-x3npakpswq-stack-AWSEBCloudwatchAlarmLow-1VHV0EROKEV8Z

Topics

To access the environment management console, open the Elastic Beanstalk console in your region and click on the name of a running environment. Environments are shown in color coded tiles under their associated application. The color (green, grey or red) indicates the health of the environment.

At the top of the environment console, the name of the application is shown, followed by the name of the environment and the public DNS name of the running application.

Environment Dashboard

The main view of the environment management console is the **Dashboard**.

Within the environment management dashboard is an overview, which shows the environment's health, the application version, and information about the in-use platform, and a list of recent events generated by the environment.

Click **Refresh** to update the information shown. The overview contains the following information and options.

Health

The overall health of the environment. With Enhanced Health Reporting and Monitoring (p. 250) enabled, the environment status is shown with a **Causes** button you can click to view more information about the current status.

For Basic Health Reporting (p. 247) environments, a link to the Monitoring Console (p. 244) is shown.

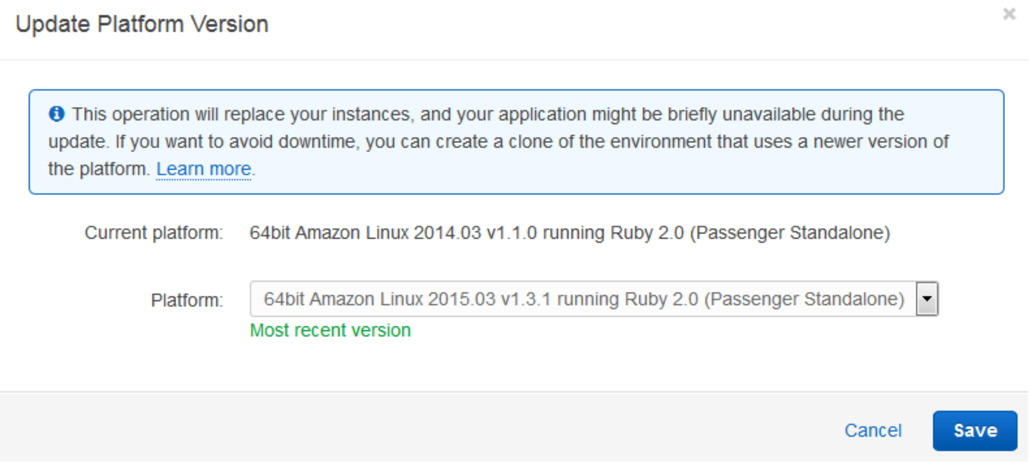
Running Version

The name of the application version running on your environment. Click **Upload and Deploy** to upload a source bundle (p. 43) and deploy it to your environment. This option creates a new application version.

Configuration

Shows the architecture, OS version, and platform running on your environment. Click **Change** to select a different. This option is only available if another compatible version of the platform is available. To be considered compatible, the architecture, OS name, and platform name must be the same.

Updating the platform version using this option replaces instances running in your environment with new instances.



Update Platform Version

This operation will replace your instances, and your application might be briefly unavailable during the update. If you want to avoid downtime, you can create a clone of the environment that uses a newer version of the platform. Learn more.

Current platform: 64bit Amazon Linux 2014.03 v1.1.0 running Ruby 2.0 (Passenger Standalone)

Platform: 64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.0 (Passenger Standalone)
Most recent version

> **Note**
> When you first use Beanstalk, only the latest version of each platform is available for use. The **Change** first becomes available when a new version of the operating system or platform is released. After upgrading, you have the option to change back to the previous version.

Recent Events

The **Recent Events** section of the environment management dashboard shows the most recent events emitted by your environment. This list is updated in real time when your environment is being updated.

Click **Show All** to open the **Events** menu.

Environment Management Actions

The environment management console contains an **Actions** drop down menu that you can use to perform common operations on your environment. This menu is shown on the right side of the environment header under the **Create New Environment** option.

> **Note**
> Some actions are only available under certain conditions, and will be greyed out unless these conditions are met.

Load Configuration

Load a previously saved configuration. Configurations are saved to your application and can be loaded by any associated environment. If you've made changes to your environment's configuration, you can load a saved configuration to undo those changes. You can also load a configuration that you saved from a different environment running the same application to propagate configuration changes between them.

Save Configuration

Save the current configuration of your environment to your application. Prior to making changes to your environment's configuration, save the current configuration so that you can roll back later if needed. You can also apply a saved configuration when you launch a new environment.

Swap Environment URLs

Swap the CNAME of the current environment with a new environment. After a CNAME swap, all traffic to the application using the environment URL will go to the new environment. When you are ready to deploy a new version of your application, you can launch a separate environment under the new version. When the new environment is ready to start taking requests, perform a CNAME swap to start routing traffic to the new environment with no interruption of service. For more information see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

Clone Environment

Launch a new environment with the same configuration as your currently running environment.

Clone with Latest Platform

Clone your current environment with the latest version of the in-use application platform. This option is only available when a newer version of the current environment's platform is available for use.

Abort Current Operation

Stop an in progress environment update. Aborting an operation can cause some of the instances in your environment to be in a different state than others, depending on how far the operation progressed. This option is only available when your environment is being updated.

Restart App Server(s)

Restart the web server running on your environment's instances. This option does not terminate or restart any AWS resources. If your environment is acting strangely in response to some bad requests, restarting the application server may restore functionality temporarily while you troubleshoot the root cause.

Rebuild Environment

Terminate all resources in the running environment and build a new environment with the same settings. This operation takes several minutes, equivalent to deploying a new environment from scratch. Any RDS instances running in your environment's data tier are deleted during a rebuild. If you need the data, create a snapshot. You can create a snapshot manually in the RDS console or configure your data tier's Deletion Policy to create a snapshot automatically prior to deleting the instance (this is the default setting when you create a data tier).

Terminate Environment

Terminate all resources in the running environment, and remove the environment from the application. If you have an RDS instance running in a data tier and need to retain the data, ensure that a snapshot is taken prior to terminating your environment. You can create a snapshot manually in the RDS console or configure your data tier's Deletion Policy to create a snapshot automatically prior to deleting the instance (this is the default setting when you create a data tier).

Launching a New AWS Elastic Beanstalk Environment

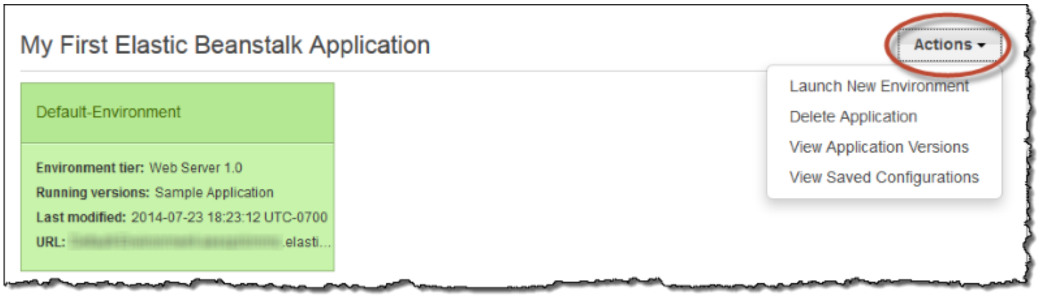
You can deploy multiple environments when you need to run multiple versions of an application. For example, you might have development, integration, and production environments. When launching, you can deploy a different version to any environment quickly and easily. For more information about deploying with zero downtime, see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

Important
After you create an environment, the environment URL is publicly visible.

AWS Management Console

To launch a new environment

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that has the application that you want to work in.
3. From the Elastic Beanstalk console applications page, click **Actions** for the application in which you want to launch a new environment.



My First Elastic Beanstalk Application

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2014-07-23 18:23:12 UTC-0700
URL: [redacted].elasti...

4. Click **Launch New Environment**.
5. On the **New Environment** page, select an environment tier. The environment tier setting specifies whether you want a **Web Server** or **Worker** tier. For more information, see Architectural Overview (p. 16).

   Note
   After you launch an environment, you cannot change the environment tier. If your application requires a different environment tier, you must launch a new environment. When you remove an environment, the AWS resources associated with the environment and the application version you deployed in that environment are deleted. You can save the environment configuration to use later if you want to rebuild an environment using the same environment tier.

6. In the **Permissions** window, choose the instance profile that you want to use to launch the Amazon EC2 instances in the new environment. (An instance profile is associated with the IAM role that is configured with the appropriate permissions to access the AWS resources that your application will require. A worker environment tier requires different permissions from a web server environment tier. For more information, see IAM Roles for Elastic Beanstalk Environment Tiers (p. 381).) Do one of the following:

   - For a web server environment tier, click the name of the instance profile with the appropriate IAM role, and then click **Next**.
   - For a worker environment tier, do one of the following:

- If you want Elastic Beanstalk to create a role with the default permissions required by a worker application, click **Create an IAM role and instance profile**, and then click **Next**.
- If you have an existing instance profile with an IAM role that is configured with the permissions required by a worker application, click **Select an existing instance profile**, and then click **Next**.

7. If, in the previous step, you clicked **Create an IAM role and instance profile**, click **View Details** to see options for the role, and then do one of the following:

   - To add default permissions for a worker application to an existing role, click **IAM Role**, click the name of the instance profile for which you want to add a role policy, and then click **Allow**. You can optionally associate a saved policy with an existing role by clicking **Policy Name**, and then clicking the name of the policy before you click **Allow**.
   - To create a new role with default permissions, click **Allow**. You can optionally change the **Role Name** and edit the policy document.

8. On the **Environment Type** page, select a platform and environment type, and then click **Next**.

   - The **Predefined configuration** setting specifies the platform and version that will be used for the environment. For more information, see Supported Platforms (p. 24).

     Note
     After you launch an environment with a specific configuration, you cannot change the configuration. If your application requires a different configuration, you must launch a new environment.
   - The **Saved configuration** setting lists all environment configurations that you previously saved for this application, if any. If you have no saved configurations for this application, Elastic Beanstalk does not display this option in the console.
   - The **Environment type** specifies whether the environment is load balancing and autoscaling or only a single Amazon EC2 instance. For more information, see Environment Types (p. 68).

     Note
     The single-instance environment type is not available for legacy containers. For instructions on migrating from a legacy container, see Migrating Your Application from a Legacy Container Type (p. 91).

9. On the **Application Version** page, you can use the sample application, upload your own, or specify the URL for the Amazon S3 bucket that contains your application code.

   Note
   Depending on the platform configuration you selected, you can upload your application in a ZIP source bundle (p. 43), a WAR file, or a plaintext Docker configuration. You can include multiple `WAR` files inside a `ZIP` file to deploy multiple Tomcat applications to each instance in your environment. The file size limit is 512 MB.
10. For load-balancing, autoscaling environments only, you can also control downtime when application versions are deployed to your environment later. Next to **Batch size**, click **Percentage** or **Fixed**. Enter a percentage or fixed number of instances to which you want to deploy the new application version at any given time, and then click **Next**.
11. On the **Environment Information** page, enter the details of your environment.

    a. For a web server environment tier:

       - Enter a name for the environment.
       - Enter a unique environment URL. Even though the environment URL is populated with the environment name, you can enter a different name for the URL. Elastic Beanstalk uses this

name to create a unique CNAME for the environment. You can check the availability of the URL by clicking **Check Availability**

- (Optional) Enter a description for this environment.
- Click **Next**.

b. For a worker environment tier:

- Enter a name for the environment.
- (Optional) Enter a description for this environment.
- Click **Next**.

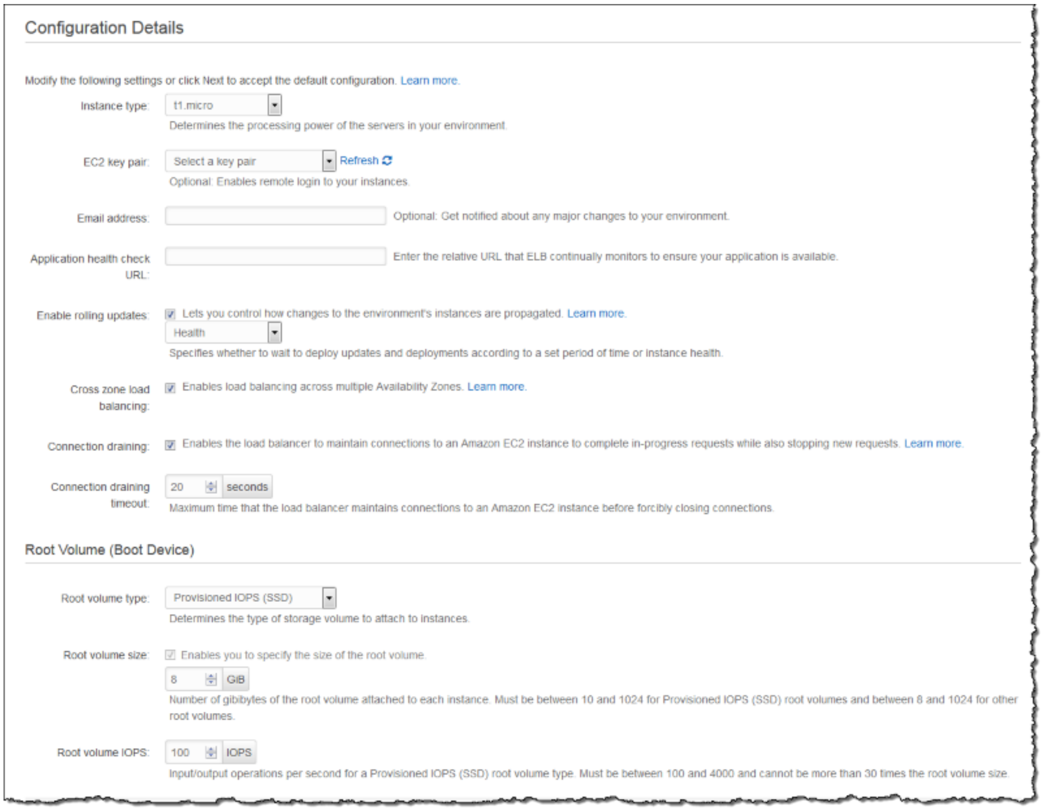
12. (Optional) On the **Additional Resources** page, select additional resources for the environment, and then click **Next**. Note the following:

- Unless you are creating an application using a legacy container type, you have the option to associate an Amazon RDS database. If you want to associate an Amazon RDS DB with this application, select **Create an RDS Database with this environment**. For more information about Amazon RDS, go to Amazon Relational Database Service (Amazon RDS). For a detailed list of container types that provide the option to include an Amazon RDS database, see Supported Platforms (p. 24).

  Note
  If you are using a legacy container type, then the Amazon RDS option does not appear. For a list of supported legacy container types, see Why are some container types marked legacy? (p. 92). For more information about configuring databases with legacy container types, see Configuring Databases with Elastic Beanstalk (p. 195).

- Unless you are creating an application using a legacy container type, you have the option to create your environment inside a VPC. To do this, select **Create this environment inside a VPC**. For more information about Amazon VPC, go to Amazon Virtual Private Cloud (Amazon VPC). For a list of supported legacy container types, see Why are some container types marked legacy? (p. 92).

13. Set configuration details for the environment, and then click **Next**.



Configuration Details

Modify the following settings or click Next to accept the default configuration. Learn more.

Instance type: t1.micro
Determines the processing power of the servers in your environment.

EC2 key pair: Select a key pair Refresh
Optional: Enables remote login to your instances.

Email address: Optional: Get notified about any major changes to your environment.

Application health check URL: Enter the relative URL that ELB continually monitors to ensure your application is available.

Enable rolling updates: ☑ Lets you control how changes to the environment's instances are propagated. Learn more.
Health
Specifies whether to wait to deploy updates and deployments according to a set period of time or instance health.

Cross zone load balancing: ☑ Enables load balancing across multiple Availability Zones. Learn more.

Connection draining: ☑ Enables the load balancer to maintain connections to an Amazon EC2 instance to complete in-progress requests while also stopping new requests. Learn more.

Connection draining timeout: 20 seconds
Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections.

Root Volume (Boot Device)

Root volume type: Provisioned IOPS (SSD)
Determines the type of storage volume to attach to instances.

Root volume size: ☑ Enables you to specify the size of the root volume.
8 GiB
Number of gibibytes of the root volume attached to each instance. Must be between 10 and 1024 for Provisioned IOPS (SSD) root volumes and between 8 and 1024 for other root volumes.

Root volume IOPS: 100 IOPS
Input/output operations per second for a Provisioned IOPS (SSD) root volume type. Must be between 100 and 4000 and cannot be more than 30 times the root volume size.

- **Instance type** displays the instance types available to your Elastic Beanstalk environment. Select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application.

  Note
  Elastic Beanstalk is free, but the AWS resources that it provisions might not be. For information on Amazon EC2 usage fees, go to Amazon EC2 Pricing.

  For more information about the Amazon EC2 instance types available for your Elastic Beanstalk environment, go to Instance Families and Types in the *Amazon EC2 User Guide for Linux Instances*.
- **EC2 key pair** shows all the Amazon EC2 key pairs in your AWS account. Select a key pair if you need to log in securetly to the Amazon EC2 instances provisioned for your Elastic Beanstalk application.

  For more information about Amazon EC2 key pairs, see Using Credentials in the *Amazon EC2 User Guide for Linux Instances*. For more information on connecting to Amazon EC2 instances, see Connecting to Instances and Connecting to an Instance from Windows using PuTTY in the *Amazon EC2 User Guide for Linux Instances*.
- **Email address** specifies who receives Amazon Simple Notification Service notifications about important events regarding your application. If you want to receive email notifications of important events, enter an email address. You can disable Amazon SNS notifications at a later time by removing the email address in the configuration settings of your running environment.
- For load-balancing, autoscaling environments, **Application health check URL** specifies a resource in your application that Elastic Load Balancing checks for a `200 OK` response. For more information, see Basic Health Reporting (p. 247).
- **Enable rolling updates** provides options for managing how instances are updated or replaced. For more information, see Updating Elastic Beanstalk Environments with Rolling Updates (p. 169).

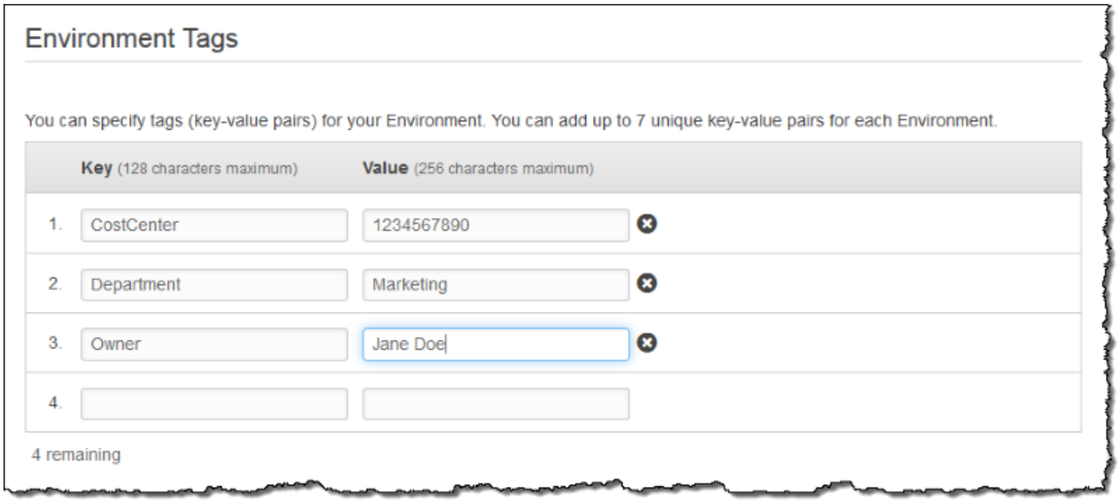
> **Note**
> You can configure rolling updates after you have saved your environment configuration. Rolling updates uses default settings when first enabled.

- For load-balancing, autoscaling environments, **Cross-zone load balancing** configures the load balancer to route traffic evenly among all Amazon EC2 instances, regardless of the Availability Zone, instead of within a single Availability Zone only. For more information, see Enabling cross-zone load balancing (p. 189).
- For load-balancing, autoscaling environments, **Connection draining** keeps connections open between the load balancer and Amazon EC2 instances that are unhealthy or deregistering for the purposes of completing in-progress requests. For more information, see Connection Draining (p. 189).
- When you enable connection draining, specify the **Connection draining timeout** value as the maximum number of seconds that the load balancer allows for the completion of in-progress requests. Connections are automatically forced closed after 300 seconds when you do not specify a draining timeout.
- **Root volume type** displays the types of storage volumes provided by Amazon EBS that you can attach to Amazon EC2 instances in your Elastic Beanstalk environment. Select the volume type that meets your performance and price requirements. For more information, see Amazon EBS Volume Types and Amazon EBS Product Details.
- With **Root volume size**, you can specify the size of the storage volume that you selected. You must specify your desired root volume size if you choose **Provisioned IOPS (SSD)** as the root volume type that your instances will use. For other root volumes, if you do not specify your own value, Elastic Beanstalk will use the default volume size for the storage volume type. The default volume size varies according to the AMI of the solution stack on which your environment is based. For Provisioned IOPS (SSD) root volumes, the minimum number of gibibytes is 10 and the maximum is 1024. For other root volumes, the minimum number of gibibytes is 8 and the maximum is 1024.
- If you selected **Provisioned IOPS (SSD)** as your root volume type, you must specify your desired input/output operations per second (IOPS). The minimum is 100 and the maximum is 4000. The maximum ratio of IOPS to your volume size is 30 to 1. For example, a volume with 3000 IOPS must be at least 100 GiB.

14. (Optional) On the **Environment Tags** page, create tags for the environment, and then click **Next**. Restrictions on tag keys and tag values include the following:

    - Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / = + \ - @
    - Keys can contain up to 128 characters. Values can contain up to 256 characters.
    - Keys and values are case sensitive.
    - Values cannot match the environment name.
    - Values cannot include either **`aws:`** or **`elasticbeanstalk:`**.

    For more information about using tags, see Tagging Your Amazon EC2 Resources in the *Amazon EC2 User Guide for Linux Instances*.



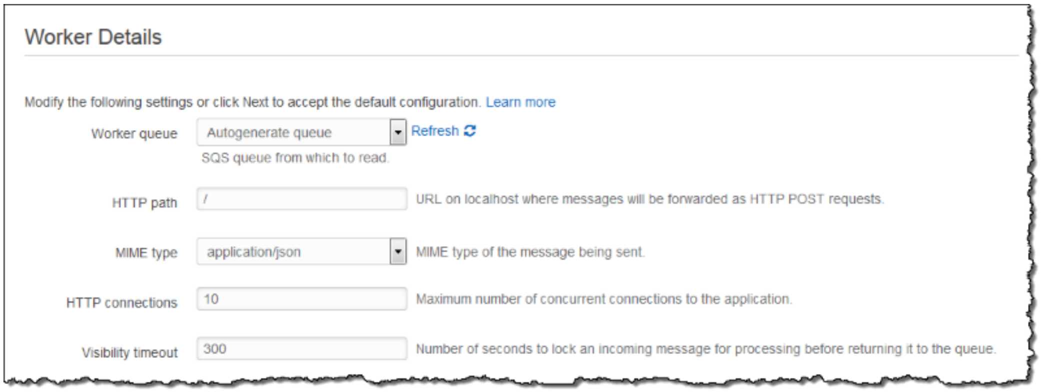
Environment Tags

You can specify tags (key-value pairs) for your Environment. You can add up to 7 unique key-value pairs for each Environment.

	Key (128 characters maximum)	**Value** (256 characters maximum)
1.	CostCenter	1234567890
2.	Department	Marketing
3.	Owner	Jane Doe
4.		

4 remaining

15. If you are creating a worker environment tier, on the **Worker Details** page, set the following preliminary worker environment tier details. Then click **Next**. You can also click **Next** to accept the default values.



Worker Details

Modify the following settings or click Next to accept the default configuration. Learn more

Worker queue: Autogenerate queue — Refresh
SQS queue from which to read.

HTTP path: / — URL on localhost where messages will be forwarded as HTTP POST requests.

MIME type: application/json — MIME type of the message being sent.

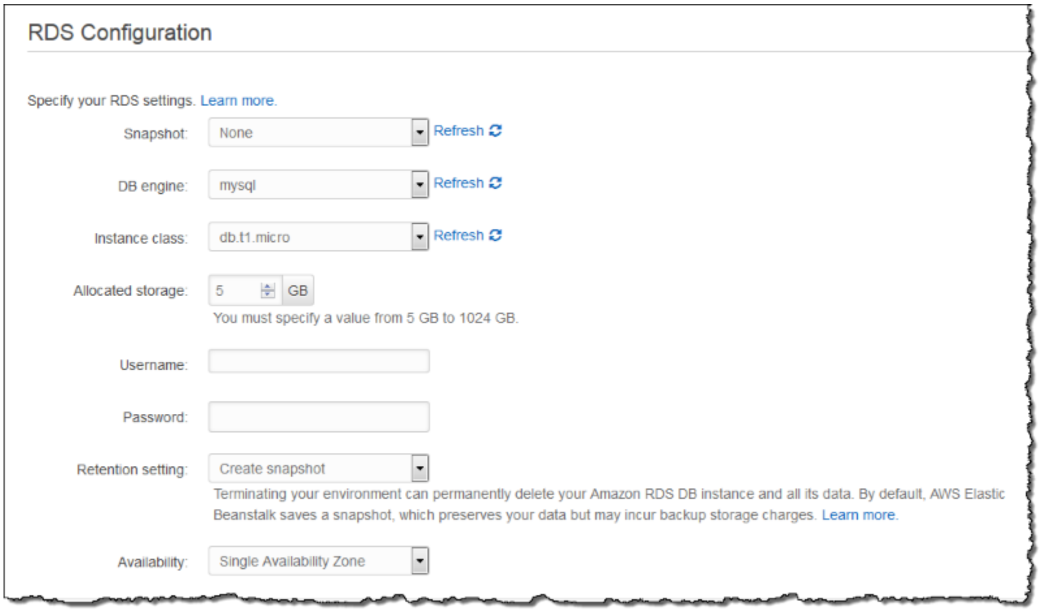
HTTP connections: 10 — Maximum number of concurrent connections to the application.

Visibility timeout: 300 — Number of seconds to lock an incoming message for processing before returning it to the queue.

- **Worker queue** specifies the queue from which the worker environment tier reads messages that it will process. If you do not provide a value, then Elastic Beanstalk automatically creates one for you.
- **HTTP path** specifies the relative path on the local host to which messages from the queue are forwarded in the form of HTTP POST requests.
- **MIME type** specifies the MIME type of the message sent in the HTTP POST request.
- **HTTP connections** specifies the maximum number of concurrent connections to the application. Set this to the number of process or thread messages your application can process in parallel.
- **Visibility timeout** specifies how long an incoming message is locked for processing before being returned to the queue. Set this to the potentially longest amount of time that might be required to process a message.

16. If you chose to associate an Amazon RDS DB earlier in the environment configuration process, on the **RDS Configuration** page, set the Amazon RDS configuration settings, and then click **Next**.

> **Note**
> If you are using a legacy container type, you cannot use Amazon RDS with Elastic Beanstalk. Elastic Beanstalk does not display an **RDS Configuration** page when you create an environment with a legacy container type. For more information about configuring databases with legacy container types, see Configuring Databases with Elastic Beanstalk (p. 195).



RDS Configuration

Specify your RDS settings. Learn more.

Snapshot: None

DB engine: mysql

Instance class: db.t1.micro

Allocated storage: 5 GB

You must specify a value from 5 GB to 1024 GB.

Username:

Password:

Retention setting: Create snapshot

Terminating your environment can permanently delete your Amazon RDS DB instance and all its data. By default, AWS Elastic Beanstalk saves a snapshot, which preserves your data but may incur backup storage charges. Learn more.

Availability: Single Availability Zone

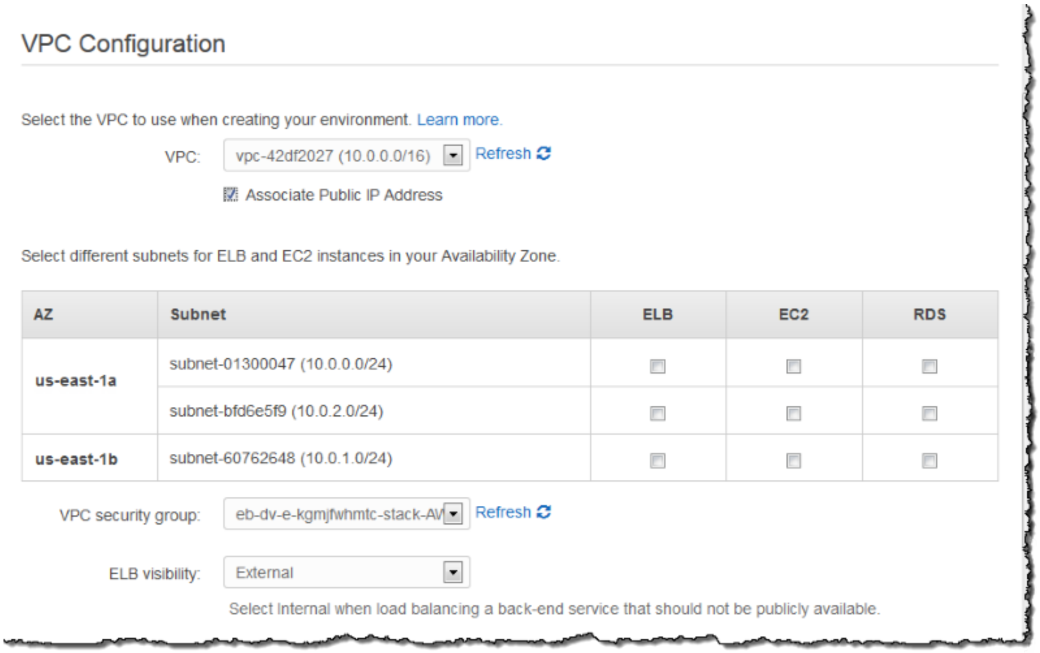
- (Optional) For **Snapshot**, select whether to create an Amazon RDS DB from an existing snapshot.
- (Optional) For **DB engine**, select a database engine.
- (Optional) For **Instance Class**, select a database instance class. For information about the DB instance classes, go to http://aws.amazon.com/rds/.
- For **Allocated Storage**, type the space needed for your database. You can allocate between 5 GB and 1024 GB. You cannot update the allocated storage for a database to a lower amount after you set it. In some cases, allocating a larger amount of storage for your DB instance than the size of your database can improve IO performance. For information about storage allocation, go to Features.
- For **Master Username**, type a name using alphanumeric characters that you will use to log in to your DB instance with all database privileges.
- For **Master Password**, type a password containing 8 to 16 printable ASCII characters (excluding /, \, and @).
- For **Deletion Policy**, select **Create snapshot** to create a snapshot that you can use later to create another Amazon RDS database. Select **Delete** to delete the DB instance when you terminate the environment. If you select **Delete**, you lose your DB instance and all the data in it when you terminate the Elastic Beanstalk instance associated with it. By default, Elastic Beanstalk creates and saves a snapshot. You can use a snapshot to restore data to use in a new environment, but cannot otherwise recover lost data.

  Note
  You may incur charges for storing database snapshots. For more information, see the "Backup Storage" section of Amazon RDS Pricing.

- For **Availability**, select one of the following:
  - To configure your database in one Availability Zone, select **Single Availability Zone**. A database instance launched in one Availability Zone does not have protection from the failure of a single location.
  - To configure your database across multiple availability zones, select **Multiple Availability Zones**. Running your database instance in multiple Availability Zones helps safeguard your data in the unlikely event of a database instance component failure or service health disruption in one Availability Zone.

17. If you chose to create an environment inside a VPC earlier in the environment creation process, set the VPC configuration settings, and then click **Next**.

    Note
    If you are using a legacy container type, you cannot configure Amazon VPC with Elastic Beanstalk. Elastic Beanstalk does not display an **VPC Configuration** page when you create an environment with a legacy container type.



VPC Configuration

Select the VPC to use when creating your environment. Learn more.

VPC: vpc-42df2027 (10.0.0.0/16) Refresh

☑ Associate Public IP Address

Select different subnets for ELB and EC2 instances in your Availability Zone.

AZ	Subnet	ELB	EC2	RDS
us-east-1a	subnet-01300047 (10.0.0.0/24)	□	□	□
	subnet-bfd6e5f9 (10.0.2.0/24)	□	□	□
us-east-1b	subnet-60762648 (10.0.1.0/24)	□	□	□

VPC security group: eb-dv-e-kgmjfwhmtc-stack-AV Refresh

ELB visibility: External

Select Internal when load balancing a back-end service that should not be publicly available.

- Select the VPC ID of the VPC in which you want to launch your environment.

  Note
  If you do not see the VPC information, then you have not created a VPC in the same region in which you are launching your environment. To learn how to create a VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).

- For a load-balancing, autoscaling environment, select the subnets for the Elastic Load Balancing load balancer and the Amazon EC2 instances. If you created a single public subnet, select the **Associate Public IP Address** check box, and then select the check boxes for the load balancer and the Amazon EC2 instances. If you created public and private subnets, make sure the load balancer (public subnet) and the Amazon EC2 instances (private subnet) are associated with the correct subnet. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your existing subnets in the Amazon VPC console at https://console.aws.amazon.com/vpc/.
- For a single-instance environment, select a public subnet for the Amazon EC2 instance. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24. You can view your existing subnets in the Amazon VPC console at https://console.aws.amazon.com/vpc/.
- If you are using Amazon RDS, you must select at least two subnets in different Availability Zones. To learn how to create subnets for your VPC, go to Task 1: Create the VPC and Subnets in the *Amazon VPC User Guide*.
- If you have a NAT instance (usually when you have instances in a private subnet), select the VPC security group that is assigned to the NAT instance. For instructions on how to create this security group and update your default VPC security group, see Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 305). If you do not have a NAT instance, you can use the default security group.

- For a load-balancing, autoscaling environment, select whether you want to make the load balancer external or internal. If you do not want your load balancer to be available to the Internet, select **Internal**.

18. On the **Review Information** page, review your application and environment information, and then click **Launch**.

    Elastic Beanstalk launches your application in a new environment. Note that it can take several minutes for the new environment to start while Elastic Beanstalk is provisioning AWS resources. You can view the status of your deployment on the environment's dashboard. While Elastic Beanstalk creates your AWS resources and launches your application, the environment displays a gray state. Status messages about launch events appear in the environment's dashboard. When the deployment is complete, AWS Elastic Beanstalk performs an application health check. The environment status becomes green when the application responds to the health check.

CLI

To launch a new environment

1. Check if the CNAME for the environment is available.

```
$ aws elasticbeanstalk check-dns-availability --cname-prefix my-cname
{
    "Available": true,
    "FullyQualifiedCNAME": "my-cname.elasticbeanstalk.com"
}
```

2. Make sure your application version exists.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-
app --version-label v1
```

3. Create a configuration template for an existing application.

```
$ aws elasticbeanstalk create-configuration-template --application-name my-
app --template-name v1
```

4. Create environment.

```
$ aws elasticbeanstalk create-environment --cname-prefix my-cname --applic
ation-name my-app --template-name v1 --version-label v1 --environment-name
 v1clone --option-settings file://options.txt
```

Option Settings are defined in the **options.txt** file:

```
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "aws-elasticbeanstalk-ec2-role"
    }
]
```

The above option setting defines the IAM instance profile. You can specify the ARN or the profile name.

5. Determine if the new environment is Green and Ready.

```
$ aws elasticbeanstalk describe-environments --environment-names my-env
```

If the new environment does not come up Green and Ready, you should decide if you want to retry the operation or leave the environment in its current state for investigation. Make sure to terminate the environment after you are finished, and clean up any unused resources.

Note
You can adjust the timeout period if the environment doesn't launch in a reasonable time.

API

To launch a new environment

1. Call `CheckDNSAvailability` with the following parameter:

   - *`CNAMEPrefix`* = `SampleApp`

   ### Example

```
https://elasticbeanstalk.us-east-1.amazon.com/?CNAMEPrefix=sampleapplication
&Operation=CheckDNSAvailability
&AuthParams
```

2. Call `DescribeApplicationVersions` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`VersionLabel`* = `Version2`

   ### Example

```
https://elasticbeanstalk.us-east-1.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&Operation=DescribeApplicationVersions
&AuthParams
```

3. Call `CreateConfigurationTemplate` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`TemplateName`* = `MyConfigTemplate`

Example

```
https://elasticbeanstalk.us-east-1.amazon.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&Operation=CreateConfigurationTemplate
&AuthParams
```

4. Call `CreateEnvironment` with one of the following sets of parameters.

   a. Use the following for a web server environment tier:

   - *`EnvironmentName`* = `SampleAppEnv2`
   - *`VersionLabel`* = `Version2`
   - *`Description`* = `description`
   - *`TemplateName`* = `MyConfigTemplate`
   - *`ApplicationName`* = `SampleApp`
   - *`CNAMEPrefix`* = `sampleapplication`
   - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:launchconfiguration`
   - *`OptionSettings.member.1.OptionName`* = `IamInstanceProfile`
   - *`OptionSettings.member.1.Value`* = `ElasticBeanstalkProfile`

   ### Example

   ```
   https://elasticbeanstalk.us-east-1.amazon.com/?ApplicationName=SampleApp
   &VersionLabel=Version2
   &EnvironmentName=SampleAppEnv2
   &TemplateName=MyConfigTemplate
   &CNAMEPrefix=sampleapplication
   &Description=description
   &Operation=CreateEnvironment
   &OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
   &OptionSettings.member.1.OptionName=IamInstanceProfile
   &OptionSettings.member.1.Value=ElasticBeanstalkProfile
   &AuthParams
   ```

   b. Use the following for a worker environment tier:

   - *`EnvironmentName`* = `SampleAppEnv2`
   - *`VersionLabel`* = `Version2`
   - *`Description`* = `description`
   - *`TemplateName`* = `MyConfigTemplate`
   - *`ApplicationName`* = `SampleApp`
   - *`Tier`* = `Worker`
   - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:launchconfiguration`
   - *`OptionSettings.member.1.OptionName`* = `IamInstanceProfile`
   - *`OptionSettings.member.1.Value`* = `ElasticBeanstalkProfile`
   - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:sqsd`

- *OptionSettings.member.2.OptionName* = `WorkerQueueURL`
- *OptionSettings.member.2.Value* = `sqsd.elasticbeanstalk.us-east-1.amazon.com`
- *OptionSettings.member.3.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.3.OptionName* = `HttpPath`
- *OptionSettings.member.3.Value* = `/`
- *OptionSettings.member.4.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.4.OptionName* = `MimeType`
- *OptionSettings.member.4.Value* = `application/json`
- *OptionSettings.member.5.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.5.OptionName* = `HttpConnections`
- *OptionSettings.member.5.Value* = `75`
- *OptionSettings.member.6.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.6.OptionName* = `ConnectTimeout`
- *OptionSettings.member.6.Value* = `10`
- *OptionSettings.member.7.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.7.OptionName* = `InactivityTimeout`
- *OptionSettings.member.7.Value* = `10`
- *OptionSettings.member.8.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.8.OptionName* = `VisibilityTimeout`
- *OptionSettings.member.8.Value* = `60`
- *OptionSettings.member.9.Namespace* = `aws:elasticbeanstalk:sqsd`
- *OptionSettings.member.9.OptionName* = `RetentionPeriod`
- *OptionSettings.member.9.Value* = `345600`

Example

```
https://elasticbeanstalk.us-east-1.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=ElasticBeanstalkProfile
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-east-1.amazon.com
&OptionSettings.member.3.Namespace=aws%3elasticbeanstalk%3sqsd
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.5.OptionName=HttpConnections
&OptionSettings.member.5.Value=75
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.6.OptionName=ConnectTimeout
&OptionSettings.member.6.Value=10
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.7.OptionName=InactivityTimeout
&OptionSettings.member.7.Value=10
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.8.OptionName=VisibilityTimeout
&OptionSettings.member.8.Value=60
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.9.OptionName=RetentionPeriod
&OptionSettings.member.9.Value=345600
&AuthParams
```

5. Call `DescribeEnvironments` with the following parameter:

   - *EnvironmentName* = `SampleAppEnv2`

   ### Example

   ```
   https://elasticbeanstalk.us-east-1.amazon.com/?EnvironmentName=SampleAppEnv2
   &Operation=DescribeEnvironments
   &AuthParams
   ```

Environment Types

In Elastic Beanstalk, you can create a load-balancing, autoscaling environment or a single-instance environment. The type of environment that you require depends on the application that you deploy. For example, you can develop and test an application in a single-instance environment to save costs and then upgrade that environment to a load-balancing, autoscaling environment when the application is ready for production.

> **Note**
> A worker environment tier for a web application that processes background tasks does not include a load balancer. However, a worker environment does effectively scale out by adding instances to the autoscaling group to process data from the Amazon SQS queue when the load necessitates it.

Load-balancing, Autoscaling Environment

A load-balancing and autoscaling environment uses the Elastic Load Balancing and Auto Scaling services to provision the Amazon EC2 instances that are required for your deployed application. Auto Scaling automatically starts additional instances to accommodate increasing load on your application. If the load on your application decreases, Auto Scaling stops instances but always leaves your specified minimum number of instances running. If your application requires scalability with the option of running in multiple Availability Zones, use a load-balancing, autoscaling environment. If you're not sure which environment type to select, you can pick one, and if required, switch the environment type later. For more information, see Enable Load Balancing (p. 97).

Single-instance Environment

A single-instance environment contains one Amazon EC2 instance with an Elastic IP address. A single-instance environment doesn't have a load balancer, which can help you reduce costs compared to a load-balancing, autoscaling environment. Although a single-instance environment does use the Auto Scaling service, settings for the minimum number of instances, maximum number of instances, and desired capacity are all set to 1. Consequently, new instances are not started to accommodate increasing load on your application.

Use a single-instance environment if you expect your production application to have low traffic or if you are doing remote development. If you're not sure which environment type to select, you can pick one, and, if required, you can switch the environment type later. For more information, see Enable Load Balancing (p. 97).

The single-instance environment type is available for nonlegacy containers only. For instructions on migrating from a legacy container, see Migrating Your Application from a Legacy Container Type (p. 91).

> **Important**
> In a single-instance environment, you cannot use the Elastic Beanstalk console to configure settings for the Auto Scaling group. Instead, you can edit the options file and then pass in the options file using the AWS Command Line Interface or the Elastic Beanstalk API-based command line interface. Any IAM users that work with single-instance environments must have Auto Scaling permissions. For more information about the options file, see Option Values. For more information about working with IAM, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326).
> Because single-instance environments do not use the Elastic Load Balancing service, you cannot configure load balancer settings. You also do not need to configure Elastic Load Balancing permissions for your IAM users in this case.

Worker Environments

When you launch an Elastic Beanstalk environment, you choose an environment tier, platform, and environment type. The environment tier that you choose determines whether Elastic Beanstalk provisions resources to support a web application that handles HTTP(S) requests or a web application that handles background-processing tasks. An environment whose web application processes web requests is known as a *web server environment*. An environment tier whose web application runs background jobs is known as a *worker environment*.

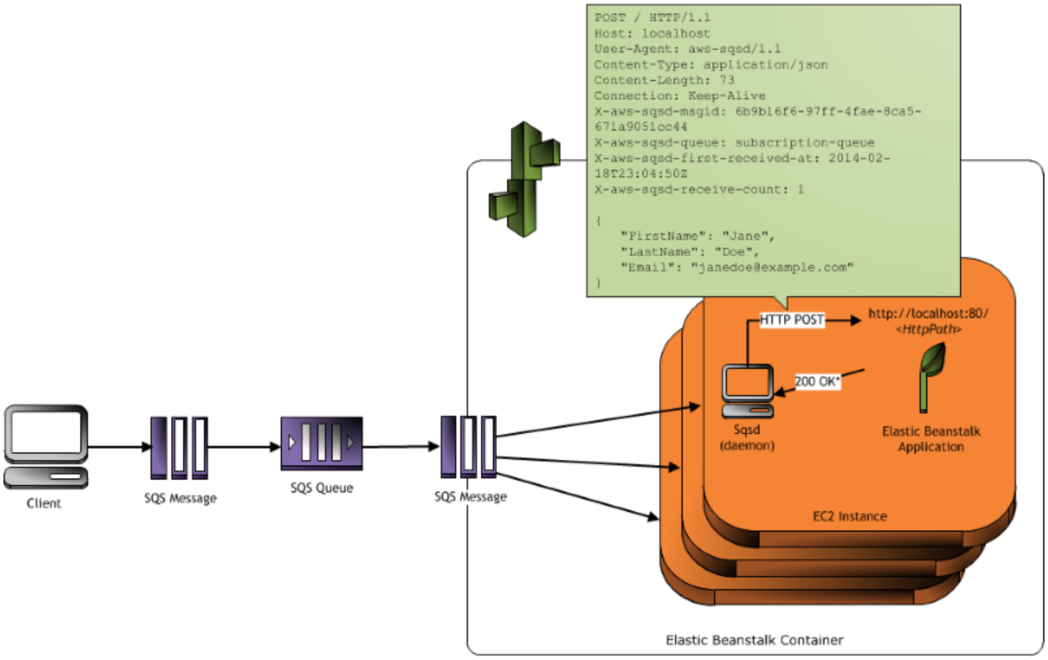
You can deploy a worker environment on its own to perform background-processing tasks for any AWS service that can write to an Amazon Simple Queue Service queue (for example, Amazon EC2 or AWS OpsWorks). Or you can deploy it alongside an Elastic Beanstalk web server tier. You can use a worker environment to execute long-running tasks or tasks that can be performed asynchronously. By offloading background-processing tasks to a worker environment, you free up the web application in your web server environment to handle web requests.

How a Worker Environment Works

Elastic Beanstalk installs a daemon on each Amazon EC2 instance in the Auto Scaling group to process Amazon SQS messages in the worker environment. The daemon pulls data off the Amazon SQS queue, inserts it into the message body of an HTTP POST request, and sends it to a user-configurable URL path on the local host. The content type for the message body within an HTTP POST request is `application/json` by default.

> **Important**
> We strongly recommend that you familiarize yourself with how Amazon SQS works if you plan to deploy a worker environment. In particular, the properties of Amazon SQS queues (message order, at-least-once delivery, and message sampling) can affect how you design a web application for a worker environment. For more information, see Properties of Distributed Queues in the Amazon Simple Queue Service Developer Guide.

The following diagram illustrates an example of the worker environment processing an Amazon SQS message.



```
POST / HTTP/1.1
Host: localhost
User-Agent: aws-sqsd/1.1
Content-Type: application/json
Content-Length: 73
Connection: Keep-Alive
X-aws-sqsd-msgid: 6b9bl6f6-97ff-4fae-8ca5-
671a9051cc44
X-aws-sqsd-queue: subscription-queue
X-aws-sqsd-first-received-at: 2014-02-
18T23:04:50Z
X-aws-sqsd-receive-count: 1

{
   "FirstName": "Jane",
   "LastName": "Doe",
   "Email": "janedoe@example.com"
}
```

* HTTP Response of 200 OK = delete the message
Any other HTTP Response = retry the message after the VisibilityTimeout period
No response = retry the message after the InactivityTimeout period

The daemon sets the following HTTP headers:

Note
HTTP header names are not case-sensitive. For more information, see 4.2 Message Headers in the Hypertext Transfer Protocol -- HTTP/1.1 specification.

HTTP Headers

Name	Value
User-Agent	`aws-sqsd` `aws-sqsd/1.1`[1]
X-Aws-Sqsd-Msgid	SQS message ID, used to detect message storms
X-Aws-Sqsd-Queue	Name of the SQS queue
X-Aws-Sqsd-First-Received-At	Time stamp showing when the message was first received (in the UTC time zone) **Note** The time stamp is conveyed using the ISO 8601 time format. For more information, go to http://www.w3.org/TR/NOTE-datetime.
X-Aws-Sqsd-Receive-Count	SQS message receive count
Content-Type	Mime type configuration; by default, `application/json`
X-Aws-Sqsd-Taskname[2]	Name of the periodic task

HTTP Headers	
X-Aws-Sqsd-Attr-*message-attribute-name*[2]	Custom message attributes assigned to the message being processed. The `message-attribute-name` is the actual message attribute name. All string and number message attributes are added to the header, Binary attributes are discarded and not included in the header.
X-Aws-Sqsd-Scheduled-At[2]	Time at which the periodic task was scheduled
X-Aws-Sqsd-Sender-Id[2]	AWS account number of the sender of the message

The requests are sent to the **HTTP Path** value that you configure. This is done in such a way as to appear to the web application in the worker environment that the daemon originated the request. In this way, the daemon serves a similar role to a load balancer in a web server environment.

The worker environment, after processing the messages in the queue, forwards the messages over the local loopback to a web application at a URL that you designate. The queue URL is only accessible from the local host. Because you can only access the queue URL from the same EC2 instance, no authentication is needed to validate the messages that are delivered to the URL.

A web application in a worker environment should only listen on the local host. When the web application in the worker environment returns a `200 OK` response to acknowledge that it has received and successfully processed the request, the daemon sends a `DeleteMessage` call to the SQS queue so that the message will be deleted from the queue. (SQS automatically deletes messages that have been in a queue for longer than the configured `RetentionPeriod`.) If the application returns any response other than `200 OK`, then Elastic Beanstalk waits to put the message back in the queue after the configured `VisibilityTimeout` period. If there is no response, then Elastic Beanstalk waits to put the message back in the queue after the `InactivityTimeout` period so that the message is available for another attempt at processing.

Dead Letter Queues[1]

Worker environments[1] include support for Amazon Simple Queue Service (SQS) dead letter queues. A dead letter queue is a queue where other (source) queues can send messages that for some reason could not be successfully processed. A primary benefit of using a dead letter queue is the ability to sideline and isolate the unsuccessfully processed messages. You can then analyze any messages sent to the dead letter queue to try to determine why they were not successfully processed.

> **Note**
> Worker environments created prior to May 27, 2014 do not support dead letter queues. You must create a new environment in order to use a dead letter queue. Before you create a new environment, you can save the configuration settings from your existing environment and then use the saved configuration to launch a new environment. For more information about saving environment configuration settings, see Saving Environment Configuration Settings (p. 101). For more information about launching a new environment, see Launching a New AWS Elastic Beanstalk Environment (p. 55).

A dead letter queue is enabled by default for a worker environment if you specify an autogenerated Amazon SQS queue at the time you create your worker environment tier. If you choose an existing SQS queue for your worker environment, you must use SQS to configure a dead letter queue independently. For information about how to use SQS to configure a dead letter queue, see Using Amazon SQS Dead Letter Queues.

You cannot disable dead letter queues. Messages that cannot be delivered will always eventually be sent to a dead letter queue. You can, however, effectively disable this feature by setting the MaxRetries option to the maximum valid value of 1000.

Note

The Elastic Beanstalk `MaxRetries` option is equivalent to the SQS `MaxReceiveCount` option. If your worker environment does not use an autogenerated SQS queue, use the `MaxReceiveCount` option in SQS to effectively disable your dead letter queue. For more information, see Using Amazon SQS Dead Letter Queues.

For more information about the lifecycle of an SQS message, go to Message Lifecycle.

Periodic Tasks[2]

Elastic Beanstalk supports periodic tasks for worker environment tiers[2] in environments running a predefined configuration with a solution stack that contains "v1.2.0" in the container name. You must create a new environment. (Elastic Beanstalk considers a clone environment a new environment.) A qualified environment can support an application that performs periodic tasks in addition to continuous polling for and processing messages from the Amazon SQS queue. However, Amazon SQS processes messages in the order that it receives them. As a result, a periodic task can be delayed when the Amazon SQS queue has many messages to process. We recommend that you design separate applications when punctuality for periodic tasks is critical.

To invoke periodic tasks, your application source bundle must include a `cron.yaml` file at the root level. The file must contain information about the periodic tasks you want to schedule. Specify this information using standard crontab syntax. For more information, see CRON expression.

The following snippet contains example file contents for the `cron.yaml` file. This file instructs a specified EC2 instance to run the `backup-job` job every 12 hours and the `audit` job every day at 11pm in local time. (11pm is represented as hour 23 of the day.) Each job `name` must be unique within the file. The `url` is appended to the application URL.

```
version: 1
cron:
 - name: "backup-job"          # required - unique across all entries in this
file
   url: "/backup"               # required - does not need to be unique
   schedule: "0 */12 * * *"     # required - does not need to be unique
 - name: "audit"
   url: "/audit"
   schedule: "0 23 * * *"
```

Use Amazon CloudWatch for Auto Scaling in Worker Environment Tiers

Together, Auto Scaling and CloudWatch monitor the CPU utilization of the running instances in the worker environment. How you configure the autoscaling limit for CPU capacity determines how many instances the autoscaling group runs to appropriately manage the throughput of messages in the SQS queue. Each EC2 instance publishes its CPU utilization metrics to CloudWatch. Auto Scaling retrieves from CloudWatch the average CPU usage across all instances in the worker environment. You configure the upper and lower threshold as well as how many instances to add or terminate according to CPU capacity. When Auto Scaling detects that you have reached the specified upper threshold on CPU capacity, Elastic Beanstalk creates new instances in the worker environment. The instances are deleted when the CPU load drops back below the threshold.

Note

Messages that have not been processed at the time an instance is terminated are once again made visible on the queue where they can be processed by another daemon on an instance that is still running.

You can also set other CloudWatch alarms, as needed, by using the AWS Management Console, CLI, or the options file. For more information, go to Using Elastic Beanstalk with Amazon CloudWatch (p. 290) and Use Auto Scaling Policies and Amazon CloudWatch Alarms for Dynamic Scaling.

To publish metrics to CloudWatch, you must configure an IAM policy that grants your IAM role permission to send data to CloudWatch. For more information, see Granting IAM Role Permissions for Worker Environment Tiers (p. 333).

About Creating a Worker Environment

When you create an Elastic Beanstalk environment or update an existing environment, whether through the AWS Management Console, CreateEnvironment API, UpdateEnvironment API, the eb command line, or the AWS command line, you specify whether you want a **Web Server** or **Worker** environment. You cannot have one environment that is both a web server environment and a worker environment because Elastic Beanstalk supports only one Auto Scaling group per environment. By default, Elastic Beanstalk launches a web server environment. You cannot change the environment tier after you launch an environment. If your web application needs a different kind of environment tier, you must launch a new environment.

> **Note**
> The CreateEnvironment and UpdateEnvironment APIs have an attribute called `tier`. (The DescribeEnvironments API has a `tier` parameter as part of its response and will omit some parameters from its response if the tier it describes is a worker environment. The DescribeEnvironmentResources API has an attribute called `EnvironmentResources`.)

If you use an existing Amazon SQS queue, the settings that you configure when you create a worker environment can conflict with settings you configured directly in Amazon SQS. For example, if you configure a worker environment with a RetentionPeriod value that is higher than the MessageRetentionPeriod value you set in Amazon SQS, then Amazon SQS will delete the message when it exceeds the MessageRetentionPeriod. Conversely, if the RetentionPeriod value you configure in the worker environment settings is lower than the MessageRetentionPeriod value you set in Amazon SQS, then the daemon will delete the message before Amazon SQS can. For VisibilityTimeout, the value that you configure for the daemon in the worker environment settings overrides the Amazon SQS VisibilityTimeout setting. Ensure that messages are deleted appropriately by comparing your Elastic Beanstalk settings to your Amazon SQS settings.

If you don't specify an existing Amazon SQS queue when you configure a worker environment tier, Elastic Beanstalk will create one for you. You can get the URL by calling DescribeEnvironmentResources.

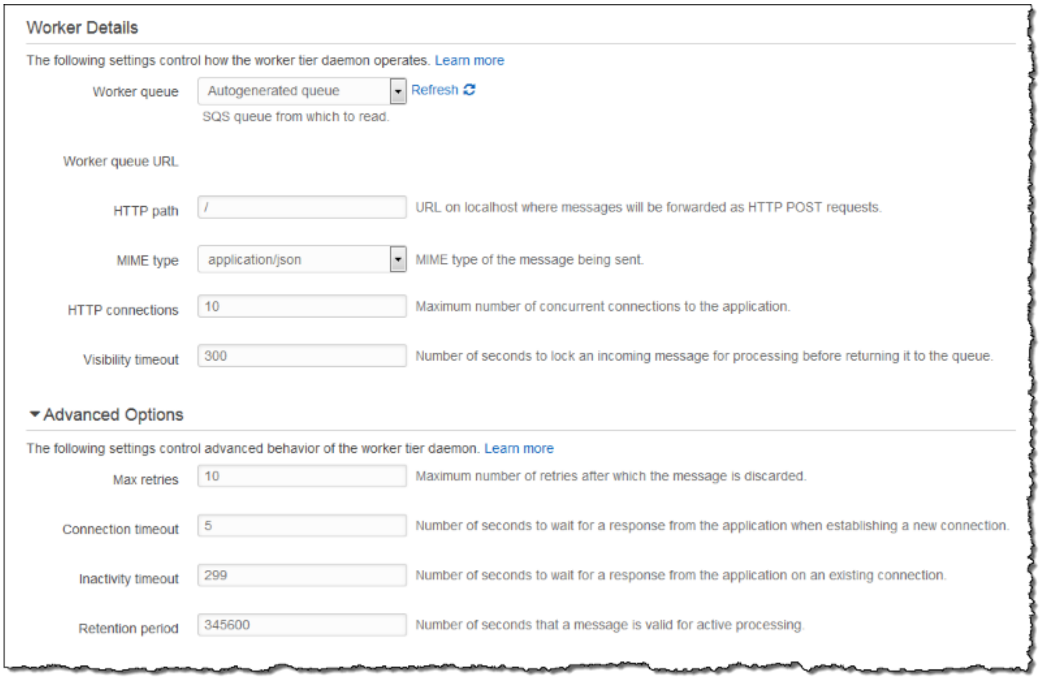
For procedures to launch an environment, go to Launching a New AWS Elastic Beanstalk Environment (p. 55).

Configuring Worker Environments with Elastic Beanstalk

As noted earlier, Elastic Beanstalk installs a daemon on each Amazon EC2 instance in the Auto Scaling group. After the worker environment is created, you can control how that daemon processes Amazon SQS messages. For example, you can configure additional settings such as the retention period during which a message is valid or the visibility timeout period during which a message is not visible in the Amazon SQS queue because it is locked for processing.

AWS Management Console

You can manage a worker environment's configuration by editing **Worker Configuration** on the **Configuration** (p. 100) page for that environment.



Worker Details

The following settings control how the worker tier daemon operates. Learn more

Worker queue: Autogenerated queue — Refresh — SQS queue from which to read.

Worker queue URL

HTTP path: / — URL on localhost where messages will be forwarded as HTTP POST requests.

MIME type: application/json — MIME type of the message being sent.

HTTP connections: 10 — Maximum number of concurrent connections to the application.

Visibility timeout: 300 — Number of seconds to lock an incoming message for processing before returning it to the queue.

Advanced Options

The following settings control advanced behavior of the worker tier daemon. Learn more

Max retries: 10 — Maximum number of retries after which the message is discarded.

Connection timeout: 5 — Number of seconds to wait for a response from the application when establishing a new connection.

Inactivity timeout: 299 — Number of seconds to wait for a response from the application on an existing connection.

Retention period: 345600 — Number of seconds that a message is valid for active processing.

The **Worker Details** page has the following options:

- **Worker queue** – Specify the Amazon SQS queue from which the daemon reads. You can choose an existing queue, if you have one. If you choose **Autogenerated queue**, Elastic Beanstalk creates a new Amazon SQS queue and a corresponding **Worker queue URL**.
- **Worker queue URL** – If you choose an existing **Worker queue**, then this setting displays the URL associated with that Amazon SQS queue.
- **HTTP path** – Specify the relative path to the application that will receive the data from the Amazon SQS queue. The data is inserted into the message body of an HTTP POST message. The default value is `/`.
- **MIME type** – Indicate the MIME type that the HTTP POST message uses. The default value is `application/json`. However, any value is valid because you can create and then specify your own MIME type.
- **Max retries** – Specify the maximum number of times Elastic Beanstalk attempts to send the message to the Amazon SQS queue before moving the message to the dead letter queue. The default value is **`10`**. You can specify a value between **`1`** and **`1000`**.
- **HTTP connections** – Specify the maximum number of concurrent connections that the daemon can make to any application(s) within an Amazon EC2 instance. The default is **`50`**. You can specify a value between **`1`** and **`100`**.
- **Connection timeout** – Indicate the amount of time, in seconds, to wait for successful connections to an application. The default value is **`5`**. You can specify a value between **`1`** and **`60`** seconds.
- **Inactivity timeout** – Indicate the amount of time, in seconds, to wait for a response on an existing connection to an application. The default value is **`180`**. You can specify a value between **`1`** and **`1800`** seconds.
- **Visibility timeout** – Indicate the amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, the message is again made visible in the queue for any other daemon to read. For worker environments created after May 27, 2014, the default value is **`300`** seconds. For worker environments created before May 27, 2014, the default value is **`30`** seconds. You can specify a value between **`0`** and **`43200`**. We recommend that you specify a value that is higher than you expect your application requires to process messages.

- **Error visibility timeout** – Indicate the amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after an attempt to process it fails with an explicit error. You can specify a value between **`0`** and **`43200`**.
- **Retention period** – Indicate the amount of time, in seconds, a message is valid and will be actively processed. The default value is **`345600`**. You can specify a value between **`60`** and **`1209600`**.

CLI

To launch an environment with a worker environment tier

- Create a worker environment.

```
$ aws elasticbeanstalk create-environment --application-name my-app --envir
onment-name my-env --tier Worker --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "MimeType",
        "Value": "application/json"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "HttpConnections",
        "Value": "75"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "ConnectTimeout",
        "Value": "10"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "InactivityTimeout",
        "Value": "10"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "VisibilityTimeout",
        "Value": "300"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "RetentionPeriod",
        "Value": "345600"
    },
    {
        "Namespace": "aws:elasticbeanstalk:sqsd",
        "OptionName": "MaxRetries",
        "Value": "50"
    }
]
```

API

For information about all the option values you can pass, see Option Values.

To launch an environment with a worker environment tier

- Call `CreateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv2`
  - *`VersionLabel`* = `Version2`
  - *`Description`* = `description`
  - *`TemplateName`* = `MyConfigTemplate`
  - *`ApplicationName`* = `SampleApp`
  - *`Tier`* = `Worker`
  - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.1.OptionName`* = `IamInstanceProfile`
  - *`OptionSettings.member.1.Value`* = `ElasticBeanstalkProfile`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.2.OptionName`* = `WorkerQueueURL`
  - *`OptionSettings.member.2.Value`* = `sqsd.elasticbeanstalk.us-west-2.amazon.com`
  - *`OptionSettings.member.3.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.3.OptionName`* = `HttpPath`
  - *`OptionSettings.member.3.Value`* = `/`
  - *`OptionSettings.member.4.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.4.OptionName`* = `MimeType`
  - *`OptionSettings.member.4.Value`* = `application/json`
  - *`OptionSettings.member.5.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.5.OptionName`* = `HttpConnections`
  - *`OptionSettings.member.5.Value`* = `75`
  - *`OptionSettings.member.6.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.6.OptionName`* = `ConnectTimeout`
  - *`OptionSettings.member.6.Value`* = `10`
  - *`OptionSettings.member.7.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.7.OptionName`* = `InactivityTimeout`
  - *`OptionSettings.member.7.Value`* = `10`
  - *`OptionSettings.member.8.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.8.OptionName`* = `VisibilityTimeout`
  - *`OptionSettings.member.8.Value`* = `300`
  - *`OptionSettings.member.9.Namespace`* = `aws:elasticbeanstalk:sqsd`
  - *`OptionSettings.member.9.OptionName`* = `RetentionPeriod`
  - *`OptionSettings.member.9.Value`* = `345600`
  - *`OptionSettings.member.10.Namespace`* = `aws:elasticbeanstalk:sqsd`

- *OptionSettings.member.10.OptionName* = MaxRetries
- *OptionSettings.member.10.Value* = 50

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=ElasticBeanstalkProfile
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-west-2.amazon.com
&OptionSettings.member.3.Namespace=aws%3elasticbeanstalk%3sqsd
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.5.OptionName=HttpConnections
&OptionSettings.member.5.Value=75
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.6.OptionName=ConnectTimeout
&OptionSettings.member.6.Value=10
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.7.OptionName=InactivityTimeout
&OptionSettings.member.7.Value=10
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.8.OptionName=VisibilityTimeout
&OptionSettings.member.8.Value=300
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.9.OptionName=RetentionPeriod
&OptionSettings.member.9.Value=345600
&amp;OptionSettings.member.10.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&amp;OptionSettings.member.10.OptionName=MaxRetries
&amp;OptionSettings.member.10.Value=50
&AuthParams
```

[1] For worker environments created after May 27, 2014

[2] For worker environments created after February 17, 2015

Update an Environment

Topics

- Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83)
- Canceling Environment Configuration Updates and Application Deployments (p. 89)
- Upgrading the Elastic Beanstalk Environment's Platform Version (p. 90)
- Migrating Your Application from a Legacy Container Type (p. 91)

Deploying Applications to AWS Elastic Beanstalk Environments

You can deploy existing Elastic Beanstalk application versions to existing environments. You may want to do this if, for instance, you need to roll back to a previous version of your application. This section describes how you can use the AWS Management Console, the CLI, or APIs to deploy an existing Elastic Beanstalk application version to an existing environment.

> **Note**
> When you deploy application versions as described in this topic, all Amazon EC2 instances are taken out of service at the same time by default. However, you can configure a load-balancing, autoscaling environment to deploy application versions to batches each consisting of a specified number of Amazon EC2 instances. You also have the option to deploy application versions with zero downtime by performing a CNAME swap. For detailed information about configuring and deploying application versions in batches of Amazon EC2 instances when you upload an application version, see Deploying Application Versions in Batches (Rolling Deployments) (p. 174). For more information about performing a CNAME swap, see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

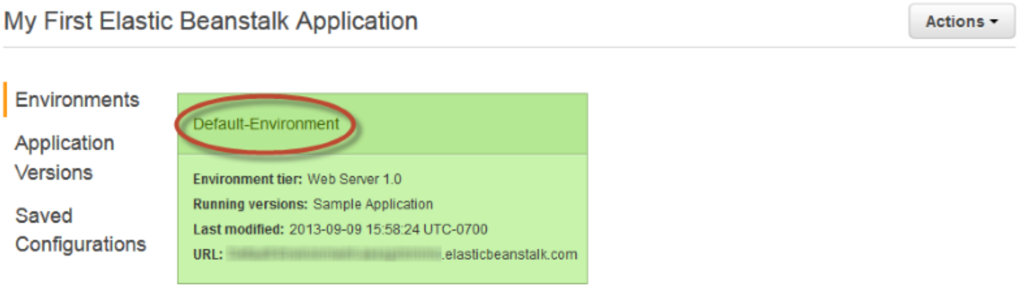
This topic describes how to deploy either new or existing application versions to existing environments. If you want to launch an environment with the configuration settings of an existing environment, you can use a saved environment configuration. For more information, see Saving Environment Configuration Settings (p. 101).

AWS Management Console

You can configure batched application deployment settings when you deploy a new application version to an existing environment. Those settings apply to current and future application version deployments. You cannot configure batched application deployment settings at the time that you deploy an existing application version to an existing environment. This section contains procedures for both scenarios.

To deploy a new application version to an existing environment

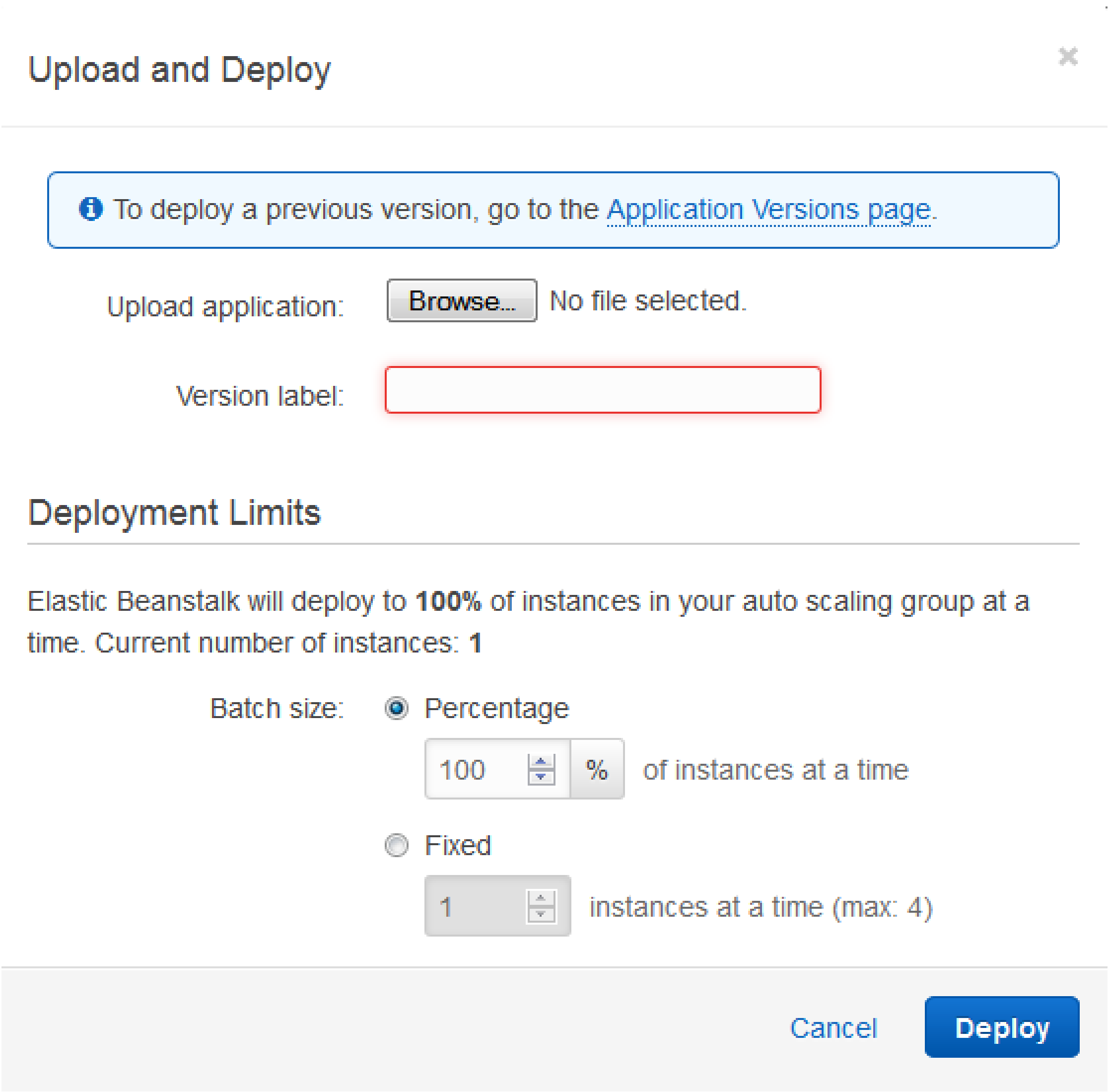
1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that want to work in.
3. From the Elastic Beanstalk console applications page, click the name of the environment that you want to deploy an existing application version to.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. Click **Upload and Deploy**,
5. If you are deploying an application version to a load-balancing, autoscaling environment, skip to the next step.

   For a single-instance environment, do the following:



Upload and Deploy

To deploy a previous version, go to the Application Versions page.

Upload application: Browse... No file selected.

Version label:

Deployment Limits

Elastic Beanstalk will deploy to **100%** of instances in your auto scaling group at a time. Current number of instances: **1**

Batch size: ● Percentage

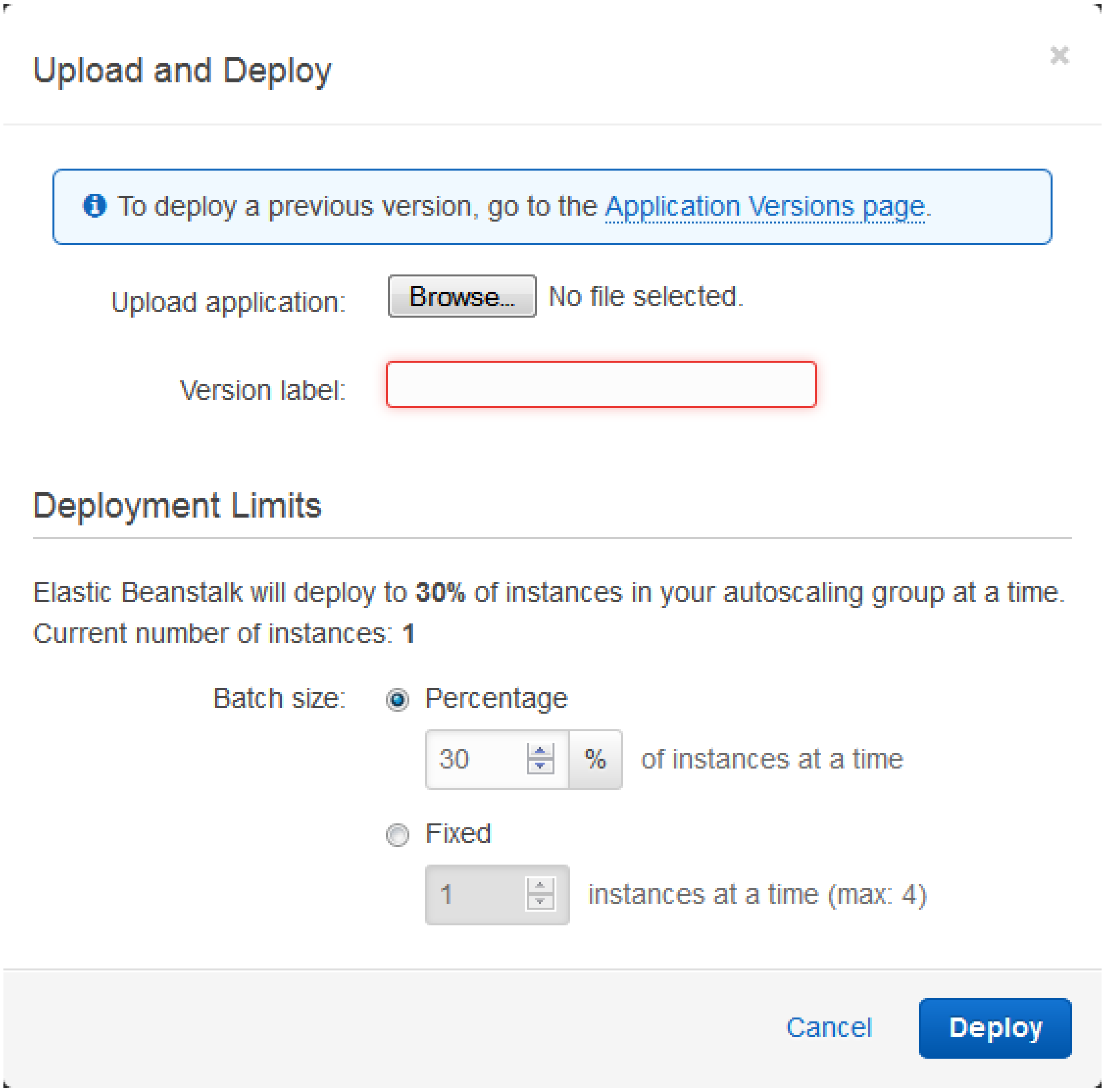
100 % of instances at a time

○ Fixed

1 instances at a time (max: 4)

   1. Click **Browse** to select the application source bundle for the application version that you want to deploy.
   2. (Optional) For **Version label**, type a new description or change any description that Elastic Beanstalk may have automatically created.
   3. When you are finished, click **Deploy**.

6. For a load-balancing, autoscaling environment, do the following:



Upload and Deploy

To deploy a previous version, go to the Application Versions page.

Upload application: Browse... No file selected.

Version label:

Deployment Limits

Elastic Beanstalk will deploy to **30%** of instances in your autoscaling group at a time.
Current number of instances: **1**

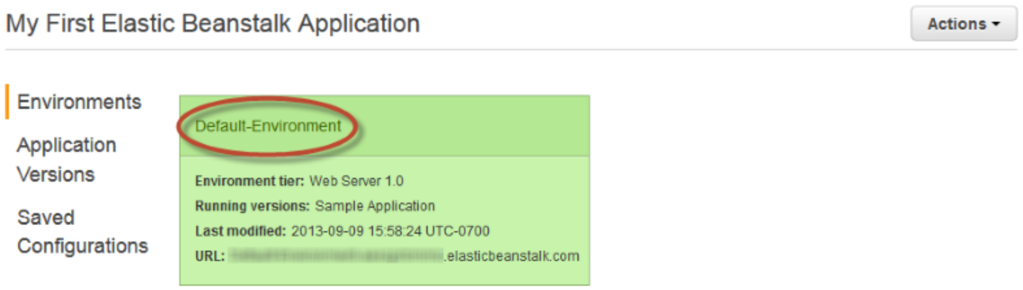
Batch size: ● Percentage
30 % of instances at a time

○ Fixed
1 instances at a time (max: 4)

1. Click **Browse** to select the application source bundle for the application version you want to deploy.
2. (Optional) For **Version label**, type a new description or change any description that Elastic Beanstalk may have automatically created.
3. Next to **Batch size**, click **Percentage** or **Fixed**, and then enter a percentage or fixed number of instances to which you want to deploy the new application version to at any given time.
4. When you are finished, click **Deploy**.

To deploy an existing application version to an existing environment

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that want to work in.
3. From the Elastic Beanstalk console applications page, click the name of the environment that you want to deploy an existing application version to.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. Click **Upload and Deploy** to view a list of all applications versions that you can deploy for this environment. The listed application versions are associated with this Elastic Beanstalk application.
5. Select the application version that you want to deploy, and then click **Deploy**.
6. Verify that you are deploying the right application version in the right environment, and then click **Deploy**.

Elastic Beanstalk now deploys your file to your Amazon EC2 instances. You will see the environment start the update process. If you specified a health check URL, there's an application health check when the deployment is complete. The environment returns to green when the application responds to the health check. For more information, see Basic Health Reporting (p. 247). If you need to deploy an application version with zero downtime, see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

If you view the environment dashboard, you can verify the application version.

CLI

You can configure batched application deployment settings when you deploy a new application version to an existing environment. Those settings apply to current and future application version deployments. You cannot configure batched application deployment settings at the time that you deploy an existing application version to an existing environment. This section contains procedures for both scenarios.

To deploy an existing application version to an existing environment

1. Make sure your application version exists.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app --version-label v1
```

2. Update your environment with your existing application version.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --version-label v1
```

3. Determine if the environment is Green and Ready.

```
$ aws elasticbeanstalk describe-environments --environment-names my-env
```

To configure batched application version deployments using the AWS CLI

- Change how application versions are applied to your environment.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:command",
        "OptionName": "BatchSize",
        "Value": "50"
    },
    {
        "Namespace": "aws:elasticbeanstalk:command",
        "OptionName": "BatchType",
        "Value": "Percentage"
    }

]
```

-->

API

To deploy an existing application version to an existing environment

1. Call `UpdateEnvironment` with the following parameters:

   - *`EnvironmentName`* = `SampleAppEnv`
   - *`VersionLabel`* = `FirstRelease`
   - *`Description`* = `description`

   #### Example

   ```
   https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=MySampleAppEnv
   &Description=description
   &VersionLabel=FirstRelease
   &Operation=UpdateEnvironment
   &AuthParams
   ```

2. Call `DescribeEnvironments` with the following parameter:

   - *`EnvironmentName`* = `SampleAppEnv`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&Operation=DescribeEnvironments
&AuthParams
```

To edit rolling updates using the API

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`VersionLabel`* = `Version2`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.1.OptionName`* = `BatchSize`
  - *`OptionSettings.member.1.Value`* = `50`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.2.OptionName`* = `BatchSizeType`
  - *`OptionSettings.member.2.Value`* = `Percentage`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&VersionLabel=Version2
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.1.OptionName=BatchSize
&OptionSettings.member.1.Value=50
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.2.OptionName=BatchSizeType
&OptionSettings.member.2.Value=Percentage
&Operation=UpdateEnvironment
&AuthParams
```

Deploying a new Application Version with Zero Downtime (CNAME swap)

Because Elastic Beanstalk performs an in-place update when you update your application versions, you will experience some downtime. However, it is possible to avoid this downtime by swapping the CNAMEs for your environments. To do the swap, you need to upload the updated application version and then create a new environment running that version. This section walks you through how to perform a CNAME swap using the AWS Management Console, the command line interface, or APIs.

Although using Elastic Beanstalk to attach an Amazon RDS database to your environment can help you avoid downtime in accessing your application, you might lose data during the CNAME swap process. This can happen as the application continues to generate data between the time the original database is deleted and a new database becomes available for the new environment. Taking a snapshot of the original database (as recommended in the instructions that follow) only saves some data because the snapshot will not contain any newly generated data.

To avoid that, manage your Amazon RDS database from the Amazon RDS console, CLI, or API instead of Elastic Beanstalk. That way, the database is unaffected by what you do to your Elastic Beanstalk environments. If you manage the database using Amazon RDS instead of managing it using Elastic Beanstalk, during a CNAME swap from one environment to another, the original environment continues to read from and write to the existing database until you delete the environment. When traffic stops flowing to the database after the CNAME swap, you can delete the original environment without losing data.

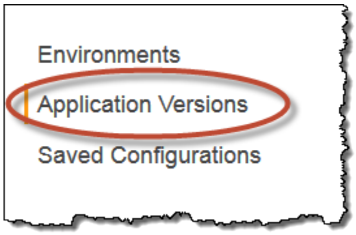
Depending on your needs, using Amazon RDS to manage your database also lets you take advantage of other Amazon RDS features and configuration options. For example, you might want to configure your new environment to bind to a read replica and then promote the read replica to a database instance. For more information about Amazon RDS, go to the Amazon RDS User Guide at What is Amazon Relational Database Service?.

> **Note**
> If you have created an Alias record to map your root domain to your Elastic Load Balancer using Amazon Route 53, then after you have created you new environment, you will need to change your resource record set to map your root domain to the Elastic Load Balancer in your new environment. For instructions on how to change your existing resource record set, go to Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295).

AWS Management Console

To deploy with zero downtime

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that has the application that you want to work with.
3. From the Elastic Beanstalk console applications page, click the name of the application to which you want to add a new application version.
4. In the navigation pane, click **Application Versions**.



5. Click **Upload**.

   a. Enter a label for this version in the **Version Label** field.

   b. Enter a brief description for this version in the **Description** field.

   c. Enter a label for this version in the **Version label** field.

   d. (Optional) Enter a brief description for this version in the **Description** field.

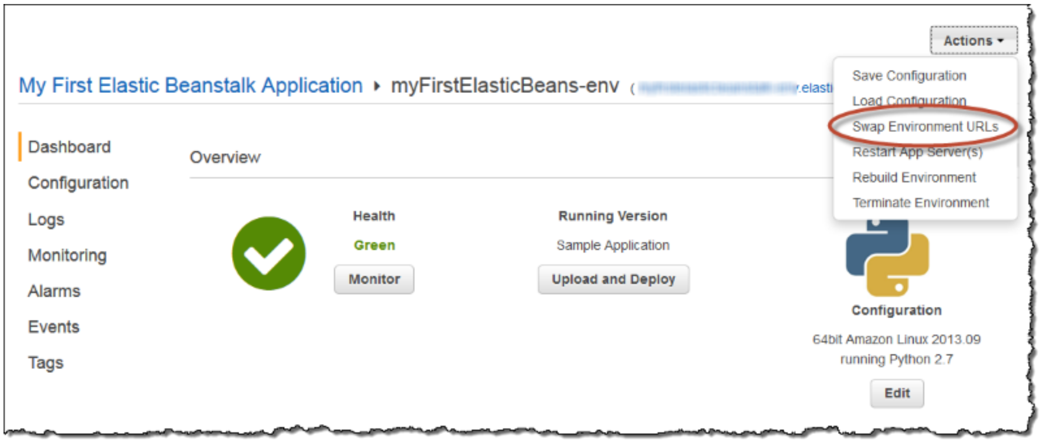
   e. Click **Browse** to specify the location of the application version (`.war` or `.zip` file).

      > **Note**
      > Elastic Beanstalk supports only a single `.war` file for a Java application version and only a single `.zip` file for other applications. The file size limit is 512 MB.

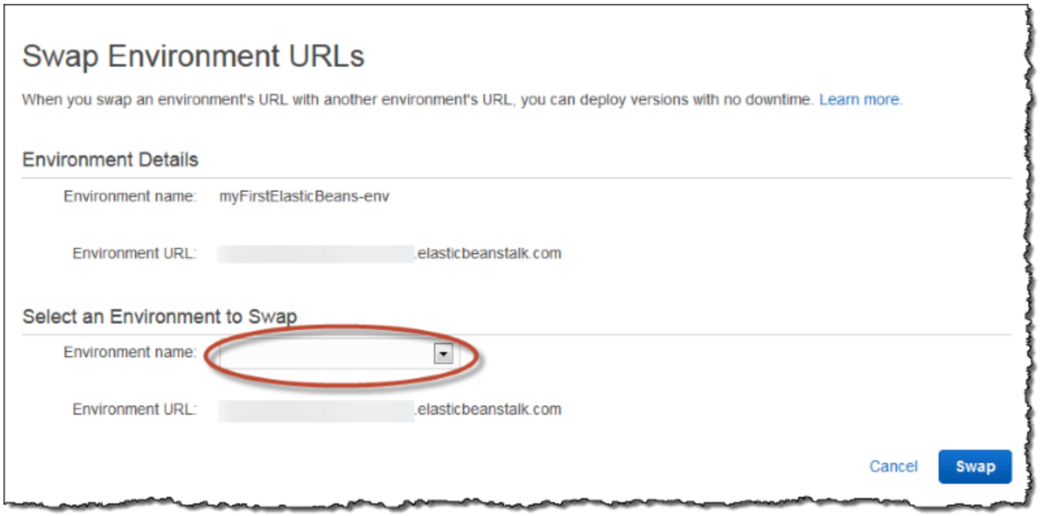
   The file you specified is associated with your application; you'll deploy the new application version to a new environment.
6. Save the configuration for the live environment by opening the environment's dashboard, clicking **Actions** and then selecting **Save Configuration**.

For more information, see Saving Environment Configuration Settings (p. 101). You'll use this configuration to create a new environment that will run the updated application version. Note that saved configurations do not include database information. When you delete an environment, you delete the data from any database instance you created in that environment. You can, however, create a snapshot of your current database instance and then use it as the basis for a new DB instance when you create a new environment in the next step. For more information, see Creating a DB Snapshot in the Amazon Relational Database Service User Guide.

7. Launch a new environment with your new application version and saved configuration. For more information, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
8. Verify that your new environment is ready by viewing its dashboard. If the environment has an error, view the events and logs for the environment in order to troubleshoot any errors.
9. From the new environment's dashboard, click **Actions** and then select **Swap Environment URLs**.



10. From the **Environment name** drop-down list, select the current live environment name in order to use that environment's URL for the new environment.



Swap Environment URLs

When you swap an environment's URL with another environment's URL, you can deploy versions with no downtime. Learn more.

Environment Details

Environment name: myFirstElasticBeans-env

Environment URL: [redacted].elasticbeanstalk.com

Select an Environment to Swap

Environment name:

Environment URL: [redacted].elasticbeanstalk.com

11. Click **Swap**.
12. After Elastic Beanstalk completes the swap operation, verify that the new environment responds when you try to connect to the old environment URL. However, do not terminate your old environment until the DNS changes have been propagated and your old DNS records expire. DNS servers do not necessarily clear old records from their cache based on the time to live (TTL) you set on your DNS

records. Therefore, we recommend that you wait an additional period of time after the TTL before you terminate your environment.

CLI

You can use the command line interface to deploy a new application with zero downtime. The following steps walk you through creating a configuration template, launching a new environment with the new application version using the configuration template, and swapping the environment CNAMEs. You can also use the following steps to perform configuration changes with zero downtime.

To deploy with zero downtime

1. Check that your current environment is not updating.

   ```
   $ aws elasticbeanstalk describe-environments --environment-names my-env
   ```

2. Verify that your new application version exists.

   ```
   $ aws elasticbeanstalk describe-application-versions --application-name my-
   app --version-label v1
   ```

3. Create a configuration template from the current environment.

   ```
   $ aws elasticbeanstalk create-configuration-template --application-name my-
   app --template-name v1 --environment-id e-wtsp2qejrp
   ```

4. Launch a new environment for the new application version and template.

   ```
   $ aws elasticbeanstalk create-environment --application-name my-app --tem
   plate-name v1 --version-label v1 --environment-name v1clone
   ```

   Note
   Environment names must be between 4 and 23 characters.

5. Check that your current environment is not updating.

   ```
   $ aws elasticbeanstalk describe-environments --environment-names my-env
   ```

   If the new environment does not come up Green and Ready, you should decide if you want to retry the operation or leave the environment in its current state for investigation. Make sure to terminate the environment after you are finished, and clean up any unused resources.

   Note
   You can adjust the timeout period if the environment doesn't launch in a reasonable time.

6. After the new environment is Green and Ready, swap the environment CNAMEs.

   ```
   $ aws elasticbeanstalk swap-environment-cnames --source-environment-name v1
    --destination-environment-name v1clone
   ```

7. After Elastic Beanstalk completes the swap operation, verify that the new environment responds when you try to connect to the old environment URL. However, do not terminate your old environment

until the DNS changes have been propagated and your old DNS records expire. DNS servers do not necessarily clear old records from their cache based on the time to live (TTL) you set on your DNS records. Therefore, we recommend that you wait an additional period of time after the TTL before you terminate your environment.

```
$ aws elasticbeanstalk terminate-environment --environment-name v1
```

API

To deploy with zero downtime

1. Check your current environment to make sure it is Green and Ready.
   Call `DescribeEnvironments` with the following parameter:

   - *EnvironmentID* = `e-pyumupm7mph`

   #### Example

   ```
   https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentID=e-pyumupm7mph
   &Operation=DescribeEnvironments
   &AuthParams
   ```

2. Verify that your new application version exists.
   Call `DescribeApplicationVersions` with the following parameters:

   - *ApplicationName* = `SampleApp`
   - *VersionLabel* = `Version2`

   #### Example

   ```
   https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
   &VersionLabel=Version2
   &Operation=DescribeApplicationVersions
   &AuthParams
   ```

3. Create a configuration template from the current environment.
   Call `CreateConfigurationTemplate` with the following parameters:

   - *EnvironmentID* = `e-pyumupm7mph`
   - *ApplicationName* = `SampleApp`
   - *TemplateName* = `MyConfigTemplate`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&EnvironmentID=e-pyumupm7mph
&Operation=CreateConfigurationTemplate
&AuthParams
```

4. Launch a new environment for the new application version and template.

   Call `CreateEnvironment` with the following parameters:

   - *`ApplicationName`* = `SampleApp`
   - *`VersionLabel`* = `Version2`
   - *`EnvironmentName`* = `SampleAppEnv2`

     Note

     Environment names must be at least 4 characters, and less than or equal to 23 characters.
   - *`TemplateName`* = `MyConfigTemplate`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentID=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Operation=CreateEnvironment
&AuthParams
```

5. Determine if the new environment is Green and Ready.

   Call `DescribeEnvironments` with the following parameters:

   - *`EnvironmentID`* = `e-eup272zdrw`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentID=e-eup272zdrw
&Operation=DescribeEnvironments
&AuthParams
```

If the new environment does not come up Green and Ready, you should decide if you want to retry the operation or leave the environment in its current state for investigation. Make sure to terminate the environment after you are finished, and clean up any unused resources.

Note

You can adjust the timeout period if the environment doesn't launch in a reasonable time.

6. After the new environment is Green and Ready, swap the environment CNAMEs.

   Call `SwapEnvironmentCNAMEs` with the following parameters:

   - *`SourceEnvironmentID`* = `e-pyumupm7mph`
   - *`DestinationEnvironmentID`* = `e-eup272zdrw`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?SourceEnvironmentID=e-pyu
mupm7mph
&DestinationEnvironmentIDe=e-eup272zdrw
&Operation=SwapEnvironmentCNAMEs
&AuthParams
```

7. After Elastic Beanstalk completes the swap operation, verify that the new environment responds when you try to connect to the old environment URL. However, do not terminate your old environment until the DNS changes have been propagated and your old DNS records expire. DNS servers do not necessarily clear old records from their cache based on the time to live (TTL) you set on your DNS records. Therefore, we recommend that you wait an additional period of time after the TTL before you terminate your environment.

   Call `TerminateEnvironment` with the following parameters:

   - *`EnvironmentID`* = `e-pyumupm7mph`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentId=e-pyumupm7mph
&Operation=TerminateEnvironment
&AuthParams
```

Canceling Environment Configuration Updates and Application Deployments

You can cancel in-progress updates that are triggered by environment configuration changes. You can also cancel the deployment of a new application version in progress. For example, you might want to cancel an update if you decide you want to continue using the existing environment configuration instead of applying new environment configuration settings. Or, you might realize that the new application version that you are deploying has problems that will cause it to not start or not run properly. By canceling an environment update or application version update, you can avoid waiting until the update or deployment process is done before you begin a new attempt to update the environment or application version.

> **Note**
> During the cleanup phase in which old resources that are no longer needed are removed, after the last batch of instances has been updated, you can no longer cancel the update.

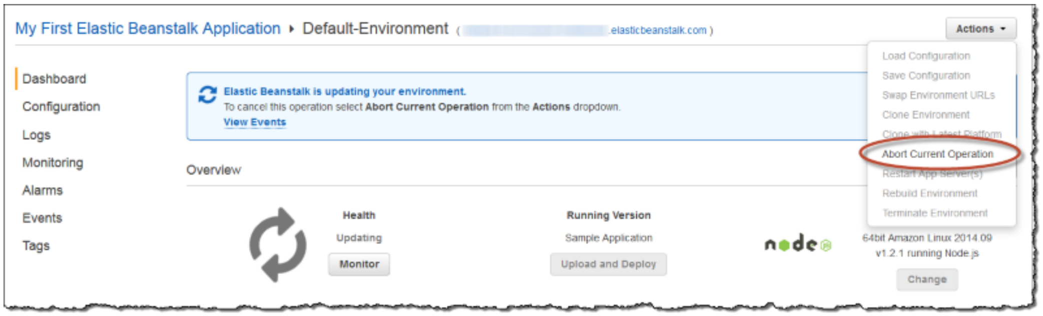
Elastic Beanstalk performs the rollback the same way that it performed the last successful update. For example, if you have time-based rolling updates enabled in your environment, then Elastic Beanstalk will wait the specified pause time between rolling back changes on one batch of instances before rolling back changes on the next batch. Or, if you recently turned on rolling updates, but the last time you successfully updated your environment configuration settings was without rolling updates, Elastic Beanstalk will perform the rollback on all instances simultaneously.

You cannot stop Elastic Beanstalk from rolling back to the previous environment configuration once it begins to cancel the update. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails. For application version deployments, canceling the deployment simply stops the deployment; some instances will have the new application version and others will continue to run the existing application version. You can deploy the same or another application version later.

For more information about rolling updates, see Updating Elastic Beanstalk Environments with Rolling Updates (p. 169). For more information about batched application version deployments, see Deploying Application Versions in Batches (Rolling Deployments) (p. 174).

To cancel an update

- On the environment dashboard, click **Actions**, and then click **Abort Current Operation**.



Elastic Beanstalk is updating your environment.
To cancel this operation select **Abort Current Operation** from the **Actions** dropdown.
View Events

Upgrading the Elastic Beanstalk Environment's Platform Version

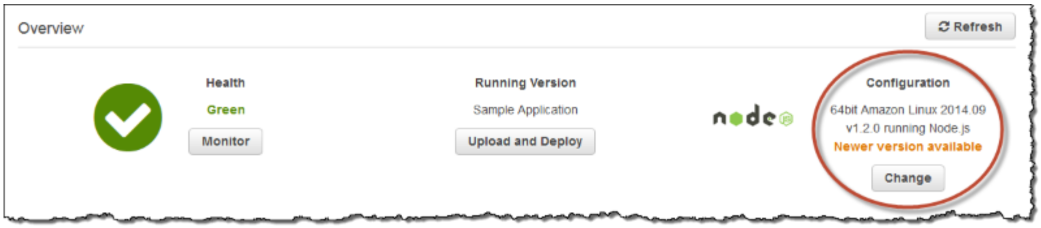
When a new version of your environment's the section called “Supported Platforms” (p. 24) is available, Elastic Beanstalk shows a message in the environment management console (p. 51) and makes the **Change** button available. If you've previously created an environment using an older version of the same configuration, or upgraded your environment from an older configuration, you can also use the **Change** button to revert to a previous configuration version.

Configurations are specific to major language and tool versions. When a configuration is updated, the language and tools will not change major versions. For example, *Java 7 Tomcat 7* and *Java 8 Tomcat 8* are two different configurations of the *Java with Tomcat* platform. You cannot perform a platform update between configurations, only between versions of the same configuration, such as *Java 8 Tomcat 8 version 1.4.5* and *Java 8 Tomcat 8 version 2.0.0*.

AWS Management Console

To upgrade the platform

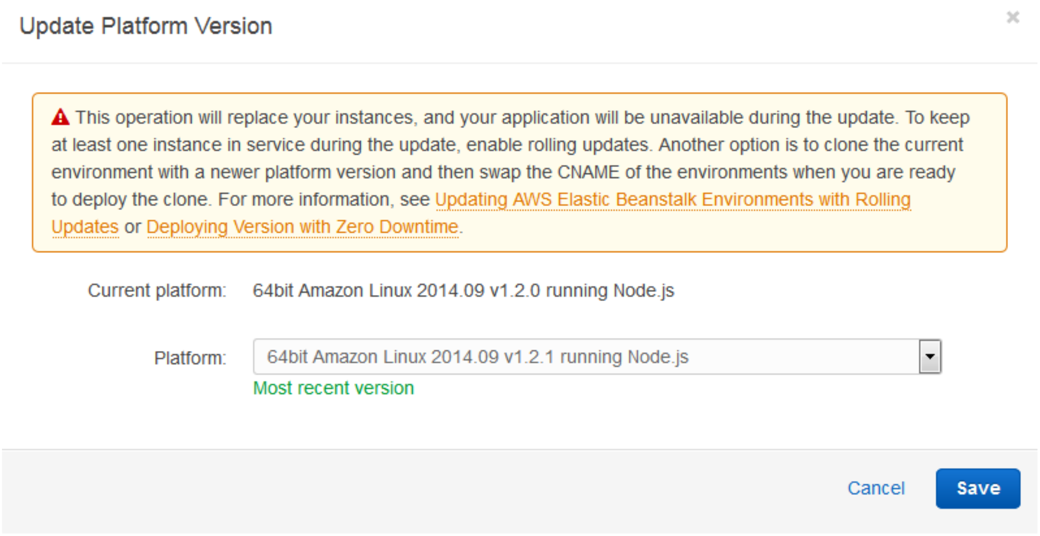
1. Navigate to the management console (p. 51) for your environment.
2. In the **Overview** section, under **Configuration**, click **Change**.



3. On the **Update Platform Version** page, click **Platform**, and then click the platform version that you want the environment to use.

Note

The availability of your application during the platform version upgrade depends on whether you have rolling updates enabled. The **Update Platform Version** window explains what will happen and what procedures you can follow to minimize downtime. Elastic Beanstalk displays a window similar to one of the following:

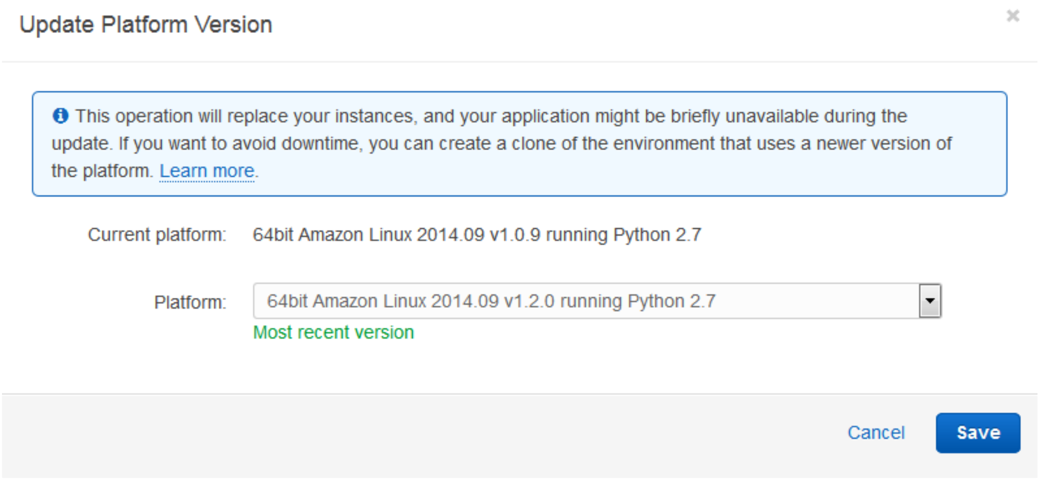
- 

  Update Platform Version

  ⚠ This operation will replace your instances, and your application will be unavailable during the update. To keep at least one instance in service during the update, enable rolling updates. Another option is to clone the current environment with a newer platform version and then swap the CNAME of the environments when you are ready to deploy the clone. For more information, see Updating AWS Elastic Beanstalk Environments with Rolling Updates or Deploying Version with Zero Downtime.

  Current platform: 64bit Amazon Linux 2014.09 v1.2.0 running Node.js

  Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Node.js
  Most recent version

- 

  Update Platform Version

  ℹ This operation will replace your instances, and your application might be briefly unavailable during the update. If you want to avoid downtime, you can create a clone of the environment that uses a newer version of the platform. Learn more.

  Current platform: 64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7

  Platform: 64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7
  Most recent version

4. After you choose the platform version you want to use, click **Save**.

Migrating Your Application from a Legacy Container Type

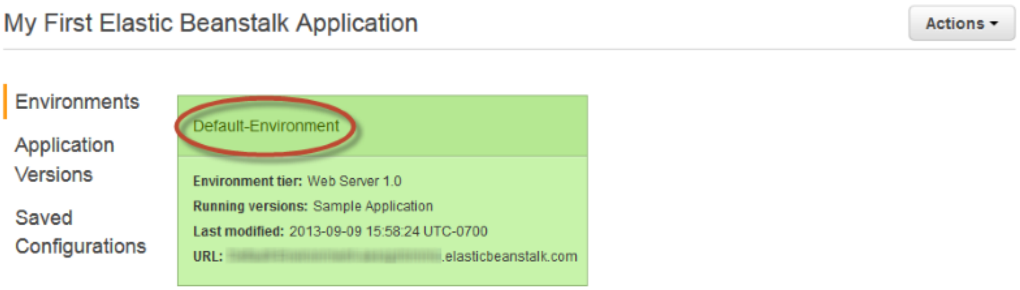
If you have deployed an Elastic Beanstalk application that uses a legacy container type, you should migrate your application to a new environment using a non-legacy container type so that you can get access to new features. If you are unsure whether you are running your application using a legacy container type, you can check in the Elastic Beanstalk console. For instructions, see To check if you are using a legacy container type (p. 92).

Why are some container types marked legacy?

Some older platform configurations do not support the latest Elastic Beanstalk features. These configurations are marked **(legacy)** on the environment configuration page in the AWS Management Console.

To check if you are using a legacy container type

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to verify.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, view the **Configuration** name.
   Your application is using a legacy container type if you see **(legacy)** next to the configuration.

To migrate your application

1. Deploy your application to a new environment. For instructions, go to Launching a New AWS Elastic Beanstalk Environment (p. 55).
2. If you have an Amazon RDS DB Instance, update your database security group to allow access to your EC2 security group for your new environment. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information about configuring your EC2 security group, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Swap your environment URL. For instructions, go to Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).
4. Terminate your old environment. For instructions, go to Terminate an Environment (p. 95).

> **Note**
> If you use AWS Identity and Access Management (IAM) then you will need to update your policies to include AWS CloudFormation and Amazon RDS (if applicable). For more information, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326).

Clone an Environment

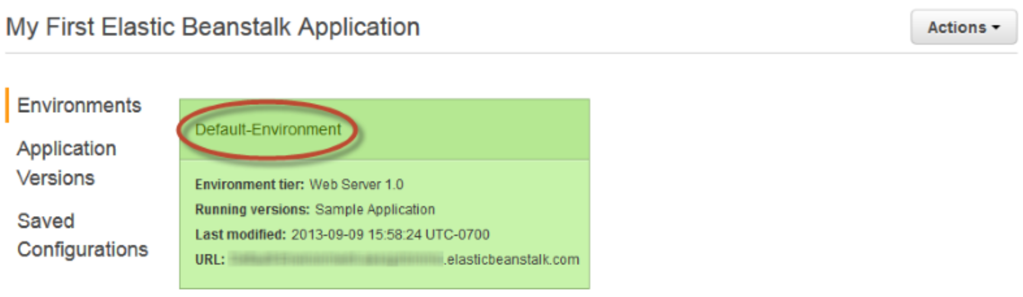
You can use an existing environment as the basis for a new environment by creating a clone of the existing environment. For example, you might want to create a clone so that you can use a newer version of the solution stack used by the original environment's platform. Elastic Beanstalk configures the clone with the same environment settings used by the original environment. By cloning an existing environment instead of creating a new environment, you do not have to manually configure option settings, environment variables, and other settings. Elastic Beanstalk also creates a copy of any AWS resource associated with the original environment. However, during the cloning process, Elastic Beanstalk does not copy data from Amazon RDS to the clone. After you have created the clone environment, you can modify environment configuration settings as needed.

> **Note**
> Elastic Beanstalk does not include any unmanaged changes to resources in the clone. Changes to AWS resources that you make using tools other than the Elastic Beanstalk management console, command-line tools, or API are considered unmanaged changes.

AWS Management Console

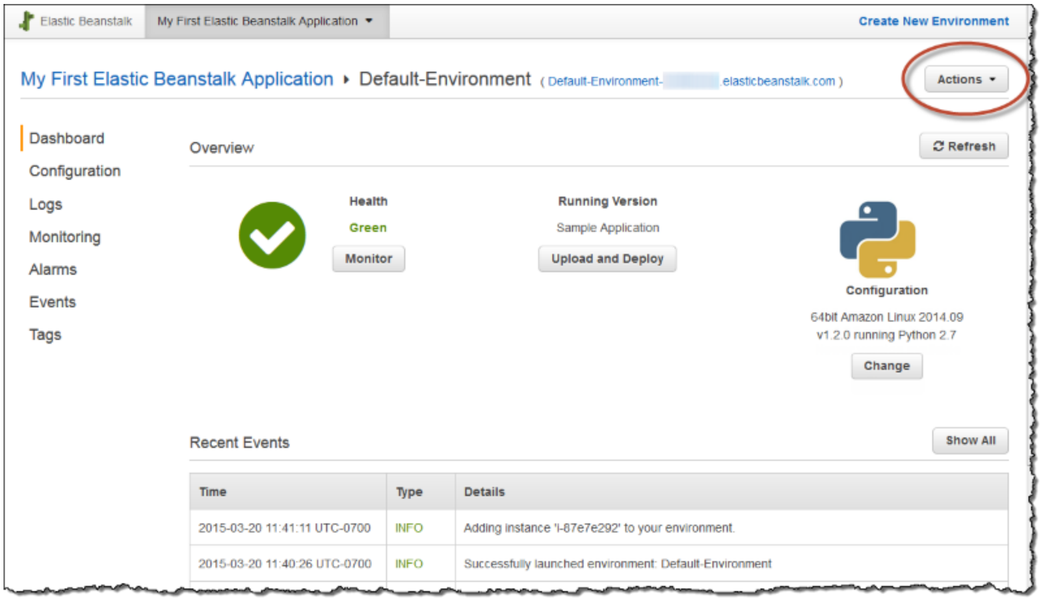
To clone an environment

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work with.
3. On the Elastic Beanstalk console applications page, click the name of the application, and then the name of the environment that you want to clone.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. On the environment dashboard, click **Actions**, and then do one of the following:
   - Click **Clone Environment** if you want to clone the environment without any changes to the solution stack version.
   - Click **Clone with Latest Platform** if you want to clone the environment, but with a newer version of the original environment's solution stack.



My First Elastic Beanstalk Application ▸ Default-Environment (Default-Environment-[redacted].elasticbeanstalk.com)

Health: Green

Running Version: Sample Application

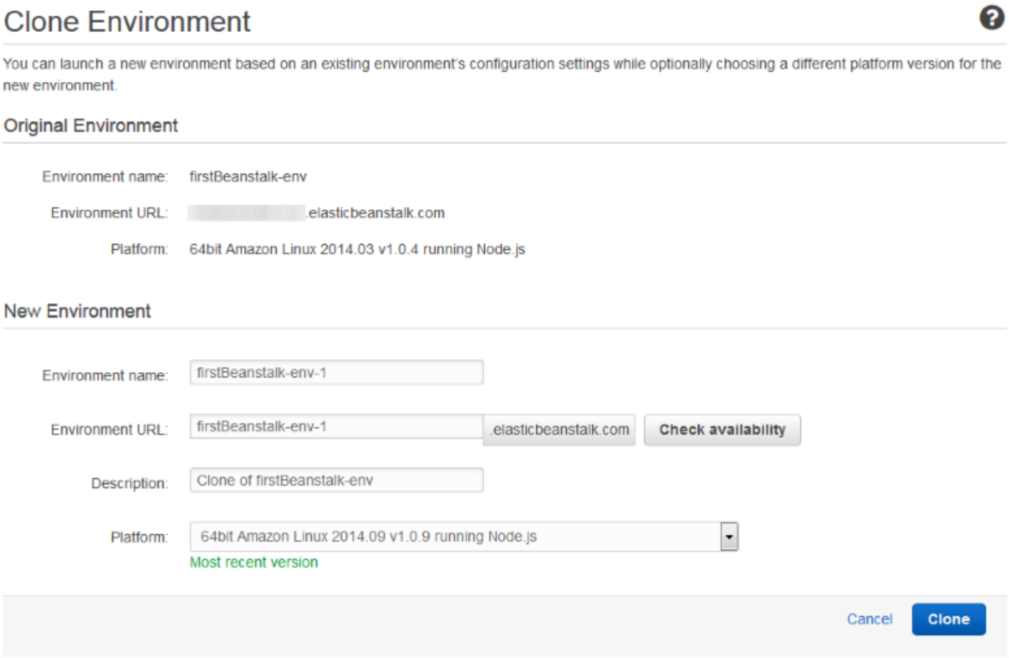
Configuration: 64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

Time	Type	Details
2015-03-20 11:41:11 UTC-0700	INFO	Adding instance 'i-87e7e292' to your environment.
2015-03-20 11:40:26 UTC-0700	INFO	Successfully launched environment: Default-Environment

5. On the **Clone Environment** page, review the information in the **Original Environment** section to verify that you chose the environment from which you want to create a clone.
6. In the **New Environment** section, you can optionally change the **Environment name**, **Environment URL**, **Description**, and **Platform** values that Elastic Beanstalk automatically set based on the original environment.

   ### Note

   For **Platform**, only solution stacks with the same language and web server configuration are shown. If a newer version of the solution stack used with the original environment is available, you will be prompted to update, but you cannot choose a different stack, even if it is for a different version of the same language. For more information, see Supported Platforms (p. 24).



Clone Environment

You can launch a new environment based on an existing environment's configuration settings while optionally choosing a different platform version for the new environment.

Original Environment

Environment name: firstBeanstalk-env

Environment URL: [redacted].elasticbeanstalk.com

Platform: 64bit Amazon Linux 2014.03 v1.0.4 running Node.js

New Environment

Environment name: firstBeanstalk-env-1

Environment URL: firstBeanstalk-env-1 .elasticbeanstalk.com

Description: Clone of firstBeanstalk-env

Platform: 64bit Amazon Linux 2014.09 v1.0.9 running Node.js

Most recent version

7. When you are ready, click **Clone**.

EB Command Line Interface (CLI)

To clone an environment using the EB CLI

- Create a new environment by cloning an existing environment.

  ```
  PROMPT> eb clone [environment_name] [--update]
  ```

  > **Note**
  > You can specify the environment name. If you don't provide *`environment_name`* as a command line parameter, EB CLI will use the default environment. The `--update` option clones the original environment to a new environment, but with the most recent solution stack available for the platform used by the original environment. Do not include the `--update` option if you want to use the same solution stack as the original environment. For more information, including other command line parameters for the `eb clone` command, see clone (p. 402).

Terminate an Environment

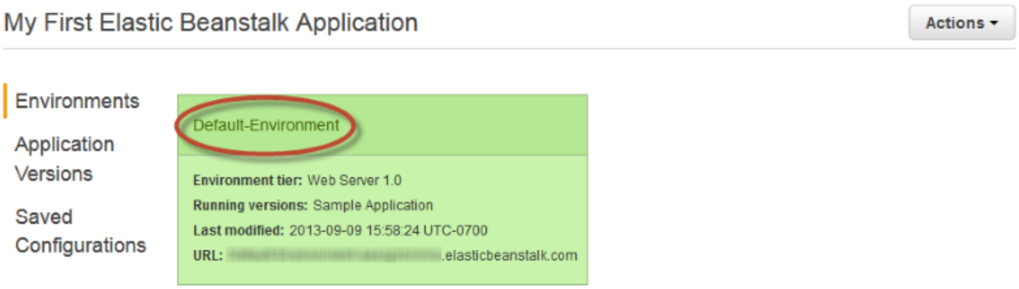
You can terminate a running environment using the AWS Management Console to avoid incurring charges for unused AWS resources. For more information about terminating an environment using the AWS Toolkit for Eclipse, see Terminating an Environment (p. 610).

> **Note**
> You can always launch a new environment using the same version later. If you have data from an environment that you would like to preserve, create a snapshot of your current database instance before you terminate the environment. You can later use it as the basis for new DB instance when you create a new environment. For more information, see Creating a DB Snapshot in the Amazon Relational Database Service User Guide.

AWS Management Console

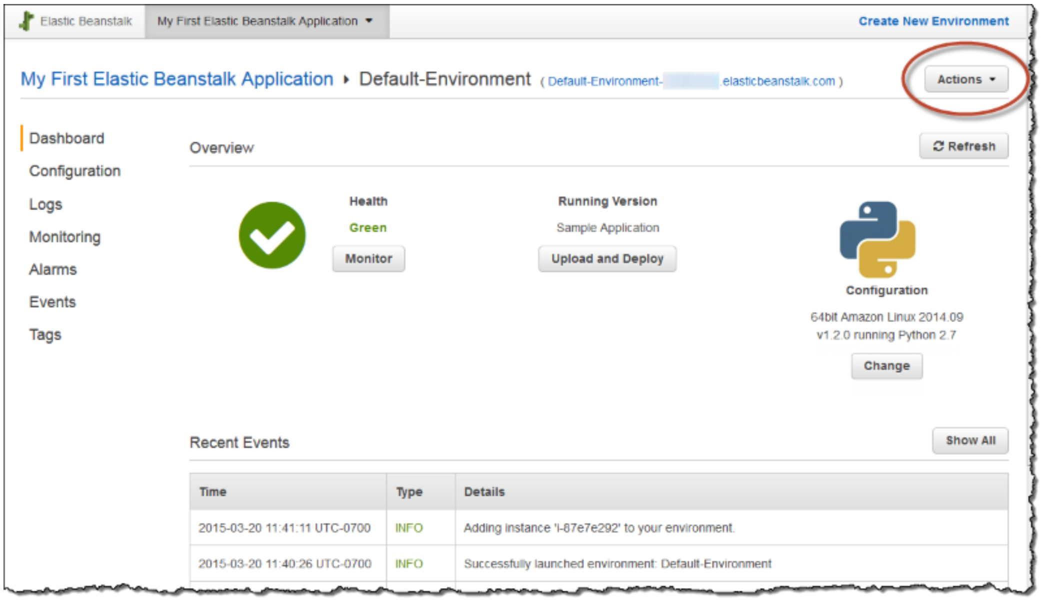
To terminate an environment

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that want to terminate.
3. From the Elastic Beanstalk console applications page, click the name of the environment that you want to terminate.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. Click **Actions** and the select **Terminate Environment**.



Time	Type	Details
2015-03-20 11:41:11 UTC-0700	INFO	Adding instance 'i-87e7e292' to your environment.
2015-03-20 11:40:26 UTC-0700	INFO	Successfully launched environment: Default-Environment

5. Confirm that you are terminating the correct environment and then click **Terminate**.

   Note
   When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

   It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

CLI

To terminate an environment

-
  ```
  $ aws elasticbeanstalk terminate-environment --environment-name my-env
  ```

API

To terminate an environment

- Call `TerminateEnvironment` with the following parameter:
  - *`EnvironmentName`* = `SampleAppEnv`

  Example

  ```
  https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
  &Operation=TerminateEnvironment
  &AuthParams
  ```

Tag an Environment

When you create a new environment, you can specify tags to categorize the environment. Tags can help you identify environments in cost allocation reports, especially if you have many to manage. You can use cost allocation reports to track your usage of AWS resources. The reports include both tagged and untagged resources, but they aggregate costs according to tags. You can also use tags to manage permissions at the resource level. When Elastic Beanstalk launches a new environment, it automatically applies tags to your Amazon EC2, Amazon RDS, and Auto Scaling resources. For more information about Amazon EC2 tags, including examples, see Tagging Your Amazon EC2 Resources in the *Amazon EC2 User Guide for Linux Instances*. For information specifically about how cost allocation reports use tags, see Use Cost Allocation Tags for Custom Billing Reports in the *AWS Billing and Cost Management User Guide*.

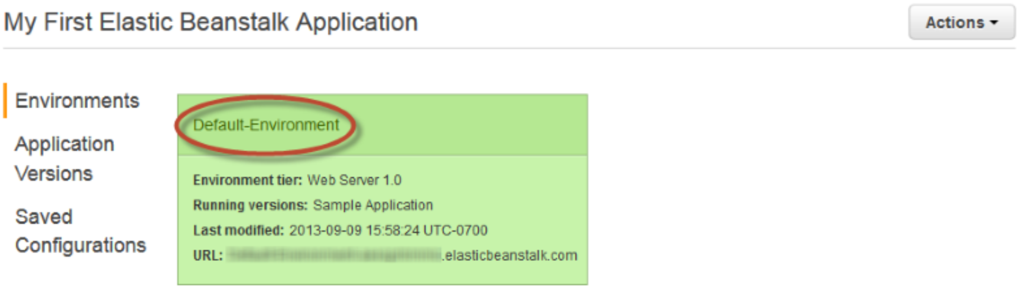
You can apply tags to an environment in the form of key-value pairs by using the console, the AWS Command Line Interface, or the CreateEnvironment API. For more information, see the AWS Management Console (p. 55) section of Launching a New AWS Elastic Beanstalk Environment (p. 55), create-environment in the AWS CLI Reference, or CreateEnvironment in the Elastic Beanstalk API Reference.

Enable Load Balancing

You can change your environment type to a single-instance or load-balancing, autoscaling environment by editing your environment's configuration. In some cases, you might want to change your environment type from one type to another. For example, let's say that you developed and tested an application in a single-instance environment in order to save costs. When your application is ready for production, you can change the environment type to a load-balancing, autoscaling environment so that it can scale to meet the demands of your customers.

To change an environment's type

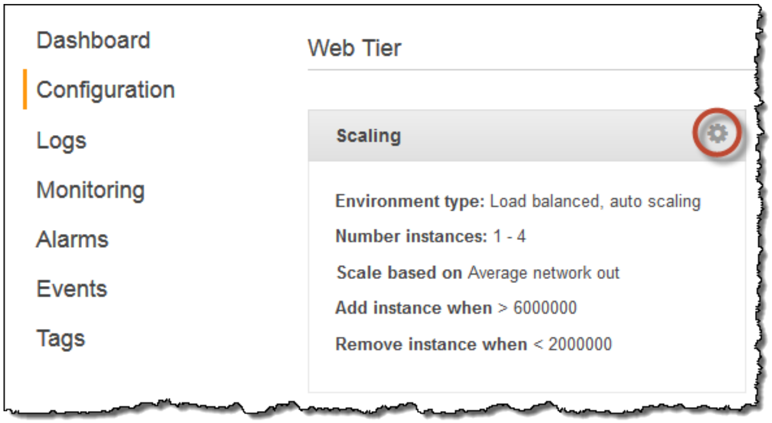
1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work with.
3. On the Elastic Beanstalk console applications page, click the name of the environment.



My First Elastic Beanstalk Application

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. In the **Overview** section of the environment dashboard, click **Edit**.
5. Click ⚙ for the **Scaling** settings.



Scaling

Environment type: Load balanced, auto scaling

Number instances: 1 - 4

Scale based on Average network out

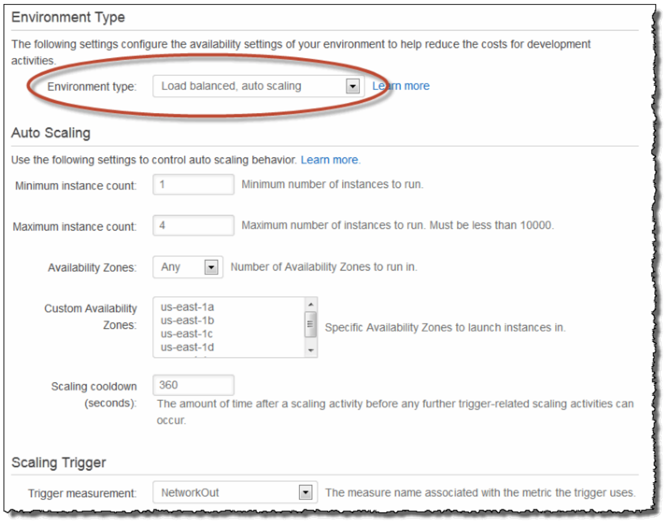
Add instance when > 6000000

Remove instance when < 2000000

6. In the **Environment Type** section, select the type of environment that you want.

   Note

   The single-instance environment type is available for nonlegacy containers only. For instructions on migrating from a legacy container, see Migrating Your Application from a Legacy Container Type (p. 91).



Environment Type

The following settings configure the availability settings of your environment to help reduce the costs for development activities.

Environment type: Load balanced, auto scaling Learn more

Auto Scaling

Use the following settings to control auto scaling behavior. Learn more.

Minimum instance count: 1 Minimum number of instances to run.

Maximum instance count: 4 Maximum number of instances to run. Must be less than 10000.

Availability Zones: Any Number of Availability Zones to run in.

Custom Availability Zones: us-east-1a
us-east-1b
us-east-1c
us-east-1d Specific Availability Zones to launch instances in.

Scaling cooldown (seconds): 360 The amount of time after a scaling activity before any further trigger-related scaling activities can occur.

Scaling Trigger

Trigger measurement: NetworkOut The measure name associated with the metric the trigger uses.

7. If your environment is in a VPC, select subnets to place ELB and EC2 instances in. Each availability zone that your application runs in must have both. See Using Elastic Beanstalk with Amazon VPC (p. 297) for details.
8. Click **Save**.

   Note that it can take several minutes for the environment to update while Elastic Beanstalk provisions AWS resources.

Elastic Beanstalk Environment Configuration

When deploying your applications, you may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages from the yum repository—or they could be configuration files such as a replacement for httpd.conf to override specific settings that are defaulted by AWS Elastic Beanstalk. You may also want to customize your environment resources that are part of your AWS Elastic Beanstalk environment (e.g., SQS queues, ElastiCache clusters). For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster.

You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle. When customizing the software on your instance, it is more advantageous to use a configuration file than to create a custom AMI because you do not need to maintain a set of AMIs.

Using Configuration Files

You can include one or more configuration files with your source bundle. Configuration files must be named with the extension **.config** (for example, `myapp.config`) and placed in an `.ebextensions` top-level directory in your source bundle. Configuration files are executed in alphabetical order. For example, `.ebextensions/01run.config` is executed before `.ebextensions/02do.config`.

Configuration files should conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively.

For Visual Studio, `.ebextensions` needs to be part of the project to be included in the archive. Alternatively, instead of including the configuration files in the project, you can use Visual Studio to deploy all files in the project folder.

To deploy all files in the project folder using Visual Studio

1. In Visual Studio, in **Solution Explorer**, right-click the project name, and then click **Properties**.
2. Click the **Package/Publish Web** tab.
3. In the **Items to deploy** section, select **All Files in the Project Folder** in the drop-down list.

When you are ready, deploy your application version. For more information, see Create an Application Version (p. 39).

> **Note**
> You can view the output of the steps executed during deployment by reviewing logs. For instructions on how to view logs, see Instance Logs (p. 282). If any error occurs during the deployment process, Elastic Beanstalk does not deploy the new application version. Elastic Beanstalk will continue to run the last application version that you successfully deployed.

When customizing your Elastic Beanstalk environment, you can configure the software on your Amazon EC2 instances as well as the AWS resources in your environment. This section is split into the following parts:

- Customizing Software on Linux Servers (p. 147)
- Customizing Software on Windows Servers (p. 160)
- Customizing Environment Resources (p. 127)

Each section describes the supported configuration settings and their syntax, as well as provides examples. For a full list of available options, namespaces, and supported values, see Configuration Options (p. 103).

This feature is not supported by **legacy** containers. See To check if you are using a legacy container type (p. 92) if you are unsure if you are on a legacy configuration or not.

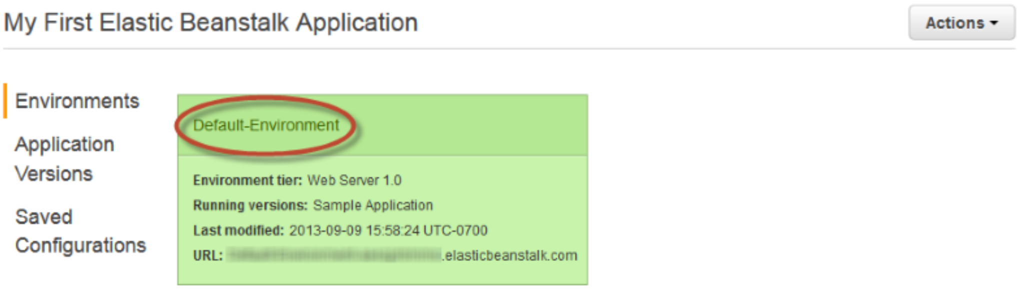
Changing Environment Configuration Settings

Elastic Beanstalk configures a number of AWS cloud computing services when deploying your application. You can edit your environment's configuration settings, which include the settings for these individual services.

AWS Management Console

To edit an application's environment settings

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work with.
3. On the Elastic Beanstalk console applications page, click the name of the environment whose settings you want to edit.



My First Elastic Beanstalk Application

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. In the **Overview** section of the environment dashboard, click **Configuration**.
5. For any of the configuration settings, click ⚙ in order to edit its configuration.

Command Line Interface (CLI)

To edit an application's environment settings

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:autoscaling:trigger",
        "OptionName": "LowerThreshold",
        "Value": "1000000"
    }
]
```

API

To edit an application's environment settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:trigger`
  - *`OptionSettings.member.1.OptionName`* = `LowerThreshold`
  - *`OptionSettings.member.1.Value`* = `1000000`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Atrigger
&OptionSettings.member.1.OptionName=LowerThreshold
&OptionSettings.member.1.Value=1000000
&Operation=UpdateEnvironment
&AuthParams
```

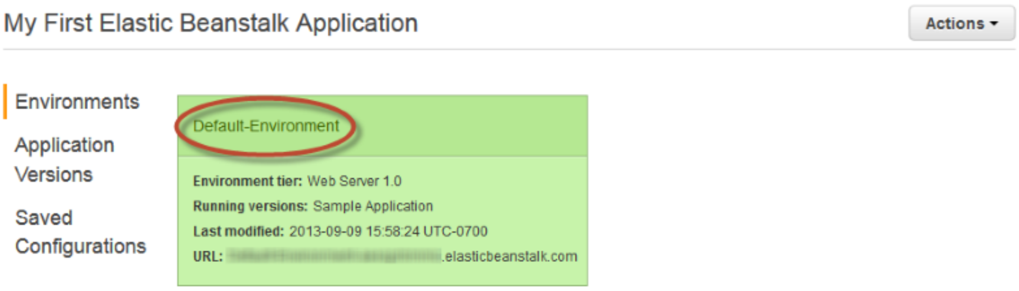
Saving Environment Configuration Settings

Elastic Beanstalk configures a number of AWS cloud computing services when it deploys your application. You can save your preferred environment's configuration settings, which include the settings for these individual services. You can easily apply your saved environment's configuration settings to other environments.

AWS Management Console

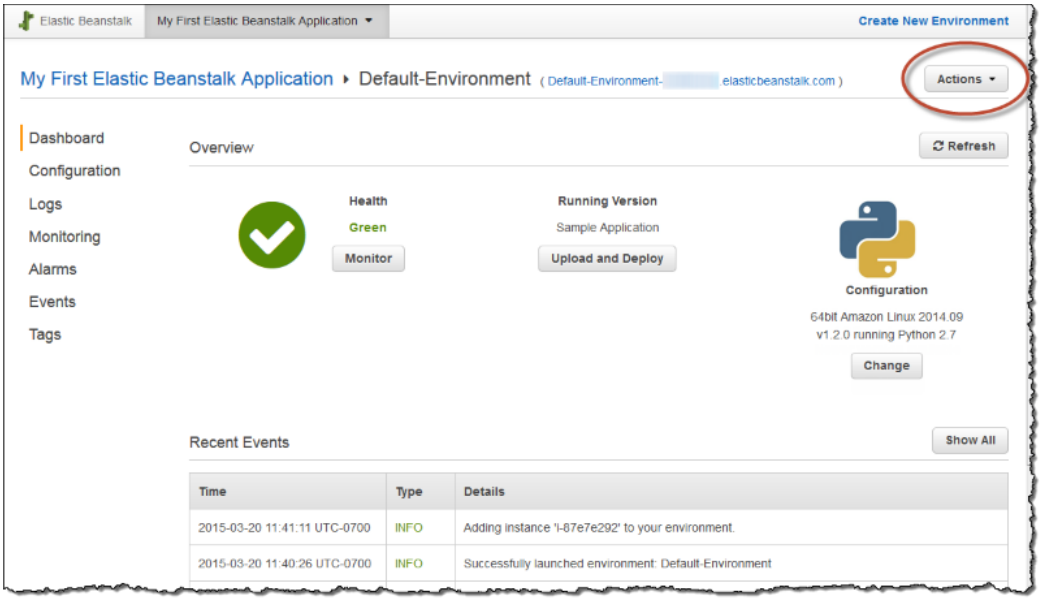
To save an application's environment settings

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work with.
3. On the Elastic Beanstalk console applications page, click the name of the environment whose settings you want to save.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

4. On the environment dashboard, click **Actions** and then select **Save Configuration**.



Time	Type	Details
2015-03-20 11:41:11 UTC-0700	INFO	Adding instance 'i-87e7e292' to your environment.
2015-03-20 11:40:26 UTC-0700	INFO	Successfully launched environment: Default-Environment

5. For **Configuration Name**, type the name of the configuration.
6. (Optional) For **Description**, type a description for this configuration.
7. Click **Save**.

Command Line Interface (CLI)

To save an application's environment settings

- Create a configuration template from an existing application.

```
$ aws elasticbeanstalk create-configuration-template --application-name my-
app --template-name my-app-v1 --environment-id e-ewrtp2pqsj
```

API

To save an application's environment settings

- Call `CreateConfigurationTemplate` with the following parameters:
  - *`ApplicationName`* = `SampleApp`
  - *`TemplateName`* = `MyConfigTemplate`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&Operation=CreateConfigurationTemplate
&AuthParams
```

Configuration Options

This section lists the namespaces, options and values that can be specified in an EB CLI configuration or .ebextensions configuration file. For instructions on using these files, see eb config (p. 405) and Using Configuration Files (p. 99).

Categories

- General Options for all Environments (p. 103)
- Docker Container Options (p. 121)
- Java with Tomcat Container Options (p. 121)
- .NET Container Options (p. 122)
- Node.js Container Options (p. 123)
- PHP Container Options (p. 125)
- Python Container Options (p. 125)
- Ruby Container Options (p. 126)

General Options for all Environments

Namespaces

- aws:autoscaling:asg (p. 104)
- aws:autoscaling:launchconfiguration (p. 105)
- aws:autoscaling:scheduledaction (p. 108)
- aws:autoscaling:trigger (p. 109)
- aws:autoscaling:updatepolicy:rollingupdate (p. 111)
- aws:ec2:vpc (p. 114)

aws:autoscaling:asg

Namespace: `aws:autoscaling:asg`			
Name	**Description**	**Default**	**Valid Values**
Availability Zones	Availability Zones are distinct locations within a region that are engineered to be isolated from failures in other Availability Zones and provide inexpensive, low-latency network connectivity to other Availability Zones in the same region. Choose the number of Availability Zones for your instances.	Any `1`	Any `1` Any `2`
Cooldown	Cooldown periods help to prevent Auto Scaling from initiating additional scaling activities before the effects of previous activities are visible.	`360`	`0` to `10000`

Custom Availability Zones	Define the Availability Zones for your instances.	n/a	`us-east-1a`
			`us-east-1b`
			`us-east-1c`
			`us-east-1d`
			`us-east-1e`
			`eu-central-1`

MinSize	Minimum number of instances you want in your Auto Scaling group.	`1`	`1` to `10000`
MaxSize	Maximum number of instances you want in your Auto Scaling group.	`4`	`1` to `10000`

aws:autoscaling:launchconfiguration

Namespace: `aws:autoscaling:launchconfiguration`			
Name	**Description**	**Default**	**Valid Values**
EC2KeyName	A key pair enables you to securely log into your Amazon EC2 instance.	n/a	n/a
IamInstanceProfile	An instance profile enables IAM users and AWS services to access temporary security credentials to make AWS API calls. Specify the profile name or the ARN. Example: `ElasticBeanstalkProfile` Example: `arn:aws:iam::123456789012:instance-profile/ElasticBeanstalkProfile`	n/a	n/a
ImageId	You can override the default Amazon Machine Image (AMI) by specifying your own custom AMI ID. Example: `ami-cbab67a2`	n/a	n/a
InstanceType	The instance type used to run your application in an Elastic Beanstalk environment. The instance types available depend on platform, solution stack (configuration) and region. To get a list of available instance types for your solution stack of choice, use the DescribeConfigurationOptions action in the API, describe-configuration-options command in the AWS CLI (p. 523). For example, the following command lists the available instance types for version 1.4.3 of the PHP 5.6 stack in the current region: `$ aws elasticbeanstalk describe-configuration-options --options Namespace=aws:autoscaling:launchconfiguration,OptionName=InstanceType --solution-stack-name "64bit Amazon Linux 2015.03 v1.4.3 running PHP 5.6"`	`t1.micro`	varies
MonitoringInterval	Interval at which you want Amazon CloudWatch metrics returned.	5 minutes	1 minute 5 minutes

Namespace: `aws:autoscaling:launchconfiguration`			
SecurityGroups	Lists the Amazon EC2 security groups to assign to the Amazon EC2 instances in the Auto Scaling group in order to define firewall rules for the instances. You can provide a single string of comma-separated values that contain the name of existing Amazon EC2 security groups or references to AWS::EC2::SecurityGroup resources created in the template. If you use Amazon VPC with Elastic Beanstalk so that your instances are launched within a virtual private cloud (VPC), specify security group IDs instead of a security group name.	`elasticbeanstalk-default`	n/a
SSHSourceRestriction	Used to lock down SSH access to an environment. For instance, you can lock down SSH access to the EC2 instances so that only a bastion host can access the instances in the private subnet. `protocol`–the allowed values for a security group ingress rule `fromPort`–the starting port for the firewall rule `toPort`–the end port for the firewall rule `sourceRestriction`– CIDR range or name of a security group to allow traffic from. To specify a security group from another account (EC2 classic only, must be in the same region), include the account ID prior to the security group name (e.g., otheraccountid/their-security-group). Example: `tcp, 22, 22, 54.240.196.185/32` Example: `tcp, 22, 22, my-security-group` Example (EC2-classic): `tcp, 22, 22, 0123456789012/their-security-group`	n/a	n/a

Namespace: `aws:autoscaling:launchconfiguration`			
BlockDeviceMappings	Used to attach additional Amazon Elastic Block Store volumes or instance store volumes on all the instances in the autoscaling group. When you map Amazon EBS volumes, you can specify either a volume size or a snapshot ID. When you map instance store volumes, you specify the virtual device name. Example: `/dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0` The format of the mapping is `device name=volume` where the device mappings are specified as a single string with mappings separated by a comma. This example attaches to all instances in the autoscaling group an empty 100-GB Amazon EBS volume, an Amazon EBS volume with the snapshot ID `snap-51eef269`, and an instance store volume.	n/a	n/a
RootVolumeType	Type of storage volume to attach to Amazon EC2 instances in your environment.	n/a	`standard`, which is a Magnetic storage volume `gp2`, which is a General Purpose (SSD) storage volume `io1`, which is a Provisioned IOPS (SSD) storage volume
RootVolumeSize	Size of the storage volume that you specified as the `RootVolumeType`. You must specify your desired root volume size if you choose Provisioned IOPS (SSD) as the root volume type. There is no default root volume size for Provisioned IOPS (SSD) volumes. For Magnetic and General Purpose (SSD) volumes, if you do not specify your own value, Elastic Beanstalk will use the default volume size for the storage volume type. The default volume size varies according to the AMI of the solution stack on which your environment is based.	n/a	`10` to `1024` gibibytes for Provisioned IOPS (SSD) root volumes `8` to `1024` gibibytes for Magnetic volumes and General Purpose (SSD) volumes

Namespace: `aws:autoscaling:launchconfiguration`			
RootVolumeIOPS	Desired input/output operations per second (IOPS) for a Provisioned IOPS (SSD) root volume. The maximum ratio of IOPS to your volume size is 30 to 1. For example, a volume with 3000 IOPS must be at least 100 GiB.	n/a	`100` to `4000`

aws:autoscaling:scheduledaction

Namespace: `aws:autoscaling:scheduledaction`			
Name	**Description**	**Default**	**Valid Values**
DesiredCapacity	The number of instances that you want running in the autoscaling group.	n/a	null `1` to `10000`
EndTime	A date and time in the future (in the UTC/GMT time zone) when you want the scheduled scaling action to stop. If you don't specify an **EndTime**, the recurrence is ongoing until you configure it with an end date and time. Example: `2015-04-28T04:07:2Z` For more information on the ISO-8601 timestamp format, visit http://www.w3.org/TR/NOTE-datetime.	n/a	null A unique date and time across all scheduled scaling actions.
MaxSize	The maximum number of instances that you want in the autoscaling group for the duration of the scheduled action.	n/a	`0` to `10000`
MinSize	The minimum number of instances that you want in the autoscaling group for the duration of the scheduled action.	n/a	`0` to `10000`
Recurrence	The frequency with which you want the scheduled action to occur. If you do not specify a recurrence, then the scaling action will occur only once, as specified by the StartTime.	n/a	null A CRON expression. For more information about CRON, see Cron.

Namespace: `aws:autoscaling:scheduledaction`			
StartTime	A date and time in the future (in the UTC/GMT time zone) when you want the scheduled scaling action to start.	n/a	null A unique date and time across all scheduled actions in ISO 8601 time format. For example, 2015-04-28T04:07:02Z. For more information about ISO 8601 time format, go to Date and Time Formats.

Suspend	Whether to temporarily stop the scheduled action.	`false`	`True`
			`False`
			null

Note

When you configure a scheduled scaling action, include a resource name with a unique value as the name for the scheduled scaling action. All option settings that share this resource name apply to the same scheduled action. For example:

```
option_settings:
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledAction01
    option_name: MinSize
    value: 2
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledAction01
    option_name: MaxSize
    value: 5
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledAction01
    option_name: StartTime
    value: "2015-05-14T19:25:00Z"
```

aws:autoscaling:trigger

Namespace: `aws:autoscaling:trigger`			
Name	**Description**	**Default**	**Valid Values**
BreachDuration	Amount of time a metric can be beyond its defined limit (as specified in the `UpperThreshold` and `LowerThreshold`) before the trigger fires.	5	`1` to `600`
LowerBreachScaleIncrement	How many Amazon EC2 instances to remove when performing a scaling activity.	`-1`	n/a

Namespace: `aws:autoscaling:trigger`			
LowerThreshold	If the measurement falls below this number for the breach duration, a trigger is fired.	`2000000`	`0` to `20000000`

MeasureName	Metric used for your auto scaling trigger.	`NetworkOut`	`CPUUtilization`
			`NetworkIn`
			`NetworkOut`
			`DiskWriteOps`
			`DiskReadBytes`
			`DiskReadOps`
			`DiskWriteBytes`
			`Latency`
			`RequestCount`
			`HealthyHostCount`
			`UnhealthyHostCount`

Period	Specifies how frequently Amazon CloudWatch measures the metrics for your trigger.	`5`	n/a

Statistic	Statistic the trigger should use, such as `Average`.	`Average`	`Minimum`
			`Maximum`
			`Sum`
			`Average`

Unit	Unit for the trigger measurement, such as `Bytes`.	`Bytes`	`Seconds`
			`Percent`
			`Bytes`
			`Bits`
			`Count`
			`Bytes/Second`
			`Bits/Second`
			`Count/Second`
			`None`

UpperBreachScaleIncrement	How many Amazon EC2 instances to add when performing a scaling activity.	`1`	n/a

Namespace: `aws:autoscaling:trigger`			
UpperThreshold	If the measurement is higher than this number for the breach duration, a trigger is fired.	`6000000`	`0` to `20000000`

aws:autoscaling:updatepolicy:rollingupdate

Namespace: `aws:autoscaling:updatepolicy:rollingupdate`			
Name	**Description**	**Default**	**Valid Values**
MaxBatchSize	The number of instances included in each batch of the rolling update.	One-third of the minimum size of the autoscaling group, rounded to the next highest integer	`1` to `10000`
MinInstancesInService	The minimum number of instances that must be in service within the autoscaling group while other instances are terminated.	Equal to either the minimum size of the autoscaling group or one less than the maximum size of the autoscaling group, whichever number is lower	`0` to `9999`
PauseTime	The amount of time the Elastic Beanstalk service will wait after it has completed updates to one batch of instances before it continues on to the next batch.	Automatically computed based on instance type and container	`PT0S` (0 seconds) to `PT1H` (1 hour) The value must be in ISO8601 duration format, in the form: `PT#H#M#S` where each # is the number of hours, minutes, and/or seconds, respectively.

Namespace: `aws:autoscaling:updatepolicy:rollingupdate`			
RollingUpdateEnabled	If true, enables rolling updates for an environment. Rolling updates are useful when you need to make small, frequent updates to your Elastic Beanstalk software application and you want to avoid application downtime. **Note** Setting this value to true automatically enables the `MaxBatchSize`, `MinInstancesInService`, and `PauseTime` options. Setting any of those options also automatically sets the `RollingUpdateEnabled` option value to `true`. Setting this option to `false` disables rolling updates.	`false`	`true` `false`

Namespace: `aws:autoscaling:updatepolicy:rollingupdate`			
RollingUpdateType	Whether to apply rolling updates (of environment configuration changes) to a batch of instances according to one of the following: • Wait the specified amount of time (PauseTime) after completing updates to one batch of instances before applying updates to the next batch. • Begin applying rolling updates to a new batch of instances after receiving reports that the current batch of instances being updated is healthy. **Note** Only load-balanced web server environments can use health-based rolling updates. Single-instance web server environments and worker environments can apply rolling updates only according to time-based criteria.	`Time`	`Time` `Health`
Timeout	Maximum amount of time to wait for all instances in a batch of instances to report healthy status before canceling the update and rolling back to the previous environment configuration settings.	`PT30M` (30 minutes)	`PT15M` (15 minutes) to `PT1H` (1 hour) The value must be in ISO8601 duration format, in the form: `PT#H#M#S` where each # is the number of hours, minutes, and/or seconds, respectively.

aws:ec2:vpc

Namespace: `aws:ec2:vpc`			
Name	**Description**	**Default**	**Valid Values**
VPCId	The ID for your VPC.	n/a	n/a
Subnets	The ID of the Auto Scaling group subnet or subnets. If you have multiple subnets, specify the value of the option name as a single comma-delimited string of subnet IDs (for example, `subnet-11111111,subnet-22222222`).	n/a	n/a
ELBSubnets	The ID of the subnet for the elastic load balancer.	n/a	n/a
ELBScheme	Specify `internal` if you want to create an internal load balancer in your VPC so that your Elastic Beanstalk application cannot be accessed from outside your VPC.	n/a	`internal`
DBSubnets	Contains the ID of the DB subnets. This is only used if you want to add an Amazon RDS DB Instance as part of your application.	n/a	n/a

AssociatePublicIpAddress	Specifies whether to launch instances with public IP addresses in your VPC. Instances with public IP addresses do not require a NAT instance to communicate with the Internet. You must set the value to `true` if you want to include your load balancer and instances in a single public subnet.	n/a	`true`
			`false`
			`null`

aws:elasticbeanstalk:application

Namespace: `aws:elasticbeanstalk:application`			
Name	**Description**	**Default**	**Valid Values**
Application Healthcheck URL	The URL the Elastic Load Balancer uses to query for instance health.	`/`	A blank string is treated as `/`, or specify a string starting with `/`.

aws:elasticbeanstalk:application:environment

Namespace: `aws:elasticbeanstalk:application:environment`			
Name	**Description**	**Default**	**Valid Values**
Any environment variable name.	Pass in key-value pairs.	n/a	Any environment variable value.

Note
The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **`KEY1=VALUE1, KEY2=VALUE2`**, which means that both the value and key of each variable are included in the total. When a platform has one

or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

aws:elasticbeanstalk:command

Namespace: `aws:elasticbeanstalk:command`			
Name	**Description**	**Default**	**Valid Values**
Timeout	Number of seconds to wait for an instance to complete executing commands. For example, if source code deployment tasks are still running when you reach the configured timeout period, Elastic Beanstalk displays the following error: "Some instances have not responded to commands. Responses were not received from *`<instance id>`*." You can increase the amount of time that the Elastic Beanstalk service waits for your source code to successfully deploy to the instance.	`"600"`	`"1"` to `"3600"`
BatchSize	Percentage or fixed number of Amazon EC2 instances in the Auto Scaling group on which to simultaneously deploy an application version.	`100`	For `Percentage`, `1` to `100`. For `Fixed`, a number that is less than or equal to the maximum number of instances to run at any given time in the autoscaling group. (The maximum number of instances that you can run in an autoscaling group is 10000.)
BatchSizeType	The type of number that is specified in **BatchSizeType**.	`Percentage`	`Fixed` `Percentage`
IgnoreHealthCheck	Do not cancel a deployment due to failed health checks.	`false`	`true` `false`

aws:elasticbeanstalk:environment

Namespace: `aws:elasticbeanstalk:environment`			
Name	**Description**	**Default**	**Valid Values**
EnvironmentType	The type of environment, either a load-balanced and auto-scaled environment or a single-instance environment.	`LoadBalanced`	`SingleInstance` `LoadBalanced`
ServiceRole (p. 19)	The name of an IAM role that AEB uses manage resources for the environment.	none	Any role name.

aws:elasticbeanstalk:healthreporting:system

Namespace: `aws:elasticbeanstalk:healthreporting:system`			
Name	**Description**	**Default**	**Valid Values**
SystemType (p. 244)	Health reporting system (basic (p. 247) or enhanced (p. 250)). Enhanced health reporting requires a service role (p. 19) and a version 2 platform configuration (p. 24).	`basic`	`basic` `enhanced`
ConfigDocument (p. 265)	A JSON document describing the environment and instance metrics to publish to CloudWatch.	none	

aws:elasticbeanstalk:hostmanager

Namespace: `aws:elasticbeanstalk:hostmanager`			
Name	**Description**	**Default**	**Valid Values**
LogPublicationControl	Copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application.	`false`	`true` `false`

aws:elasticbeanstalk:monitoring

Namespace: `aws:elasticbeanstalk:monitoring`			
Name	**Description**	**Default**	**Valid Values**
Automatically Terminate Unhealthy Instances	Specify if you want to terminate unhealthy instances automatically.	`true`	`true` `false`

aws:elasticbeanstalk:sns:topics

Namespace: `aws:elasticbeanstalk:sns:topics`			
Name	**Description**	**Default**	**Valid Values**
Notification Endpoint	Endpoint where you want to be notified of important events affecting your application.	n/a	n/a

Notification Protocol	Protocol used to send notifications to your endpoint.	`email`	`http`
			`https`
			`email`
			`email-json`
			`sqs`

Notification Topic ARN	Amazon Resource Name for the topic you subscribed to.	n/a	n/a
Notification Topic Name	Name of the topic you subscribed to.	n/a	n/a

aws:elasticbeanstalk:sqsd

Namespace: `aws:elasticbeanstalk:sqsd`			
Name	**Description**	**Default**	**Valid Values**
WorkerQueueURL	The URL of the queue from which the daemon in the worker environment tier reads messages	automatically generated	**Note** If you don't specify a value, then Elastic Beanstalk automatically creates a queue.
HttpPath	The relative path to the application to which HTTP POST messages are sent	/	

MimeType	The MIME type of the message sent in the HTTP POST request	`application/json`	`application/json`
			`application/x-www-form-urlencoded`
			`application/xml`
			`text/plain`
			Note
			You can create your own MIME type.

HttpConnections	The maximum number of concurrent connections to any application(s) within an Amazon EC2 instance	`15`	`1` to `100`

Namespace: `aws:elasticbeanstalk:sqsd`			
ConnectTimeout	The amount of time, in seconds, to wait for successful connections to an application	`5`	`1` to `60`
InactivityTimeout	The amount of time, in seconds, to wait for a response on an existing connection to an application **Note** The message is reprocessed until the daemon receives a 200 OK response from the application in the worker environment tier or the `RetentionPeriod` expires.	`180`	`1` to `1800`
VisibilityTimeout	The amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, then the message is again made visible in the queue for any other daemon to read. **Note** If you configure this value, it overrides the SQS VisibilityTimeout setting.	300	`0` to `43200`
RetentionPeriod	The amount of time, in seconds, a message is valid and will be actively processed	`345600`	`60` to `1209600`
MaxRetries	The maximum number of attempts that Elastic Beanstalk attempts to send the message to the web application that will process it before moving the message to the dead letter queue.	`10`	`1` to `1000`
ErrorVisibilityTimeout	The amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after a processing attempt fails with an explicit error.	2 seconds	`0` to `43200` seconds

aws:elb:healthcheck

Namespace: `aws:elb:healthcheck`			
Name	**Description**	**Default**	**Valid Values**
HealthyThreshold (p. 190)	Consecutive successful requests before Elastic Load Balancing changes the instance health status.	3	`2` to `10`

Namespace: `aws:elb:healthcheck`			
Interval (p. 190)	The interval at which Elastic Load Balancing will check the health of your application's Amazon EC2 instances.	`10`	`5` to `300`
Timeout (p. 190)	Number of seconds Elastic Load Balancing will wait for a response before it considers the instance nonresponsive.	`5`	`2` to `60`
UnhealthyThreshold (p. 190)	Consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	`5`	`2` to `10`

aws:elb:loadbalancer

Namespace: `aws:elb:loadbalancer`			
Name	**Description**	**Default**	**Valid Values**
CrossZone	Specifies whether the load balancer routes traffic evenly across all instances in all Availability Zones rather than only within each zone.	`false`	`true` `false`

LoadBalancerHTTPPort	External facing port used by the listener.	`80`	`OFF`
			`80`

LoadBalancerPortProtocol	Protocol used by the listener.	`HTTP`	`HTTP` `TCP`

LoadBalancerHTTPSPort	External facing port used by the secure listener.	`OFF`	`OFF`
			`443`
			`8443`

LoadBalancerSSLPortProtocol	Protocol used by the secure listener.	`HTTPS`	`HTTPS` `SSL`
SSLCertificateId	Amazon Resource Name (ARN) for the SSL certificate you've uploaded for AWS Access and Identity Management.	n/a	n/a

aws:elb:policies

Namespace: `aws:elb:policies`			
Name	**Description**	**Default**	**Valid Values**
ConnectionDrainingEnabled	Specifies whether the load balancer maintains existing connections to instances that have become unhealthy or deregistered to complete in-progress requests.	`false`	`true` `false`

Namespace: `aws:elb:policies`			
ConnectionDrainingTimeout	Maximum number of seconds that the load balancer maintains existing connections to an instance during connection draining before forcibly closing the connections.	`20`	`1` to `3600`
ConnectionSettingIdleTimeout	Number of seconds that the load balancer waits for any data to be sent or received over the connection. If no data has been sent or received after this time period elapses, the load balancer closes the connection.	`60`	`1` to `3600`
Stickiness Cookie Expiration	Duration of validity for each cookie.	`0`	`0` to `1000000`
Stickiness Policy	Binds a user's session to a specific server instance so that all requests coming from the user during the session will be sent to the same server instance.	`false`	`true` `false`

aws:rds:dbinstance

Namespace: `aws:rds:dbinstance`			
Name	**Description**	**Default**	**Valid Values**
DBAllocatedStorage	The allocated database storage size, specified in gigabytes.	MySQL: `5` Oracle: `10` sqlserver-se: `200` sqlserver-ex: `30` sqlserver-web: `30`	MySQL: `5-1024` Oracle: `10-1024` sqlserver: cannot be modified
DBDeletionPolicy	Decides whether to delete or snapshot the DB instance on environment termination. **Warning** Deleting a DB instance results in permanent data loss.	`Delete`	`Delete` `Snapshot`

Namespace: `aws:rds:dbinstance`			

DBEngine	The name of the database engine to use for this instance.	`mysql`	`mysql`
			`oracle-se1`
			`oracle-se`
			`oracle-ee`
			`sqlserver-ee`
			`sqlserver-ex`
			`sqlserver-web`
			`sqlserver-se`
			`postgres`

DBEngineVersion	The version number of the database engine.	`5.5`	n/a
DBInstanceClass	The database instance type.	`db.t1.micro`	Go to DB Instance Class in the *Amazon Relational Database Service User Guide*.
DBPassword	The name of master user password for the database instance.	n/a	n/a
DBSnapshotIdentifier	The identifier for the DB snapshot to restore from.	n/a	n/a
DBUser	The name of master user for the DB Instance.	`ebroot`	n/a
MultiAZDatabase	Specifies whether a database instance Multi-AZ deployment needs to be created. For more information about Multi-AZ deployments with Amazon Relational Database Service (RDS), go to Regions and Availability Zones in the *Amazon Relational Database Service User Guide*.	`false`	`true` `false`

Docker Container Options

No Docker-specific configuration options are provided by Elastic Beanstalk

Java with Tomcat Container Options

Namespaces

- aws:elasticbeanstalk:application:environment (p. 122)
- aws:elasticbeanstalk:container:tomcat:jvmoptions (p. 122)

aws:elasticbeanstalk:application:environment

Namespace: `aws:elasticbeanstalk:application:environment`			
Name	**Description**	**Default**	**Valid Values**
JDBC_CONNECTION_STRING	Connection string to an external database.	n/a	n/a

> **Note**
> The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **`KEY1=VALUE1, KEY2=VALUE2`**, which means that both the value and key of each variable are included in the total. When a platform has one or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

aws:elasticbeanstalk:container:tomcat:jvmoptions

Namespace: `aws:elasticbeanstalk:container:tomcat:jvmoptions`			
Name	**Description**	**Default**	**Valid Values**
JVM Options	Pass command-line options to the JVM at startup.	n/a	n/a
Xmx	Maximum JVM heap sizes.	`256m`	n/a
XX:MaxPermSize	Section of the JVM heap that is used to store class definitions and associated metadata.	`64m`	n/a
Xms	Initial JVM heap sizes.	`256m`	n/a

> **Note**
> You can extend the number of parameters and specify the parameter names in the `aws:elasticbeanstalk:container:tomcat:jvmoptions` and `aws:elasticbeanstalk:application:environment` namespaces.

.NET Container Options

Namespaces

- aws:elasticbeanstalk:application:environment (p. 122)
- aws:elasticbeanstalk:container:dotnet:apppool (p. 123)

aws:elasticbeanstalk:application:environment

Namespace: `aws:elasticbeanstalk:application:environment`			
Name	**Description**	**Default**	**Valid Values**
PARAM1 – PARAM5	Pass in key-value pairs.	n/a	n/a

Note

The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **`KEY1=VALUE1, KEY2=VALUE2`**, which means that both the value and key of each variable are included in the total. When a platform has one or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

aws:elasticbeanstalk:container:dotnet:apppool

Namespace: **`aws:elasticbeanstalk:container:dotnet:apppool`**			
Name	**Description**	**Default**	**Valid Values**

Target Runtime	You can choose the version of .NET Framework for your application.	`4.0`	`2.0`
			`4.0`

Enable 32-bit Applications	Enable 32-bit applications.	`False`	`True` `False`

Node.js Container Options

Namespaces

- aws:elasticbeanstalk:container:nodejs (p. 123)
- aws:elasticbeanstalk:container:nodejs:staticfiles (p. 124)

aws:elasticbeanstalk:container:nodejs

Namespace: **`aws:elasticbeanstalk:container:nodejs`**			
Name	**Description**	**Default**	**Valid Values**
NodeCommand	Command used to start the Node.js application. If an empty string is specified, `app.js` is used, then `server.js`, then "npm start" in that order.	""	n/a

Namespace: `aws:elasticbeanstalk:container:nodejs`			

NodeVersion	Version of Node.js.
	Supported Node.js versions vary between versions of the Node.js platform configuration. The versions shown to the right are from version 2.0.0. See Node.js (p. 27) on the supported platforms page for a list of the currently supported versions.
	Note
	When support for the version of Node.js that you are using is removed from the platform configuration, you must change the version setting prior to doing a platform upgrade (p. 90). This may occur when a security vulnerability is identified for one or more versions of Node.js
	When this occurs, attempting to upgrade to a new version of the platform that does not support the configured NodeVersion (p. 123) will fail. To avoid needing to create a new environment, change the *NodeVersion* configuration option to a version that is supported by both the old configuration version and the new one, and then perform the platform upgrade.
		`0.12.6`	`0.8.28`
			`0.10.31`
			`0.10.38`
			`0.10.39`

GzipCompression	Specifies if gzip compression is enabled. If ProxyServer is set to `"none"`, then gzip compression will be disabled.	`false`	`true` `false`

ProxyServer	Specifies which web server should be used to proxy connections to Node.js. If ProxyServer is set to `"none"`, then static file mappings will not take affect and gzip compression will be disabled.	`nginx`	`apache`
			`nginx`
			`none`

aws:elasticbeanstalk:container:nodejs:staticfiles

Namespace: `aws:elasticbeanstalk:container:nodejs:staticfiles`			
Name	**Description**	**Default**	**Valid Values**
/public (this can be any arbitrary name)	The directory that contains static content. If ProxyServer is set to `"none"`, then the static file mappings will not take affect. Example: `/public`	n/a	n/a

PHP Container Options

aws:elasticbeanstalk:container:php:phpini

Namespace: `aws:elasticbeanstalk:container:php:phpini`			
Name	**Description**	**Default**	**Valid Values**
document_root	Specify the child directory of your project that is treated as the public-facing web root.	`/`	A blank string is treated as `/`, or specify a string starting with `/`
memory_limit	Amount of memory allocated to the PHP environment.	`128M`	n/a
zlib.output_compression	Specifies whether or not PHP should use compression for output.	`false`	`true` `false`
allow_url_fopen	Specifies if PHP's file functions are allowed to retrieve data from remote locations, such as websites or FTP servers.	`true`	`true` `false`
display_errors	Specifies if error messages should be part of the output.	`Off`	`On` `Off`
max_execution_time	Sets the maximum time, in seconds, a script is allowed to run before it is terminated by the environment.	`60`	`0` to `9223372036854775807` (PHP_INT_MAX)
composer_options	Sets custom options to use when installing dependencies using Composer through composer.phar install. For more information including available options, go to http://getcomposer.org/doc/03-cli.md#install.	n/a	n/a

Python Container Options

Namespaces

- aws:elasticbeanstalk:application:environment (p. 125)
- aws:elasticbeanstalk:container:python (p. 126)
- aws:elasticbeanstalk:container:python:staticfiles (p. 126)

aws:elasticbeanstalk:application:environment

Namespace: `aws:elasticbeanstalk:application:environment`			
Name	**Description**	**Default**	**Valid Values**
DJANGO_SETTINGS_MODULE	Specifies which settings file to use.	n/a	n/a

Note

The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **`KEY1=VALUE1, KEY2=VALUE2`**, which means that both the value and key of each variable are included in the total. When a platform has one or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

aws:elasticbeanstalk:container:python

Namespace: `aws:elasticbeanstalk:container:python`			
Name	**Description**	**Default**	**Valid Values**
WSGIPath	The file that contains the WSGI application. This file must have an "application" callable.	`application.py`	n/a
NumProcesses	The number of daemon processes that should be started for the process group when running WSGI applications.	`1`	n/a
NumThreads	The number of threads to be created to handle requests in each daemon process within the process group when running WSGI applications.	`15`	n/a

aws:elasticbeanstalk:container:python:staticfiles

Namespace: `aws:elasticbeanstalk:container:python:staticfiles`			
Name	**Description**	**Default**	**Valid Values**
/static/ (this can be any arbitrary name)	A mapping of the URL to a local directory. Example: `static/` This would map files in your directory on your EC2 instance (`/opt/python/current/app/static/*`) to *<your domain>*`/static/*`	n/a	n/a

Ruby Container Options

aws:elasticbeanstalk:application:environment

Namespace: `aws:elasticbeanstalk:application:environment`			
Name	**Description**	**Default**	**Valid Values**

Namespace: **aws:elasticbeanstalk:application:environment**			
RAILS_SKIP_MIGRATIONS	Specifies whether to run `rake db:migrate` on behalf of the users' applications; or whether it should be skipped. This is only applicable to Rails 3.x applications.	false	true false
RAILS_SKIP_ASSET_COMPILATION	Specifies whether the container should run `rake assets:precompile` on behalf of the users' applications; or whether it should be skipped. This is also only applicable to Rails 3.x applications.	false	true false
BUNDLE_WITHOUT	A colon (:) separated list of groups to ignore when installing dependencies from a Gemfile.	test:development	n/a
RACK_ENV	Specifies what environment stage an application can be run in. Examples of common environments include development, production, test.	production	n/a
RAILS_ENV	Specifies what environment stage an application can be run in. Examples of common environments include development, production, test.	production	n/a

Note

The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **KEY1=VALUE1, KEY2=VALUE2**, which means that both the value and key of each variable are included in the total. When a platform has one or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

Customizing Environment Resources

You may also want to customize your environment resources that are part of your Elastic Beanstalk environment. For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster. You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle.

This section describes the type of information you can include in a configuration file to customize your AWS resources. For general information on customizing and configuring your Elastic Beanstalk environments, see Elastic Beanstalk Environment Configuration (p. 99).

Note

Elastic Beanstalk requires you to use IAM roles to launch an environment and to manage environments and applications. An instance profile is associated with an IAM role that can be configured to provide applications and services access to AWS resources using temporary security credentials. If you are unsure whether you have deployed your application using an instance profile, click **Instances** on the **Configuration** page in the Elastic Beanstalk console. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199). To learn more about instance profiles, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326) and Using IAM Roles with Elastic Beanstalk (p. 332).

Resources

You can use the `Resources` key to create and customize AWS resources in your environment. For a complete reference, see Customizing AWS Resources (p. 800).

Syntax

When defining the values for the properties for your resource, there are three different methods you can use to define the values for the properties for a resource:

- Pass the value
- Pass a list of values
- Pass the option name and the value

There are two different functions you can use to retrieve the values for the properties:

- Use `Fn::GetAtt` to return the value of an attribute from a resource. For more information, see Fn::Join.
- Use `Ref` to return the value of a specified parameter or resource. For more information, see Ref.

```
Resources:
  <name of resource>:
    Type: <resource type identifier>
    Properties:
      # Example syntax of a property that takes in the actual value
      <property name>: <literal string>

      # Example syntax of a property that takes a list of strings
      <property name>: ["<literal string>", "<literal string>"]

      # Example syntax of a property that takes the option name and the value
       <property name>:
         - Name: <option name>
           Value: <literal string>

      # Example syntax showing how to use Fn::GetAtt to return the value of an
 attribute from a resource in the configuration file
      <property name>:
         - Name: <option name>
           Value : { "Fn::GetAtt" : [ "<logicalNameOfResource>", "<attribute
Name>"] }

     # Example syntax showing how to use Ref to return the value of a specified
 parameter or resource. You can use Ref for single property values and lists.
      <property name>:
          Ref: <parameter reference>
```

Options

This table shows the available keys and descriptions for the **Resources** key.

Key	Description
`<name of resource>`	The name for what you want to create your resource. Each resource must have a logical name unique within the configuration file. This is the name you use elsewhere in the configuration file to reference the resource.

This table shows the available keys and descriptions for each resource name you provide.

Key	Description
`Type`	This is the resource type identifier. For a list of resource type identifiers, see AWS Resource Types Reference (p. 800).
`Properties`	Optional. A `Properties` section is declared for each resource immediately after the resource `Type` declaration. Property values can be literal strings, lists of strings, parameter references, pseudo references, or the value returned by a function. If a resource does not require any properties to be declared, you can omit the `Properties` section of that resource.

Elastic Beanstalk Resource Names

Elastic Beanstalk provides fixed resource names for the AWS resources that it creates for you when you deploy your application. You will need to know these resource names when you reference them in your configuration file.

Resource Name	Description
`AWSEBAutoScalingGroup`	The name of the Auto Scaling group that Elastic Beanstalk uses when it launches EC2 instances.
`AWSEBAutoScalingLaunchConfiguration`	The name for the launch configuration settings that Elastic Beanstalk uses when it launches EC2 instances.
`AWSEBEnvironmentName`	The name of the Elastic Beanstalk environment.
`AWSEBLoadBalancer`	The name of the elastic load balancer used in the Elastic Beanstalk environment.
`AWSEBRDSDatabase`	The name of the Amazon RDS database.
`AWSEBSecurityGroup`	The name for the EC2 security group that Elastic Beanstalk uses when it launches EC2 instances.
`AWSEBWorkerQueue`	The Amazon SQS queue from which the daemon in a worker environment tier pulls requests that need to be processed.
`AWSEBWorkerDeadLetterQueue`	The Amazon SQS queue that stores messages that cannot be delivered or otherwise were not successfully processed by the daemon in a worker environment tier.
`AWSEBWorkerCronLeaderRegistry`	The Amazon DynamoDB table that is the internal registry used by the daemon in a worker environment tier for periodic tasks.

Example Snippets: ElastiCache

The following samples add an Amazon ElastiCache cluster to EC2-Classic and EC2-VPC (both default VPC and nondefault VPC) platforms. For more information about these platforms and how you can determine which ones EC2 supports for your region and your AWS account, see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-supported-platforms.html. Then refer to the section in this topic that applies to your platform.

- EC2-Classic Platforms (p. 130)
- EC2-VPC (Default) (p. 132)
- EC2-VPC (Nondefault) (p. 134)

> **Note**
> Elastic Beanstalk requires you to use IAM roles to launch an environment and to manage environments and applications. An instance profile is associated with an IAM role that can be configured to provide applications and services access to AWS resources using temporary security credentials. If you are unsure whether you have deployed your application using an instance profile, click **Instances** on the **Configuration** page in the Elastic Beanstalk console. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199). To learn more about instance profiles, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326) and Using IAM Roles with Elastic Beanstalk (p. 332).

EC2-Classic Platforms

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-Classic platform. All of the properties that are listed in this example are the minimum required properties that must be set for each resource type. You can download the example at ElastiCache Example.

> **Note**
> This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

   Note

   > **Note**
   > Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Create a configuration file (e.g., `elasticache.config`) that defines the resources. In this example, we create the ElastiCache cluster by specifying the name of the ElastiCache cluster resource (`MyElastiCache`), declaring its type, and then configuring the properties for the cluster. The example references the name of the ElastiCache security group resource that gets created and defined in this configuration file. Next, we create an ElastiCache security group. We define the name for this resource, declare its type, and add a description for the security group. Finally, we set the ingress rules for the ElastiCache security group to allow access only from instances inside the ElastiCache security group (`MyCacheSecurityGroup`) and the Elastic Beanstalk security group (`AWSEBSecurityGroup`). The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.

```
#This sample requires you to create a separate configuration file that defines
 the custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
         Fn::GetOptionSetting:
             OptionName : CacheNodeType
             DefaultValue: cache.m1.small
      NumCacheNodes:
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
      Engine:
           Fn::GetOptionSetting:
             OptionName : Engine
             DefaultValue: memcached
      CacheSecurityGroupNames:
        - Ref: MyCacheSecurityGroup
  MyCacheSecurityGroup:
    Type: AWS::ElastiCache::SecurityGroup
    Properties:
      Description: "Lock cache down to webserver access only"
  MyCacheSecurityGroupIngress:
    Type: AWS::ElastiCache::SecurityGroupIngress
    Properties:
      CacheSecurityGroupName:
        Ref: MyCacheSecurityGroup
      EC2SecurityGroupName:
        Ref: AWSEBSecurityGroup
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::SecurityGroup
- AWS::ElastiCache:SecurityGroupIngress

Create a separate configuration file called `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     CacheNodeType : cache.m1.small
     NumCacheNodes : 1
     Engine : memcached
```

These lines tell Elastic Beanstalk to get the values for the **CacheNodeType, NumCacheNodes, and Engine** properties from the **CacheNodeType, NumCacheNodes, and Engine** values in a config file (options.config in our example) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means cache.m1.small, 1, and memcached would be used for the values. For more information about `Fn::GetOptionSetting`, see Fn::GetOptionSetting (p. 801).

EC2-VPC (Default)

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-VPC platform. Specifically, the information in this section applies to a scenario where EC2 launches instances into the default VPC. All of the properties in this example are the minimum required properties that must be set for each resource type. For more information about default VPCs, see Your Default VPC and Subnets.

> **Note**
> This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

> **Note**
>
> > **Note**
> > Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Now name the resources configuration file `elasticache.config`. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (`MyElastiCache`), declares its type, and then configures the properties for the cluster. The example references the ID of the security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (`AWSEBSecurityGroup`). (The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is `6379`.

```
#This sample requires you to create a separate configuration file that defines
 the custom option settings for CacheCluster properties.

Resources:
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupName:
            Ref: "AWSEBSecurityGroup"
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t1.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      VpcSecurityGroupIds:
        -
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId

Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElastiCache"
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup

Next, name the options configuration file `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t1.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
```

These lines tell Elastic Beanstalk to get the values for the CacheNodeType, NumCacheNodes, Engine, and CachePort properties from the CacheNodeType, NumCacheNodes, Engine, and CachePort values in a config file (options.config in our example). That file includesan aws:elasticbeanstalk:customoption section (under options_settings) that contains name-value pairs with the actual values to use. In the preceding example, cache.t1.micro, 1, redis, and 6379 would be used for the values. For more information about Fn::GetOptionSetting, see Fn::GetOptionSetting (p. 801).

EC2-VPC (Nondefault)

If you create a nondefault VPC on the EC2-VPC platform and specify it as the VPC into which EC2 launches instances, the process of adding an Amazon ElastiCache cluster to your environment differs from that of a default VPC. The main difference is that you must create a subnet group for the ElastiCache cluster. All of the properties in this example are the minimum required properties that must be set for each resource type.

Note

This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an .ebextensions directory in the top-level directory of your source bundle.
2. Create two configuration files with the .config extension and place them in your .ebextensions directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

Note

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Now name the resources configuration file elasticache.config. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (MyElastiCache), declares its type, and then configures the properties for the cluster. The properties in the example reference the name of the subnet group for the ElastiCache cluster as well as the ID of security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, the VPC ID, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (AWSEBSecurityGroup). (The parameter name, AWSEBSecurityGroup, is a fixed resource name provided by Elastic Beanstalk. You must add AWSEBSecurityGroup to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is 6379. Finally, this example creates a subnet group for the ElastiCache cluster. We define the name for this resource, declare its type, and add a description and ID of the subnet in the subnet group.

> **Note**
>
> We recommend that you use private subnets for the ElastiCache cluster. For more information about a VPC with a private subnet, see http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Scenario2.html.

```
#This sample requires you to create a separate configuration file that defines
 the custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t1.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      CacheSubnetGroupName:
        Ref: "MyCacheSubnets"
      VpcSecurityGroupIds:
        - Ref: "MyCacheSecurityGroup"
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      VpcId:
        Fn::GetOptionSetting:
          OptionName : "VpcId"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupId:
            Ref: "AWSEBSecurityGroup"
  MyCacheSubnets:
    Type: "AWS::ElastiCache::SubnetGroup"
    Properties:
      Description: "Subnets for ElastiCache"
      SubnetIds:
        Fn::GetOptionSetting:
          OptionName : "CacheSubnets"
Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElastiCache"
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup
- AWS::ElastiCache::SubnetGroup

Next, name the options configuration file `options.config` and define the custom option settings.

> **Note**
> In the following example, replace the example `CacheSubnets` and `VpcId` values with your own subnets and VPC.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t1.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
    CacheSubnets:
      - subnet-1a1a1a1a
      - subnet-2b2b2b2b
      - subnet-3c3c3c3c
    VpcId: vpc-4d4d4d4d
```

These lines tell Elastic Beanstalk to get the values for the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` properties from the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` values in a config file (`options.config` in our example). That file includes an `aws:elasticbeanstalk:customoption` section (under `options_settings`) that contains name-value pairs with sample values. In the example above, `cache.t1.micro`, `1`, `redis`, `6379`, `subnet-1a1a1a1a`, `subnet-2b2b2b2b`, `subnet-3c3c3c3c`, and `vpc-4d4d4d4d` would be used for the values. For more information about `Fn::GetOptionSetting`, see Fn::GetOptionSetting (p. 801).

Example Snippet: SQS, CloudWatch, and SNS

This example adds an Amazon SQS queue and an alarm on queue depth to the environment. The properties that you see in this example are the minimum required properties that you must set for each of these resources. You can download the example at SQS, SNS, and CloudWatch.

> **Note**
> Elastic Beanstalk requires you to use IAM roles to launch an environment and to manage environments and applications. An instance profile is associated with an IAM role that can be configured to provide applications and services access to AWS resources using temporary security credentials. If you are unsure whether you have deployed your application using an instance profile, click **Instances** on the **Configuration** page in the Elastic Beanstalk console. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199). To learn more about instance profiles, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326) and Using IAM Roles with Elastic Beanstalk (p. 332).

Note

This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

Note

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Create a configuration file (e.g., sqs.config) that defines the resources. In this example, we create an SQS queue and define the `VisbilityTimeout` property in the `MySQSQueue` resource. Next, we create an SNS `Topic` and specify that email gets sent to `someone@example.com` when the alarm is fired. Finally, we create a CloudWatch alarm if the queue grows beyond 10 messages. In the `Dimensions` property, we specify the name of the dimension and the value representing the dimension measurement. We use `Fn::GetAtt` to return the value of `QueueName` from `MySQSQueue`.

```
#This sample requires you to create a separate configuration file to define the
 custom options for the SNS topic and SQS queue.
Resources:
  MySQSQueue:
    Type: AWS::SQS::Queue
    Properties:
      VisibilityTimeout:
        Fn::GetOptionSetting:
          OptionName: VisibilityTimeout
          DefaultValue: 30
  AlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: AlarmEmail
              DefaultValue: "nobody@amazon.com"
          Protocol: email
  QueueDepthAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: "Alarm if queue depth grows beyond 10 messages"
      Namespace: "AWS/SQS"
      MetricName: ApproximateNumberOfMessagesVisible
      Dimensions:
        - Name: QueueName
```

```
          Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"] }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold: 10
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: AlarmTopic
      InsufficientDataActions:
        - Ref: AlarmTopic

Outputs :
  QueueURL:
    Description : "URL of newly created SQS Queue"
    Value : { Ref : "MySQSQueue" }
  QueueARN :
    Description : "ARN of newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "Arn"]}
  QueueName :
    Description : "Name newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"]}
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::SQS::Queue
- AWS::SNS::Topic
- AWS::CloudWatch::Alarm

Create a separate configuration file called `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     VisibilityTimeout : 30
     AlarmEmail : "nobody@amazon.com"
```

These lines tell Elastic Beanstalk to get the values for the **VisibilityTimeout and Subscription Endpoint** properties from the **VisibilityTimeout and Subscription Endpoint** values in a config file (options.config in our example) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means 30 and "nobody@amazon.com" would be used for the values. For more information about `Fn::GetOptionSetting`, see Fn::GetOptionSetting (p. 801)

Example: DynamoDB, CloudWatch, and SNS

This section walks you through deploying a sample application to Elastic Beanstalk using eb (an updated command line interface) and Git, and then updating your Elastic Beanstalk to add an DynamoDB table to the Elastic Beanstalk environment. The configuration file sets up the DynamoDB table as a session handler for a PHP-based application using the AWS SDK for PHP 2. To use this example, you must have an IAM instance profile, which is added to the instance(s) in your environment and used to access the DynamoDB table. If you are unsure whether you have deployed your application using an instance profile, click **Instances** on the **Configuration** page in the Elastic Beanstalk console. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199). To learn more about instance profiles, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326) and Using IAM Roles with Elastic Beanstalk (p. 332). To see an example creating a custom policy for DynamoDB using instance profiles, see Example: Granting Permissions to Elastic Beanstalk Applications to Access DynamoDB (p. 335).

> **Note**
> This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

Step 1: Set Up Your Git Repository

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Step 2: Configure Elastic Beanstalk

Elastic Beanstalk needs the following information to deploy an application:

- AWS access key ID
- AWS secret key
- Service region
- Application name
- Environment name
- Solution stack

Use the `eb init` command, and the EB CLI will prompt you to enter this information. If a default value is available, and you want to use it, press **Enter**.

If you get a "command not found" error when you run `eb --version`, ensure that the directory that contains the `eb` executable is in your PATH environment variable

Linux, OS X, or Unix

```
$ export PATH=/eb/executable/directory:$PATH
```

Windows

```
> set PATH=%PATH%;C:\eb\executable\directory
```

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the access key ID, type your access key ID. To get your access key ID, see How Do I Get Security Credentials? in the *AWS General Reference*.

```
Enter your AWS Access Key ID (current value is "AKIAIOSFODNN7EXAMPLE"):
```

3. When you are prompted for the secret access key, type your secret access key. To get your secret access key, see How Do I Get Security Credentials? in the *AWS General Reference*.

   ```
   Enter your AWS Secret Access Key (current value is "wJalrXUtnFEMI/K7MDENG/bPxR
   fiCYEXAMPLEKEY"):
   ```

4. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
5. When you are prompted for the Elastic Beanstalk application name, type the name of the application. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use **`HelloWorld`**.

   ```
   Enter an Elastic Beanstalk application name (auto-generated value is "win
   dows"): HelloWorld
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.

6. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter an Elastic Beanstalk environment name (current value is "HelloWorld-
   env"):
   ```

   Note
   If you have a space in your application name, make sure you do not have a space in your environment name.

7. When you are prompted for the solution stack, type the number of the solution stack you want. For more information about solution stacks, see Supported Platforms (p. 24). For this example, we'll use **64bit Amazon Linux running PHP 5.4**.
8. When you are prompted to create an Amazon RDS database, type **`y`** or **`n`**. For more information about using Amazon RDS, see Using Elastic Beanstalk with Amazon RDS (p. 294). For this example, we'll type **`n`**.

   ```
   Create RDS instance? [y/n]: n
   ```

9. When you are prompted to enter your instance profile name, you can choose to create a default instance profile or use an existing instance profile. Using an instance profile enables IAM users and AWS services to gain access to temporary security credentials to make AWS API calls. Using instance profiles prevents you from having to store long-term security credentials on the EC2 instance. For more information about instance profiles, see Service Roles, Instance Profiles, and User Policies (p. 19). For this example, we'll use **`Create a default instance profile`**.

   You should see a confirmation that your AWS Credential file was successfully updated.

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key.

Step 3: Create an Application

Run `eb create` to create an Elastic Beanstalk application and launch an environment running a sample web application.

To create a new application and environment with the EB CLI

1. From the directory where you created your Elastic Beanstalk repository, type the following command:

```
eb create
```

2. Enter an application name, environment name, and CNAME (URL prefix) when prompted.

It may take several minutes to complete this process. Elastic Beanstalk provides status updates during the process. If at any time you want to stop receiving status updates, press **Ctrl+C**. When the environment status is Green, Elastic Beanstalk outputs a URL for the application.

Step 4: View the Application

In the previous step, you created an application and deployed it to Elastic Beanstalk. After the environment is ready and its status is Green, Elastic Beanstalk provides a URL to view the application. In this step, you can check the status of the environment to make sure it is set to Green and then copy and paste the URL to view the application.

Use the `status` command to check the environment status, and then use the URL to view the application.

To view the application

1. From the directory where you created your local repository, type the following command:

```
eb status --verbose
```

   Elastic Beanstalk displays the environment status. If the environment is set to Green, Elastic Beanstalk displays the URL for the application. If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB information is displayed.
2. Copy and paste the URL into your web browser to view your application.

Step 5: Update the Application

Next, we add the files for the sample application and configuration files that will set up AWS resources that the application depends on.

- The sample application, index.php
- A configuration file, `dynamodb.config`, to create and configure a DynamoDB table and other AWS resources as well as install software on the EC2 instances that host the application in an Elastic Beanstalk environment
- An options setting file, `options.config`, that overrides the defaults in dynamodb.config with specific settings for this particular installation

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

You can download the sample that we'll use in this step at DynamoDB Session Support Example.

To update the application

1. In the directory where you created your local repository, use your favorite text editor to create an index.php file and paste the following PHP code.

```
<?php

// Include the SDK using the Composer autoloader
require '../vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Grab the session table name and region from the configuration file
list($tableName, $region) = file(__DIR__ . '/../sessiontable');
$tableName = rtrim($tableName);
$region = rtrim($region);

// Create a DynamoDB client and register the table as the session handler
$dynamodb = DynamoDbClient::factory(array('region' => $region));
$handler = $dynamodb->registerSessionHandler(array('table_name' => $tableName,
 'hash_key' => 'username'));

// Grab the instance ID so we can display the EC2 instance that services
the request
$instanceId = file_get_contents("http://169.254.169.254/latest/meta-
data/instance-id");
?>
<h1>Elastic Beanstalk PHP Sessions Sample</h1>
<p>This sample application shows the integration of the Elastic Beanstalk
PHP
container and the session support for DynamoDB from the AWS SDK for PHP 2.
Using DynamoDB session support, the application can be scaled out across
multiple web servers. For more details, see the
<a href="http://aws.amazon.com/php/">PHP Developer Center</a>.</p>

<form id="SimpleForm" name="SimpleForm" method="post" action="index.php">
<?php
echo 'Request serviced from instance ' . $instanceId . '<br/>';
echo '<br/>';

if (isset($_POST['continue'])) {
  session_start();
  $_SESSION['visits'] = $_SESSION['visits'] + 1;
  echo 'Welcome back ' . $_SESSION['username'] . '<br/>';
  echo 'This is visit number ' . $_SESSION['visits'] . '<br/>';
  session_write_close();
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';

  echo '<input type="Submit" value="Delete Session" name="killsession"
id="killsession"/>';
} elseif (isset($_POST['killsession'])) {
  session_start();
  echo 'Goodbye ' . $_SESSION['username'] . '<br/>';
  session_destroy();
  echo 'Username: <input type="text" name="username" id="username" 
size="30"/><br/>';
  echo '<br/>';
  echo '<input type="Submit" value="New Session" name="newsession" 
id="newsession"/>';
} elseif (isset($_POST['newsession'])) {
  session_start();
  $_SESSION['username'] = $_POST['username'];
  $_SESSION['visits'] = 1;
  echo 'Welcome to a new session ' . $_SESSION['username'] . '<br/>';
  session_write_close();
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';

  echo '<input type="Submit" value="Delete Session" name="killsession" 
id="killsession"/>';
} else {
  echo 'To get started, enter a username.<br/>';
  echo '<br/>';
  echo 'Username: <input type="text" name="username" id="username" 
size="30"/><br/>';
  echo '<input type="Submit" value="New Session" name="newsession" 
id="newsession"/>';
}
?>
</form>
```

2. Create an `.ebextensions` directory in the top-level directory of your source bundle.
3. Create a configuration file named `dynamodb.config`, paste the code in the listing below, and save the file in the `.ebextensions` top-level directory of your source bundle.

```
Resources:
  SessionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      KeySchema:
        HashKeyElement:
          AttributeName:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyName
              DefaultValue: "username"
          AttributeType:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyType
              DefaultValue: "S"
      ProvisionedThroughput:
        ReadCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnits
            DefaultValue: 1
        WriteCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnits
            DefaultValue: 1

  SessionWriteCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
     AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName"
 }, " write capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedWriteCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionReadCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
     AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName"
 }, " read capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedReadCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionThrottledRequestsAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
     AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName"
 }, ": requests are being throttled." ]]}
      Namespace: AWS/DynamoDB
      MetricName: ThrottledRequests
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold:
        Fn::GetOptionSetting:
          OptionName: SessionThrottledRequestsThreshold
          DefaultValue: 1
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionAlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: SessionAlarmEmail
              DefaultValue: "nobody@amazon.com"
          Protocol: email

files:
  "/var/app/sessiontable":
    mode: "000444"
    content: |
      `{"Ref" : "SessionTable"}`
      `{"Ref" : "AWS::Region"}`

  "/var/app/composer.json":
    mode: "000744"
    content:
      {
        "require": {
           "aws/aws-sdk-php": "*"
        }
      }

container_commands:
 "1-install-composer":
   command: "cd /var/app; curl -s http://getcomposer.org/installer | php"
 "2-install-dependencies":
   command: "cd /var/app; php composer.phar install"
 "3-cleanup-composer":
   command: "rm -Rf /var/app/composer.*"
```

In the sample configuration file, we first create the DynamoDB table and configure the primary key structure for the table and the capacity units to allocate sufficient resources to provide the requested throughput. Next, we create CloudWatch alarms for `WriteCapacity` and `ReadCapacity`. We create an SNS topic that sends email to "nobody@amazon.com" if the alarm thresholds are breached.

After we create and configure our AWS resources for our environment, we need to customize the EC2 instances. We use the `files` key to pass the details of the DynamoDB table to the EC2 instances in our environment as well as add a "require" in the `composer.json` file for the AWS SDK for PHP

2. Finally, we run container commands to install composer, the required dependencies, and then remove the installer. The example snippet looks like the following.

For more information about the resources used in this example, see the following references:

- AWS::DynamoDB::Table
- AWS::CloudWatch::Alarm
- AWS::SNS::Topic

4. Create a configuration file named `options.config`. Replace <email here> with the email where you want alarm notifications sent, and save the file in the **.ebextensions** top-level directory of your source bundle.

```
option_settings:
 "aws:elasticbeanstalk:customoption":
    SessionHashKeyName                      : username
    SessionHashKeyType                      : S
    SessionReadCapacityUnits                : 1
    SessionReadCapacityUnitsAlarmThreshold  : 240
    SessionWriteCapacityUnits               : 1
    SessionWriteCapacityUnitsAlarmThreshold : 240
    SessionThrottledRequestsThreshold       : 1
    SessionAlarmEmail                       : <email here>
```

After you've made your changes, your file should look like the following.

```
option_settings:
 "aws:elasticbeanstalk:customoption":
    SessionHashKeyName                      : username
    SessionHashKeyType                      : S
    SessionReadCapacityUnits                : 1
    SessionReadCapacityUnitsAlarmThreshold  : 240
    SessionWriteCapacityUnits               : 1
    SessionWriteCapacityUnitsAlarmThreshold : 240
    SessionThrottledRequestsThreshold       : 1
    SessionAlarmEmail                       : <email here>
```

The options.config file contains the values used for some of the variables defined in `dynamodb.config`. For example, `dynamodb.config` contains the following lines.

```
Subscription:
  - Endpoint:
      Fn::GetOptionSetting:
        OptionName: SessionAlarmEmail
        DefaultValue: "nobody@amazon.com"
```

These lines that tell Elastic Beanstalk to get the value for the **Endpoint** property from the **SessionAlarmEmail** value in a config file (`options.config` in our sample application) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means **SessionAlarmEmail** would be assigned the value `nobody@amazon.com`.

5. Add your files to your local Git repository, and then commit your change.

```
git add .
git commit -m "eb configuration"
```

> **Note**
> For information about Git commands, go to Git - Fast Version Control System.

6. Create an application version matching your local repository and deploy to the Elastic Beanstalk environment if specified.

```
git aws.push
```

7. Use the `eb status --verbose` command to check your environment status. When your environment is green and ready, refresh your web browser to view your updated application.

Example Snippets

The following is a list of example configuration files that you can use to customize your Elastic Beanstalk environments:

- DynamoDB, CloudWatch, and SNS
- Elastic Load Balancing and CloudWatch
- ElastiCache
- RDS and CloudWatch
- SQS, SNS, and CloudWatch

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Customizing Software on Linux Servers

You may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages from the yum repository—or they could be configuration files such as a replacement for httpd.conf to override specific settings that are defaulted by Elastic Beanstalk.

This section describes the type of information you can include in a configuration file to customize the software on your EC2 instances running Linux. For general information on customizing and configuring your Elastic Beanstalk environments, see Elastic Beanstalk Environment Configuration (p. 99). For information on customizing software on your EC2 instances running Windows, see Customizing Software on Windows Servers (p. 160).

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

- Packages (p. 148)
- Sources (p. 149)

- Files (p. 150)
- Users (p. 151)
- Groups (p. 152)
- Commands (p. 152)
- Container_commands (p. 153)
- Services (p. 155)
- Option_settings (p. 156)

The order in which these are processed are as follows:

1. Packages
2. Groups
3. Users
4. Sources
5. Files
6. Commands
7. Services
8. Container_commands

> **Note**
> **Option_settings** are not processed on the Amazon EC2 instance. Instead, they are extracted from the configuration file before the contents of the configuration file are executed.

Packages

You can use the packages key to download and install prepackaged applications and components.

Syntax

```
packages:
  name of package manager:
    package name: version
```

Supported Package Formats

Elastic Beanstalk currently supports the following package managers: yum, rubygems, python, and rpm. Packages are processed in the following order: rpm, yum, and then rubygems and python. There is no ordering between rubygems and python, and packages within each package manager are not guaranteed to be installed in any order. Use a package manager supported by your operating system.

> **Note**
> Elastic Beanstalk supports two underlying package managers for Python, pip and easy_install. However, in the syntax of the configuration file, you must specify the package manager name as `python`. When you use a configuration file to specify a Python package manager, Elastic Beanstalk uses Python 2.6. If your application relies on a different version of Python, you can specify the packages to install in a `requirements.txt` file. For more information, see Customizing and Configuring a Python Container (p. 764).

Specifying Versions

Within each package manager, each package is specified as a package name and a list of versions. The version can be a string, a list of versions, or an empty string or list. An empty string or list indicates that you want the latest version. For rpm manager, the version is specified as a path to a file on disk or a URL. Relative paths are not supported.

If you specify a version of a package, Elastic Beanstalk attempts to install that version even if a newer version of the package is already installed on the instance. If a newer version is already installed, the deployment fails. Some package managers support multiple versions, but others may not. Please check the documentation for your package manager for more information. If you do not specify a version and a version of the package is already installed, Elastic Beanstalk does not install a new version—it assumes that you want to keep and use the existing version.

Example Snippet

The following snippet specifies a version URL for rpm, requests the latest version from yum, and version 0.10.2 of chef from rubygems.

```
packages:
  yum:
    libmemcached: []
    ruby-devel: []
    gcc: []
  rpm:
    epel: http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-
4.noarch.rpm
  rubygems:
    chef: '0.10.2'
```

Sources

You can use the sources key to download an archive file and unpack it in a target directory on the EC2 instance. Sources does not automatically build the unpacked sources.

Syntax

```
sources:
  target directory: location of archive file
```

Supported Formats

Supported formats are tar, tar+gzip, tar+bz2, and zip. You can reference external locations such as Amazon Simple Storage Service (Amazon S3) (e.g., http://s3.amazonaws.com/mybucket/myobject).

Example Snippet

The following example downloads a .zip file from an Amazon S3 bucket and unpacks it into /etc/myapp:

```
sources:
  /etc/myapp: http://s3.amazonaws.com/mybucket/myobject
```

Files

You can use the files key to create files on the EC2 instance. The content can be either inline in the configuration file, or the content can be pulled from a URL. The files are written to disk in lexicographic order. You can reference external locations such as Amazon S3 (e.g., http://s3.amazonaws.com/mybucket/myobject). The following table lists the supported keys.

Syntax

```
files:
  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     source: URL
     authentication: authentication name:

  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     content: |
    this is my content
     encoding: encoding format
     authentication: authentication name:
```

Options

Key	Description
content	A string.
source	A URL to load the file from. This option cannot be specified with the content key.
encoding	The encoding format. Only used if the content is a string. Encoding is not applied if you are using a source. Valid values: plain | base64
group	The name of the owning group for this file.
owner	The name of the owning user for this file.
mode	A six-digit octal value representing the mode for this file (e.g, "000444"). The first three digits are used for symlinks and the last three digits are used for setting permissions.
authentication	The name of an authentication method to use. This overrides any default authentication.

Example Snippet

```
files:
  "/home/ec2-user/myfile" :
    mode: "000755"
    owner: root
    group: root
    source: http://foo.bar/myfile

  "/home/ec2-user/myfile2" :
    mode: "000755"
    owner: root
    group: root
    content: |
      # this is my file
      # with content
```

Example using a symlink. This creates a link /tmp/myfile2.txt that points at the existing file /tmp/myfile1.txt.

```
files:
  "/tmp/myfile2.txt" :
    mode: "120400"
    content: "/tmp/myfile1.txt"
```

Users

You can use the users key to create Linux/UNIX users on the EC2 instance.

Syntax

```
users:
  name of user:
    groups:
      - name of group
    uid: "id of the user"
    homeDir: "user's home directory"
```

Options

Key	Description
uid	A user ID. The creation process fails if the user name exists with a different user ID. If the user ID is already assigned to an existing user, the operating system may reject the creation request.
groups	A list of group names. The user is added to each group in the list.
homeDir	The user's home directory.

Users are created as noninteractive system users with a shell of /sbin/nologin. This is by design and cannot be modified.

Example Snippet

```
users:
  myuser:
    groups:
      - group1
      - group2
    uid: "50"
    homeDir: "/tmp"
```

Groups

You can use the groups key to create Linux/UNIX groups and to assign group IDs. To create a group, add a new key-value pair that maps a new group name to an optional group ID. The groups key can contain one or more group names. The following table lists the available keys.

Syntax

```
groups:
  name of group:
  name of group:
    gid: "group id"
```

Options

Key	Description
gid	A group ID number. If a group ID is specified, and the group already exists by name, the group creation will fail. If another group has the specified group ID, the operating system may reject the group creation.

Example Snippet

The following snippet specifies a group named groupOne without assigning a group ID and a group named groupTwo that specified a group ID value of 45.

```
groups:
  groupOne:
  groupTwo:
    gid: "45"
```

Commands

You can use the commands key to execute commands on the EC2 instance. The commands are processed in alphabetical order by name, and they run before the application and web server are set up and the application version file is extracted.

Syntax

```
commands:
  test_command:
    command: command to run
    cwd: working directory
    env:
      variable name: variable value
    test: conditions for command
    ignoreErrors: true
```

Options

Key	Description
command	Required. Either an array or a string specifying the command to run. If you use an array, you do not need to escape space characters or enclose command parameters in quotes.
env	Optional. Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.
cwd	Optional. The working directory. By default, Elastic Beanstalk attempts to find the directory location of your project. If not found, then "/" is used.
test	Optional. A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command (e.g., a bash script) contained in the `command` key.
ignoreErrors	Optional. A boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

Example Snippet

The following example snippet runs a python script.

```
commands:
  python_install:
    command: myscript.py
    cwd: /home/ec2-user
    env:
      myvarname: myvarvalue
    test: '[ ! /usr/bin/python ] && echo "python not installed"'
```

Container_commands

You can use the `container_commands` key to execute commands for your container. The commands in `container_commands` are processed in alphabetical order by name. They run after the application and web server have been set up and the application version file has been extracted, but before the application version is deployed. They also have access to environment variables such as your AWS security credentials. Additionally, you can use `leader_only`. One instance is chosen to be the leader in an Auto Scaling group. If the `leader_only` value is set to `true`, the command runs only on the instance that is marked as the leader.

Syntax

```
container_commands:
 name of container_command:
    command: "command to run"
    leader_only: true
 name of container_command:
    command: "command to run"
```

Options

Key	Description
`command`	Required. Either an array or a string specifying the command to run. If you use an array, you do not need to escape space characters or enclose command parameters in quotes.
`env`	Optional. Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.
`cwd`	Optional. The working directory. By default, this is the directory of the unzipped application.
`leader_only`	Optional. Sets an instance in the Auto Scaling group to be the leader. If the `leader_only` value is set to `true`, the command runs only on the instance that is marked as the leader. The leader runs first.
`test`	Optional. A command that must return the value `true` in order for Elastic Beanstalk to process the command contained in the command key. If this option is set to `true`, it overrides the `leader_only` setting.
`ignoreErrors`	Optional. A boolean value that determines if other commands should run if the command contained in the command key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

Example Snippet

The following is an example snippet.

```
container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  99customize:
    command: "scripts/customize.sh"
```

Services

You can use the services key to define which services should be started or stopped when the instance is launched. The `services` key also allows you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
  sysvinit:
    name of service:
      enabled: true
      ensureRunning: true
      files: "file name"
      sources: "directory"
      packages:
        name of package manager:
          package name: version
      commands:
        name of command
```

Options

The following table lists the supported keys.

Key	Description
`ensureRunning`	Set to `true` to ensure that the service is running after Elastic Beanstalk finishes. Set to `false` to ensure that the service is not running after Elastic Beanstalk finishes. Omit this key to make no changes to the service state.
`enabled`	Set to `true` to ensure that the service is started automatically upon boot. Set to `false` to ensure that the service is not started automatically upon boot. Omit this key to make no changes to this property.
`files`	A list of files. If Elastic Beanstalk changes one directly via the files block, the service is restarted.
`sources`	A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.
`packages`	A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.
`commands`	A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example Snippet

The following is an example snippet:

```
services:
  sysvinit:
    myservice:
      enabled: true
      ensureRunning: true
```

Option_settings

`Option_settings` enables you to modify the Elastic Beanstalk configuration and define variables that can be retrieved from your application using environment variables. The following table displays the namespaces that are supported for each container type. Some namespaces allow you to extend the number of parameters, and specify the parameter names. For a list of configuration settings, see Configuration Options (p. 103).

> **Note**
> You cannot use the `option_settings` key in the configuration file to overwrite any option settings you already configured with the management console, command-line tools, a toolkit, or the API. You must first remove those option settings from the environment configuration. You can reset the value of an option setting to a null value or, if the setting has one, its default value by using the AWS CLI `update-environment` command. Use the `--options-to-remove` parameter with the command to list all option settings that you want to reset. If you launch a new environment from a configuration template that has option settings you want to overwrite, you can do the same using the `create-environment` command.

Syntax

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - option_name:  option name
    value:  option value
```

Options

Container	Namespace	Extend
Java	`aws:elasticbeanstalk:application:environment` `aws:elasticbeanstalk:container:tomcat:jvmoptions`	Yes Yes

Node.js	`aws:elasticbeanstalk:application:environment`
	`aws:elasticbeanstalk:container:nodejs`
	`aws:elasticbeanstalk:container:nodejs:staticfiles`
		Yes
		No
		Yes

Container	Namespace	Extend
PHP	aws:elasticbeanstalk:application:environment aws:elasticbeanstalk:container:php:phpini	Yes No
Python	aws:elasticbeanstalk:application:environment aws:elasticbeanstalk:container:python aws:elasticbeanstalk:container:python:staticfiles	Yes No Yes
Ruby	aws:elasticbeanstalk:application:environment	Yes

Note
If you do not specify a namespace, the default used is `aws:elasticbeanstalk:application:environment`.

Example Snippet

The following is an example snippet.

```
option_settings:
  - namespace:  aws:elasticbeanstalk:container:tomcat:jvmoptions
    option_name:  Xmx
    value:  256m
  - option_name: myparam1
    value: somevalue
```

Accessing Environment Variables

The parameters specified in the `option_settings` section of the configuration file are passed in as environment variables to the EC2 instances. For coding examples, see the following sections:

- Java
- .NET
- Node.js
- PHP
- Python
- Ruby

Example: Using Custom Amazon CloudWatch Metrics

Amazon CloudWatch is a web service that enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. You can define custom metrics for your own use, and Elastic Beanstalk will push those metrics to Amazon CloudWatch. Once Amazon CloudWatch contains your custom metrics, you can view those in the Amazon CloudWatch console.

The Amazon CloudWatch Monitoring Scripts for Linux are available to demonstrate how to produce and consume Amazon CloudWatch custom metrics. The scripts comprise a fully functional example that reports memory, swap, and disk space utilization metrics for an Amazon Elastic Compute Cloud (Amazon EC2) Linux instance. For more information about the Amazon CloudWatch Monitoring Scripts, go to Amazon CloudWatch Monitoring Scripts for Linux in the *Amazon CloudWatch Developer Guide*.

> **Note**
> Elastic Beanstalk Enhanced Health Reporting (p. 250) has native support for publishing a wide range of instance and environment metrics to CloudWatch. See Publishing Amazon CloudWatch Custom Metrics for an Environment (p. 263) for details.

Topics

.ebextensions Configuration File

This example uses commands and option settings in an .ebextensions configuration file to download, install, and run monitoring scripts provided by Amazon CloudWatch.

To use this sample, save it to a file named `cloudwatch.config` in a directory named `.ebextensions` at the top level of your project directory, then deploy your application using the AWS Management Console (include the .ebextensions directory in your source bundle (p. 43)) or the EB CLI (p. 384).

For more information about `.ebextensions`, see Using Configuration Files (p. 99).

.ebextensions/cloudwatch.config

```
sources:
  /opt/cloudwatch: http://ec2-downloads.s3.amazonaws.com/cloudwatch-
samples/CloudWatchMonitoringScripts-v1.1.0.zip

commands:
  00-installpackages:
    command: yum install -y perl-Switch perl-Sys-Syslog perl-LWP-Protocol-https

container_commands:
  01-setupcron:
    command: |
      echo '*/5 * * * * root perl /opt/cloudwatch/aws-scripts-mon/mon-put-in
stance-data.pl `{"Fn::GetOptionSetting" : { "OptionName" : "CloudWatchMetrics",
 "DefaultValue" : "--mem-util --disk-space-util --disk-path=/" }}` >>
/var/log/cwpump.log 2>&1' > /etc/cron.d/cwpump
  02-changeperm:
    command: chmod 644 /etc/cron.d/cwpump
  03-changeperm:
    command: chmod u+x /opt/cloudwatch/aws-scripts-mon/mon-put-instance-data.pl

option_settings:
  "aws:autoscaling:launchconfiguration" :
    IamInstanceProfile : "aws-elasticbeanstalk-ec2-role"
  "aws:elasticbeanstalk:customoption" :
    CloudWatchMetrics : "--mem-util --mem-used --mem-avail --disk-space-util -
-disk-space-used --disk-space-avail --disk-path=/ --auto-scaling"
```

> **Note**
>
> After you verify the configuration file works, you can conserve disk usage by changing the command redirect from a log file (>> /var/log/cwpump.log 2>&1') to /dev/null (> /dev/null).

Permissions

In order to publish custom Amazon CloudWatch metrics, the instances in your environment need permission to use CloudWatch. You can grant permissions to your environment's instances by adding them to the environment's instance profile (p. 20). You can add permissions to the instance profile before or after deploying your application.

To grant permissions to publish CloudWatch metrics

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**.
3. Choose your environment's instance profile role. By default, this is **aws-elasticbeanstalk-ec2-role** when you create an environment with the AWS Management Console or EB CLI (p. 384).
4. Under **Inline Policies** in the **Permissions** section, choose **Create Role Policy**.
5. Click **Custom Policy** and then add the following permissions:

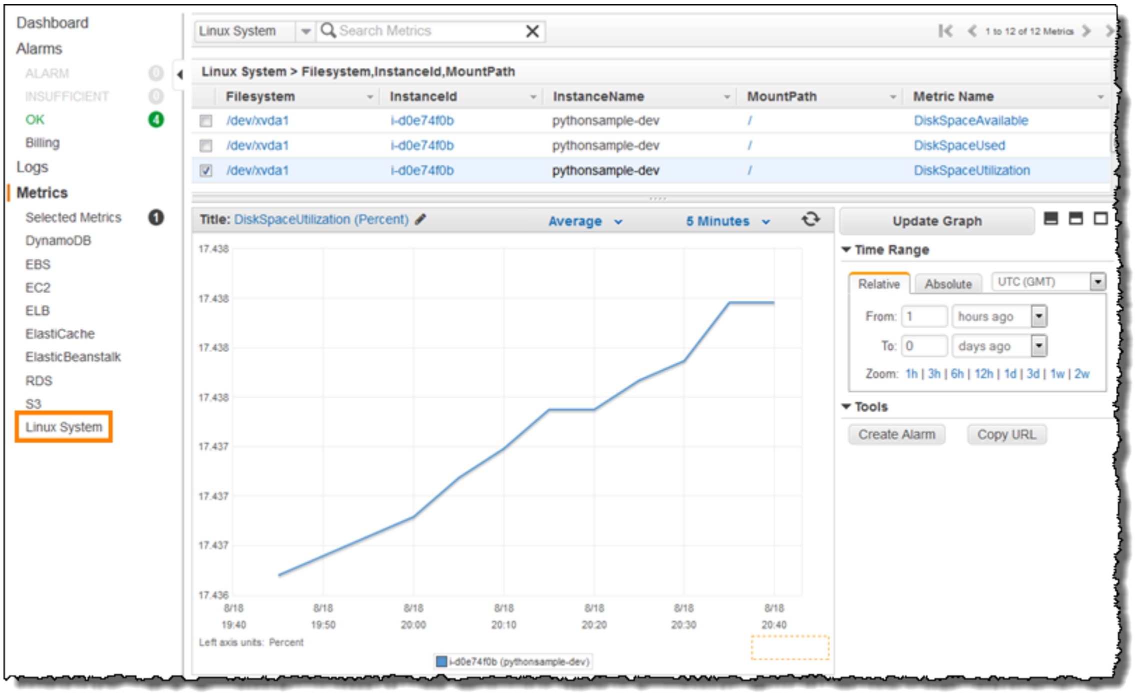
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData",
        "ec2:DescribeTags"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

6. Click **Apply Policy**.

   For more information on managing policies, go to Managing IAM Policies in *Using AWS Identity and Access Management*.

Viewing Metrics in the CloudWatch Console

After deploying the CloudWatch configuration file to your environment, check the Amazon CloudWatch console to view your metrics. Custom metrics will have the prefix **Linux System**.



Linux System > Filesystem,InstanceId,MountPath

	Filesystem	InstanceId	InstanceName	MountPath	Metric Name
□	/dev/xvda1	i-d0e74f0b	pythonsample-dev	/	DiskSpaceAvailable
□	/dev/xvda1	i-d0e74f0b	pythonsample-dev	/	DiskSpaceUsed
☑	/dev/xvda1	i-d0e74f0b	pythonsample-dev	/	DiskSpaceUtilization

Customizing Software on Windows Servers

You may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages or services that need to be run. For general information on customizing and configuring your Elastic Beanstalk environments, see Elastic Beanstalk Environment Configuration (p. 99).

This section describes the type of information you can include in a configuration file to customize the software on your EC2 instances running Windows:

- Packages (p. 161)
- Sources (p. 161)
- Files (p. 162)
- Commands (p. 163)
- Container_commands (p. 164)
- Services (p. 165)
- Option_settings (p. 166)

The order in which these are processed are as follows:

1. Packages
2. Files
3. Commands
4. Services
5. Container Commands

Note

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

When creating configuration files, we recommend you use an editor other than Visual Studio since configuration files require spaces instead of tabs.

Packages

You can use the packages key to download and install prepackaged applications and components.

Syntax

```
packages:
  name of package manager:
    package name: version
```

Supported Package Formats

Elastic Beanstalk currently supports MSI packages.

Specifying Versions

Within the package manager, the package is specified as a package name and a URL to the software.

Elastic Beanstalk attempts to install the specified version even if a newer version of the package is already installed on the instance. Some package managers support this, but others may not. Please check the documentation for your package manager for more information. If you specify a version that is already installed, the deployment fails.

Example Snippet

The following snippet specifies a URL to download mysql.

```
packages:
  msi:
    mysql: http://dev.mysql.com/get/Downloads/Connector-Net/mysql-connector-
net-6.6.5.msi/from/http://cdn.mysql.com/
```

Sources

You can use the sources key to download an archive file and unpack it in a target directory on the EC2 instance. Sources does not automatically build the unpacked sources.

Syntax

```
sources:
  target directory: location of archive file
```

Supported Formats

Elastic Beanstalk currently supports .zip format. You can reference external locations such as Amazon Simple Storage Service (Amazon S3) (e.g., http://s3.amazonaws.com/mybucket/myobject).

Example Snippet

The following example downloads a .zip file from an Amazon S3 bucket and unpacks it into `c:/myproject/myapp`:

```
sources:
  "c:/myproject/myapp": http://s3.amazonaws.com/mybucket/myobject.zip
```

Files

You can use the files key to create files on the EC2 instance. The content can be either inline in the configuration file, or the content can be pulled from a URL. The files are written to disk in lexicographic order. You can reference external locations such as Amazon S3 (e.g., http://s3.amazonaws.com/mybucket/myobject). The following table lists the supported keys.

Syntax

```
files:
  "target file location on disk":
     source: URL
     authentication: authentication name:

  "target file location on disk":
     content: |
       this is my content
     encoding: encoding format
     authentication: authentication name:
```

Options

Key	Description
`content`	A string.
`source`	A URL to load the file from. This option cannot be specified with the content key.
`encoding`	The encoding format. Only used if the content is a string. Encoding is not applied if you are using a source. Valid values: `plain` | `base64`
`authentication`	The name of an authentication method to use. This overrides any default authentication.

Example Snippet

```
files:
  "c:\\targetdirectory\\targetfile.txt":
    source: http://foo.bar/myfile

  "c:/targetdirectory/targetfile.txt":
    content: |
      # this is my file
      # with content
```

Note
If you use a "\" in your file path, make sure you use "\\" as shown in the example.

Commands

You can use the commands key to execute commands on the EC2 instance. The commands are processed in alphabetical order by name, and they run before the application and web server are set up and the application version file is extracted.

Syntax

```
commands:
  test_command:
    command: command to run
    cwd: working directory
    env:
      variable name: variable value
    ignoreErrors: true
    waitAfterCompletion: number of seconds
```

Options

Key	Description
`command`	Required. Either an array or a string specifying the command to run. If you use an array, you do not need to escape space characters or enclose command parameters in quotes.
`cwd`	Optional. The working directory. By default, Elastic Beanstalk attempts to find the directory location of your project. If not found, it uses "c:\Windows\System32" as the default.
`env`	Optional. Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.
`ignoreErrors`	Optional. A boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.
`test`	Optional. A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command contained in the `command` key.

Key	Description
`waitAfterCompletion`	Optional. Seconds to wait after the command completes before running the next command. If the system requires a reboot after the command completes, the system reboots after the specified number of seconds elapses. If the system reboots as a result of a command, Elastic Beanstalk will recover. The default value is **`60`** seconds. You can also specify **`forever`**, but the system must reboot before you can run another command.

Example Snippet

The following example snippet runs a command to copy the set values that are currently defined to the specified file. Elastic Beanstalk would run the next command, if there were one, immediately after the command completes. Or, if the command required a reboot, the system would reboot immediately after the command completes.

```
commands:
  test:
    command: set > c:\\myapp\\set.txt
    waitAfterCompletion: 0
```

Container_commands

You can use the `container_commands` key to execute commands for your container. The commands in `container_commands` are processed in alphabetical order by name. They run before the application version has been deployed. They also have access to environment variables such as your AWS security credentials. Additionally, you can use `leader_only`. One instance is chosen to be the leader in an Auto Scaling group. If the `leader_only` value is set to `true`, the command runs only on the instance that is marked as the leader.

Syntax

```
container_commands:
 name of container_command:
    command: command to run
    leader_only: true
 name of container_command:
    command: command to run
```

Options

Key	Description
`command`	Required. Either an array or a string specifying the command to run. If you use an array, you do not need to escape space characters or enclose command parameters in quotes.
`env`	Optional. Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.
`cwd`	Optional. The working directory. By default, Elastic Beanstalk attempts to find the directory location of your project. If not found, it uses "c:\" as the default.

Key	Description
leader_only	Optional. Sets an instance in the Auto Scaling group to be the leader. If the leader_only value is set to true, the command runs only on the instance that is marked as the leader. The leader runs first.
test	Optional. A command that must return the value true in order for Elastic Beanstalk to process the command contained in the command key. If this option is set to true, it overrides the leader_only setting.
ignoreErrors	Optional. A boolean value that determines if other commands should run if the command contained in the command key fails (returns a nonzero value). Set this value to true if you want to continue running commands even if the command fails. Set it to false if you want to stop running commands if the command fails. The default value is false.

Example Snippet

The following is an example that runs a command to copy the set values that are currently defined to the specified file. It will only run the command on one instance, and it will reboot immediately after the command completes.

```
container_commands:
  foo:
    command: set > c:\\myapp\\set.txt
    leader_only: true
    waitAfterCompletion: 0
```

Services

You can use the services key to define which services should be started or stopped when the instance is launched. The services key also allows you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
  windows:
    name of service:
      enabled: true
      ensureRunning: true
      files: "file name"
      sources: "directory"
      packages:
        name of package manager:
          package name: version
      commands:
        name of command:
```

Options

The following table lists the supported keys.

Key	Description
ensureRunning	Set to true to ensure that the service is running after Elastic Beanstalk finishes. Set to false to ensure that the service is not running after Elastic Beanstalk finishes. Omit this key to make no changes to the service state.
enabled	Set to true to ensure that the service is started automatically upon boot. Set to false to ensure that the service is not started automatically upon boot. Omit this key to make no changes to this property.
files	A list of files. If Elastic Beanstalk changes one directly via the files block, the service is restarted.
sources	A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.
packages	A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.
commands	A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example Snippet

The following is an example snippet:

```
services:
  windows:
    myservice:
      enabled: true
      ensureRunning: true
```

Option_settings

Option_settings enables you to modify the Elastic Beanstalk configuration and define variables that can be retrieved from your application. The following table displays the namespaces that are supported. Some namespaces allow you to extend the number of parameters, and specify the parameter names. For a list of configuration settings, see Configuration Options (p. 103).

> **Note**
> You cannot use the option_settings key in the configuration file to overwrite any option settings you already configured with the management console, command-line tools, a toolkit, or the API. You must first remove those option settings from the environment configuration. You can reset the value of an option setting to a null value or, if the setting has one, its default value by using the AWS CLI update-environment command. Use the --options-to-remove parameter with the command to list all option settings that you want to reset. If you launch a new environment from a configuration template that has option settings you want to overwrite, you can do the same using the create-environment command.

Syntax

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - option_name:  option name
    value:  option value
```

Options

Container	Namespace	Extend
.NET	aws:elasticbeanstalk:application:environment aws:elasticbeanstalk:container:net:apppool	Yes No

Note
If you do not specify a namespace, the default used is aws:elasticbeanstalk:application:environment.

Example Snippet

The following is an example snippet.

```
option_settings:
  - option_name: MYPARAM
    value: myvalue
```

Accessing Environment Variables

The parameters specified in the `option_settings` section of the configuration file are passed in and used as application settings.

You might have a code snippet that looks similar to the following to access the keys and parameters:

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string param1 = appConfig["PARAM1"];
```

Note
Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Example: Using Custom Amazon CloudWatch Metrics

Amazon CloudWatch is a web service that enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. You can define custom metrics for your own use, and Elastic Beanstalk will push those metrics to Amazon CloudWatch. Once Amazon CloudWatch contains your custom metrics, you can view those in the Amazon CloudWatch console.

The Amazon CloudWatch Monitoring Scripts for Windows are available to demonstrate how to produce and consume Amazon CloudWatch custom metrics. The scripts comprise a fully functional example that reports memory, swap, and disk space utilization metrics for an Amazon Elastic Compute Cloud (Amazon EC2) Windows instance. For more information about the Amazon CloudWatch Monitoring Scripts, go to Amazon CloudWatch Monitoring Scripts for Windows in the *Amazon CloudWatch Developer Guide*.

This section walks you through how to deploy a sample application to Elastic Beanstalk using the AWS Toolkit for Visual Studio, and then add custom Amazon CloudWatch metrics using a configuration file.

Step 1: Deploy a Sample Application

First, let's get a sample application running in an Elastic Beanstalk environment using the AWS Toolkit for Visual Studio. For instructions, see Develop, Test, and Deploy (p. 627).

Once your environment is ready and green, you are ready to create a configuration file and update your application.

Step 2: Update the Application

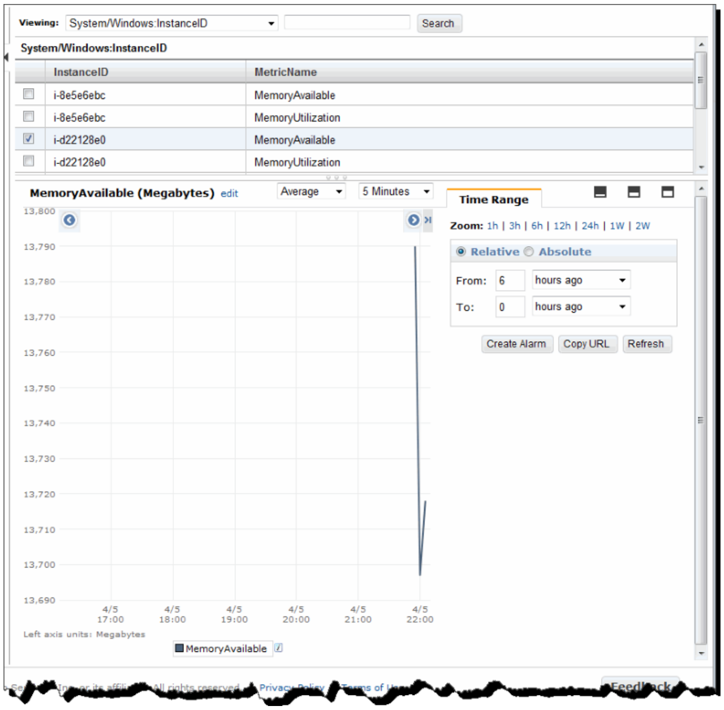
After you have deployed a sample application, you can update your Elastic Beanstalk environment to push your custom metrics to Amazon CloudWatch. In this step, we create a configuration file, and then add it to our project.

To update the Elastic Beanstalk environment with Amazon CloudWatch metrics

1. Create an `.ebextensions` directory in your project directory.
2. Create a file with the `.config` extension (e.g., cw.config) and place it in the `.ebextensions` directory. For more information about configuration files, see Customizing Software on Windows Servers (p. 160). The following snippet installs the Amazon CloudWatch monitoring scripts on to the EC2 instance, creates a file for the AWS security credentials and a batch file. The batch file contains the command that runs the monitoring script to report memory utilization and memory available to Amazon CloudWatch. The container command creates a scheduled task to run the batch file every 5 minutes.

```
sources:
  "c:/scripts": "https://s3.amazonaws.com/ec2-downloads-windows/cloudwatch-
samples/AmazonCloudWatchMonitoringWindows.zip"
files:
  "c:/scripts/monitoring_creds.conf":
    content: |
      AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
      AWSSecretKey=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  "c:/scripts/metrics_job.bat":
    content: |
      chdir "c:/scripts"
      powershell.exe -ExecutionPolicy Unrestricted .\AmazonCloudWatchMonit
oringWindows\mon-put-metrics-mem.ps1 -aws_credential_file monitor
ing_creds.conf -mem_util -mem_avail > metrics.log 2>&1
container_commands:
  01-schtask:
    command: schtasks /create /sc minute /mo 5 /f /ru System /tn "Put Metrics"
 /tr "\"C:\scripts\metrics_job.bat\""
```

3. Redeploy your application to Elastic Beanstalk. Using the AWS Toolkit for Visual Studio, you can right-click on your project, and select **Republih to Environment '*`your environment name`*'**.
4. When your environment is green and ready, check the Amazon CloudWatch console to view your metrics. Custom metrics will have the prefix **System/Windows** in the **Viewing** list. You should see something similar to the following.



Viewing: System/Windows:InstanceID

System/Windows:InstanceID

	InstanceID	MetricName
□	i-8e5e6ebc	MemoryAvailable
□	i-8e5e6ebc	MemoryUtilization
☑	i-d22128e0	MemoryAvailable
□	i-d22128e0	MemoryUtilization

MemoryAvailable (Megabytes)

Average · 5 Minutes

Time Range

Zoom: 1h | 3h | 6h | 12h | 24h | 1W | 2W

● Relative ○ Absolute

From: 6 hours ago

To: 0 hours ago

Updating Elastic Beanstalk Environments with Rolling Updates

Many configuration changes can be applied to a running environment without replacing existing instances. Setting a health check URL (p. 190), for example, triggers an environment update but does not cause any downtime because the instances running your application continue serving requests while the update is propagated.

Configuration changes that change the launch configuration (p. 105) or VPC settings (p. 114), on the other hand, require that all instances in your environment be terminated and replaced by new instances. This happens, for example, when you change the instance type or SSH key setting for your environment. To prevent downtime during this process, Elastic Beanstalk applies these configuration changes in batches, keeping a minimum number of instances up and serving traffic at all times.

Rolling configuration update batches can be processed periodically, with a delay between the start of each batch, or based on health. With health based rolling updates, Elastic Beanstalk waits until instances in a batch are healthy before moving on to the next batch.

The health of an instance is determined by the health reporting system, which can be basic (p. 247) (uses ELB health check) or enhanced (p. 250) (uses ELB health check and additional application and resource health checks). With basic health, a batch is considered healthy as soon as all instances in it pass ELB health checks and start serving traffic.

With enhanced health reporting, all of the instances in a batch must be reported healthy (OK status) across multiple consecutive checks before Elastic Beanstalk will move on to the next batch. In a web server environment with enhanced health, all instances must pass 12 health checks over the course of two minutes (18 checks over three minutes for worker environments). If any instance fails one health check, the count resets.

If a batch does not become healthy within the rolling update timeout (default is 30 minutes), the update is cancelled. Rolling update timeout is a configuration option (p. 103) that is available in the `aws:autoscaling:updatepolicy:rollingupdate (p. 111)` namespace.

> **Note**
> If your deployments are failing due to failed health checks and you need to force a deployment, change the rolling update type to time-based (p. 171) to ignore health check until you are able to get health checks to pass normally.

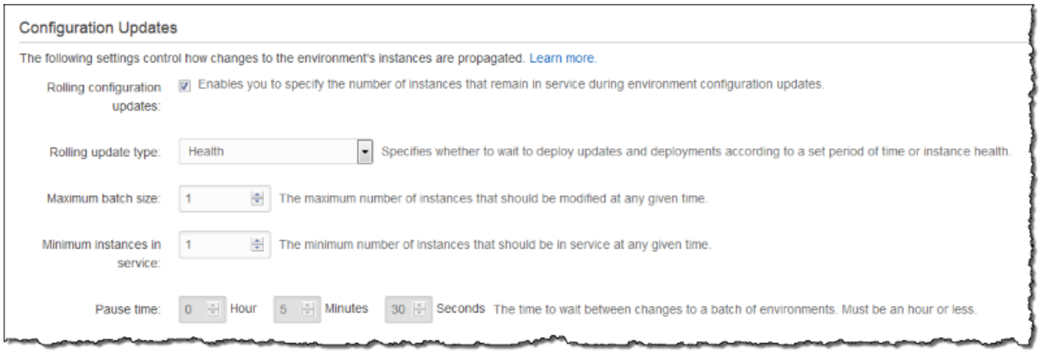
Rolling Updates vs Rolling Deployments

Application deployments can typically be performed without replacing instances in your environment. The exception to this is if your application source bundle contains configuration files in an `.ebextensions` folder that affect the launch configuration or VPC settings. Changes to these settings can trigger a full environment update (rolling or not) prior to the application version deployment.

Application version deployments can also be performed in batches to avoid downtime. Alternatively, you can deploy the new version of your application to a different environment and then perform a CNAME swap to start directing traffic to the updated environment. See Deploying Application Versions in Batches (Rolling Deployments) (p. 174) and Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83) for information about these features.

AWS Management Console

You can enable and configure rolling updates by editing **Updates and Deployments** on the environment's **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).



Configuration Updates

The following settings control how changes to the environment's instances are propagated. Learn more.

Rolling configuration updates: ☑ Enables you to specify the number of instances that remain in service during environment configuration updates.

Rolling update type: Health ▾ Specifies whether to wait to deploy updates and deployments according to a set period of time or instance health.

Maximum batch size: 1 The maximum number of instances that should be modified at any given time.

Minimum instances in service: 1 The minimum number of instances that should be in service at any given time.

Pause time: 0 Hour 5 Minutes 30 Seconds The time to wait between changes to a batch of environments. Must be an hour or less.

You must enable rolling updates before you can configure any of the parameters.

> **Note**
> Enabling rolling updates or changing its settings has no effect unless your IAM permissions are configured appropriately. To configure rolling updates, you must configure your IAM policy to allow you to perform any action on any autoscaling group in the AWS account by including a statement with `autoscaling:*`. For more information about how Elastic Beanstalk uses IAM policies, see Creating Policies to Control Access to Specific Elastic Beanstalk Resources (p. 329).

The **Configuration Updates** section of the **Updates and Deployments** page has the following options for rolling updates triggered by environment configuration changes:

Rolling Update Type

Environment configuration changes in load-balancing, autoscaling environments trigger updates to the instances in the environment. In those situations, you can specify what criteria to use to determine when to apply rolling updates to a batch of instances. The **Rolling update type** setting lets you specify whether to wait a specific amount of time or wait for instances to indicate that they are healthy before continuing to complete the update process on the next batch of instances.

It can be difficult to gauge the amount of time to wait between consecutive updates. You might choose a conservative pause time that results in an unnecessarily long total period of time that rolling updates are applied. With health-based rolling updates, you can potentially decrease the total amount of time to complete environment configuration updates to all instances. By basing the rolling update process on the results of an Elastic Load Balancing health check, you also have the assurance that instances within a batch are running and available after receiving an update.

In contrast, time-based rolling updates proceed without regard to the health of the instances in a particular batch. This can result in an environment with instances that have updated environment configuration settings, but are unavailable to receive traffic. For more information about Elastic Load Balancing health checks, see Health Checks (p. 190).

If you choose **Health** as the rolling update type, by default, Elastic Beanstalk waits 30 minutes for all instances in a batch to send events that indicate they are healthy before it cancels the update process. You can use the API or CLI to change the timeout setting to between 15 minutes and one hour. If the rolling update process fails, Elastic Beanstalk rolls back to the environment configuration settings in use prior to the rolling update attempt. If the attempt to roll back fails (for example, if you receive messages indicating that "resources failed to update" or Elastic Beanstalk cannot detect the health status of the instances in question), then you must terminate the environment and then launch a new environment. You can use a saved configuration to launch a new environment.

Batch Settings

- **Maximum batch size** – Specify the number of instances to terminate at any given time. Instances must be terminated before they can be updated or replaced during the rolling update process. By default, this value is one-third of the minimum size of the autoscaling group, rounded to the next highest integer. You can override this with a value between `1` and `10000`.
- **Minimum instances in service** – Indicate the minimum number of instances to keep running while other instances are being updated. The default value is either the minimum size of the autoscaling group or one less than the maximum size of the autoscaling group, whichever number is lower. For example, if the minimum size is 1 and the maximum size is 3, then the default is 1. However, if the minimum size is 4 and the maximum size is 4, then the default is 3. You can specify a number between `0` and `9999`.
- **Pause time** – Specify the amount of time the AWS CloudFormation service waits after it has completed updates to one batch of instances before it continues on to the next batch. The pause time accounts for the fact that instances are not immediately available after they start running. The environment's instance type and container type determine the default pause time, but you can override the recommended value. A valid pause time can range from 0 seconds to 1 hour.

  > **Note**
  > Elastic Beanstalk does not display the **Pause time** option if you have a load-balancing, autoscaling web server environment and choose to use health-based rolling updates. Rather than wait for a specific period of time to elapse, the AWS CloudFormation service continues on to the next batch of instances when it receives reports that the instances are running and available.

Default Pause Time Values

The **Pause time** boxes let you specify how long Elastic Beanstalk waits after it has completed updates to one batch of instances before it continues on to the next batch. Unless you have a specific need, we recommend that you use the default values provided by Elastic Beanstalk for each instance type and container combination. To use Elastic Beanstalk default values for pause time, you can either omit the parameter, leave it blank, or specify the option value as "null" when you configure rolling updates. If you choose to specify a custom pause time, you must follow the ISO8601 duration format in the form: `PT#H#M#S`, where each # is the number of hours, minutes, and/or seconds, respectively.

Command Line Interface

To edit rolling updates using the AWS CLI

- Change how updates are applied to your environment.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt (Example 1)

```
[
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "MaxBatchSize",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "MinInstancesInService",
        "Value": "2"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "PauseTime",
        "Value": "PT5M30S"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateEnabled",
        "Value": "true"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateType",
        "Value": "Time"
    }
]
```

options.txt (Example 2)

```
[
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "MaxBatchSize",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "MinInstancesInService",
        "Value": "2"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateEnabled",
        "Value": "true"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateType",
        "Value": "Health"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "Timeout",
        "Value": "PT45M"
    }
]
```

API

To edit rolling updates using the API

- Call `UpdateEnvironment` with the following parameters:
  - *EnvironmentName* = `SampleAppEnv`
  - *OptionSettings.member.1.Namespace* = `aws:autoscaling:updatepolicy:rollingupdate`
  - *OptionSettings.member.1.OptionName* = `MaxBatchSize`
  - *OptionSettings.member.1.Value* = `5`
  - *OptionSettings.member.2.Namespace* = `aws:autoscaling:updatepolicy:rollingupdate`
  - *OptionSettings.member.2.OptionName* = `MinInstancesInService`
  - *OptionSettings.member.2.Value* = `2`
  - *OptionSettings.member.3.Namespace* = `aws:autoscaling:updatepolicy:rollingupdate`
  - *OptionSettings.member.3.OptionName* = `PauseTime`
  - *OptionSettings.member.3.Value* = `PT5M30S`
  - *OptionSettings.member.4.Namespace* = `aws:autoscaling:updatepolicy:rollingupdate`
  - *OptionSettings.member.4.OptionName* = `RollingUpdateEnabled`
  - *OptionSettings.member.4.Value* = `true`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Aupdatepolicy%3Arol
lingupdate
&OptionSettings.member.1.OptionName=MaxBatchSize
&OptionSettings.member.1.Value=5
&OptionSettings.member.2.Namespace=aws%3Aautoscaling%3Aupdatepolicy%3Arol
lingupdate
&OptionSettings.member.2.OptionName=MinInstancesInService
&OptionSettings.member.2.Value=2
&OptionSettings.member.3.Namespace=aws%3Aautoscaling%3Aupdatepolicy%3Arol
lingupdate
&OptionSettings.member.3.OptionName=PauseTime
&OptionSettings.member.3.Value=PT5M30S
&OptionSettings.member.4.Namespace=aws%3Aautoscaling%3Aupdatepolicy%3Arol
lingupdate
&OptionSettings.member.4.OptionName=RollingUpdateEnabled
&OptionSettings.member.4.Value=true
&Operation=UpdateEnvironment
&AuthParams
```

Deploying Application Versions in Batches (Rolling Deployments)

Deploying a new version of your application to an environment is typically a fairly quick process. The new source bundle is deployed to an instance and extracted, and then the web container or application server picks up the new version and restarts if necessary.

If your application takes a long time to start up, however, there can be significant down time when a new version is deployed. Elastic Beanstalk has two features to avoid downtime during application deployment. One is CNAME Swap (p. 83), where you deploy your new version to a second environment and change the DNS settings to start directing traffic to the updated environment once the deployment is complete.

The second feature is a rolling deployment (not to be confused with a rolling configuration update (p. 169)). With rolling deployments, Elastic Beanstalk splits the environments into batches and deploys to one batch at a time, leaving the rest of the instances in the environment running the old application version. In the middle of a rolling deployment, some instances can be serving requests with the old version, while others from an already completed batch could be serving other requests with the new version at the same time.

To process a batch, Elastic Beanstalk detaches all instances in the batch from the load balancer, deploys the new application version, and then re-attaches them. If you have connection draining (p. 189) enabled, Elastic Beanstalk drains existing connections from the Amazon EC2 instances in each batch before beginning the deployment.

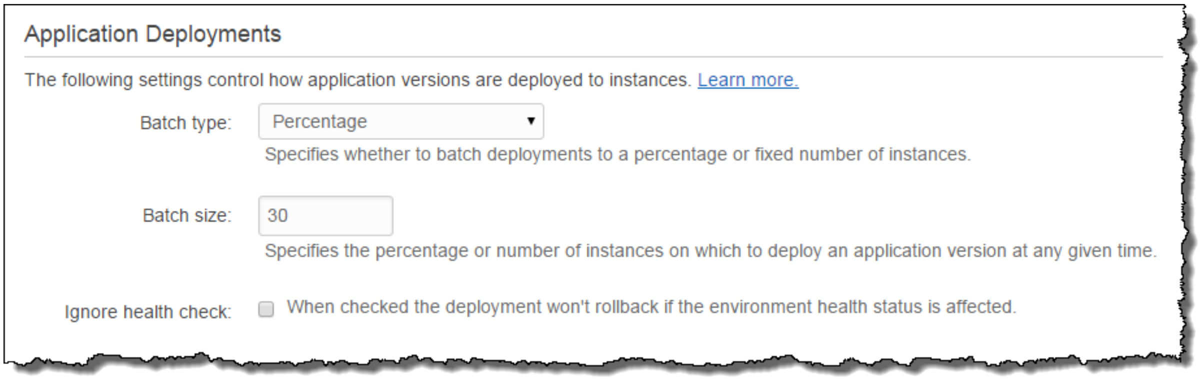
Once the instances in a batch are re-attached to the load balancer, Elastic Load Balancing waits until they pass a minimum number of ELB health checks (the healthy check count threshold) and then starts routing traffic to them. If no health check URL (p. 190) is configured, this can happen very quickly, because an instance will pass the health check as soon as it can accept a TCP connection. If a health check URL is configured, no traffic will be routed to the updated instances until they return a 200 OK status code in response to an HTTP GET request to the health check URL.

Elastic Beanstalk waits until instances in a batch are healthy before moving on to the next batch. With basic health reporting (p. 247), this is dependent on the ELB health check. Once all instances in the batch pass enough health checks to be considered healthy by Elastic Load Balancing, the batch is complete. If enhanced health reporting (p. 250) is enabled, several other factors are considered, including the result of incoming requests. With enhanced health reporting all instances must pass 12 consecutive health checks with OK status (p. 260) over 2 minutes for web server environments (18 over three minutes for worker environments).

> **Note**
> If your deployments are failing due to health checks and you need to force an update to go through regardless, set the *Ignore Health Check* option.

AWS Management Console

You can enable and configure batched application version deployments by editing **Updates and Deployments** on the environment's **Configuration** page in the Elastic Beanstalk console. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).



Application Deployments

The following settings control how application versions are deployed to instances. Learn more.

Batch type: Percentage ▾
Specifies whether to batch deployments to a percentage or fixed number of instances.

Batch size: 30
Specifies the percentage or number of instances on which to deploy an application version at any given time.

Ignore health check: □ When checked the deployment won't rollback if the environment health status is affected.

The **Application Deployments** section of the **Updates and Deployments** page has the following options for configuring batched application version deployments:

- **Batch type** – Specify whether you want a batch of Amazon EC2 instances to consist of a percentage of the total number of instances in the autoscaling group or a fixed number.
- **Batch size** – The number or percent of instances to deploy in each batch, up to 100 percent or the maximum instance count in your environment's Auto Scaling configuration.
- **Ignore health check** – Prevent a deployment from rolling back when a batch fails to become healthy within the rolling upate timeout.

Command Line Interface (CLI)

To configure batched application version deployments using the AWS CLI

- Change how application versions are applied to your environment.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:command",
        "OptionName": "BatchSize",
        "Value": "50"
    },
    {
        "Namespace": "aws:elasticbeanstalk:command",
        "OptionName": "BatchType",
        "Value": "Percentage"
    },
    {
        "Namespace": "aws:elasticbeanstalk:command",
        "OptionName": "IgnoreHealthCheck",
        "Value": "true"
    }
]
```

API

To edit rolling updates using the API

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`VersionLabel`* = `Version2`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.1.OptionName`* = `BatchSize`
  - *`OptionSettings.member.1.Value`* = `50`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.2.OptionName`* = `BatchSizeType`
  - *`OptionSettings.member.2.Value`* = `Percentage`
  - *`OptionSettings.member.3.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.3.OptionName`* = `IgnoreHealthCheck`
  - *`OptionSettings.member.3.Value`* = `true`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&VersionLabel=Version2
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.1.OptionName=BatchSize
&OptionSettings.member.1.Value=50
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.2.OptionName=BatchSizeType
&OptionSettings.member.2.Value=Percentage
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.3.OptionName=IgnoreHealthCheck
&OptionSettings.member.3.Value=true
&Operation=UpdateEnvironment
&AuthParams
```

Configuring Auto Scaling with Elastic Beanstalk

Auto Scaling is a web service designed to automatically launch or terminate Amazon EC2 instances in an autoscaling group based on parameters that you define. Elastic Beanstalk supports Auto Scaling for load-balancing, autoscaling environments. Automatically increasing or decreasing the number of Amazon EC2 instances helps you seamlessly deal with traffic changes to your application. The *triggers* that you define for an autoscaling group instruct Elastic Beanstalk to scale computing resources in response to metrics such as bandwidth usage or CPU utilization. You can also schedule scaling actions to occur at specific times. Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

> **Note**
> Autoscaling settings can impact rolling updates. If you configured rolling updates, a rolling update cannot happen if the minimum number of instances in service for rolling updates is larger than the maximum size of the autoscaling group. You must change the value of either the **Minimum instances in service** setting for rolling updates or the **Maximum instance count** setting for the autoscaling group.

For more information about Auto Scaling, go to the Auto Scaling documentation.

AWS Management Console

You can configure how Auto Scaling works by editing **Scaling** on the environment's **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).

The following section discusses how to configure Auto Scaling parameters for your application.

Launch Configuration

You can edit the launch configuration in the **Auto Scaling** section of the page to control how your Elastic Beanstalk application provisions Auto Scaling resources.

The **Minimum instance count** and **Maximum instance count** boxes let you specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.

> **Note**
> To maintain a fixed number of Amazon EC2 instances, set the **Minimum instance count** and **Maximum instance count** boxes to the same value.

The **Availability Zones** box lets you specify the number of Availability Zones in which you want Elastic Beanstalk to launch instances. We recommend that you choose multiple Availability Zones so that instances can be launched in another Availability Zone if one zone becomes unavailable. For example, if you have a minimum of 3 instances, then you should choose 3 Availability Zones.

The **Custom Availability Zones** box lets you specify which Availability Zones you want Elastic Beanstalk to launch instance(s) in across a region. If you do not select any custom Availability Zones, then Elastic Beanstalk will choose the Availability Zones for you. The number of Availability Zones must be less than or equal to the number of custom Availability Zones you select. For example, if you select **Any 2**, then you must select at least two custom Availability Zones.

> **Note**
> Here are some things to keep in mind when working with Availability Zones:
>
> - It is important to launch instances in more than one Availability Zone in order to build fault-tolerant applications. If one Availability Zone goes down, your instances will still be running in another Availability Zones.
> - If you purchased Reserved Instances, you need to specify the same Availability Zone(s) that you specified when you purchased your Reserved Instances. Reserved Instances let you make a low, one-time, upfront payment for an instance, reserve it for a one- or three-year term, and pay a significantly lower hourly rate for that instance. For more information about Reserved Instances, see Reserved Instances in the *Amazon EC2 User Guide for Linux Instances*.

The **Scaling cooldown (seconds)** box specifies how long Auto Scaling will wait before resuming any scaling activities after it launches an instance so that the new instance has time to begin handling traffic. If, by the time the cooldown period elapses, your environment has the appropriate level of resources to keep the CloudWatch alarm from continuing to fire, then no new instances will be launched.

Triggers

You can edit the **Scaling Trigger** section of the page to set metrics-based parameters for scaling actions for your Elastic Beanstalk application. A *trigger* is an Auto Scaling mechanism that you define to tell the system when to increase (*scale out*) the number of instances and when to decrease (*scale in*) the number of instances. You can configure triggers to *fire* on Amazon EC2 or Elastic Load Balancing metrics published to Amazon CloudWatch, such as an Amazon EC2 instance's CPU utilization, and determine whether the conditions you specified have been met. When the upper or lower thresholds of the conditions for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *scaling activity*. For more information about Amazon EC2 metrics, see Amazon Elastic Compute Cloud Dimensions and Metrics in the *Amazon CloudWatch Developer Guide*. For more information about Elastic Load Balancing metrics, see Monitor Your Load Balancer Using Amazon CloudWatch in the *Elastic Load Balancing Developer Guide*.

Auto Scaling triggers work by watching a specific Amazon CloudWatch metric for an instance. Triggers include CPU utilization, network traffic, disk activity, and instance health. Use the **Trigger measurement** setting to specify a metric for your trigger.

The following list describes the trigger parameters you can configure using the AWS Management Console.

- Use **Trigger statistic** to specify which statistic the trigger should use—**`Minimum`**, **`Maximum`**, **`Sum`**, or **`Average`**.
- Use **Unit of measurement** to specify the unit for the trigger measurement.
- For **Measurement period**, specify how frequently Amazon CloudWatch measures the metrics for your trigger. **Breach duration** is the amount of time a metric can extend beyond its defined limit (as specified for **Upper threshold** and **Lower threshold**) before the trigger fires.
- **Upper breach scale increment** and **Lower breach scale increment** specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

Time-based Scaling

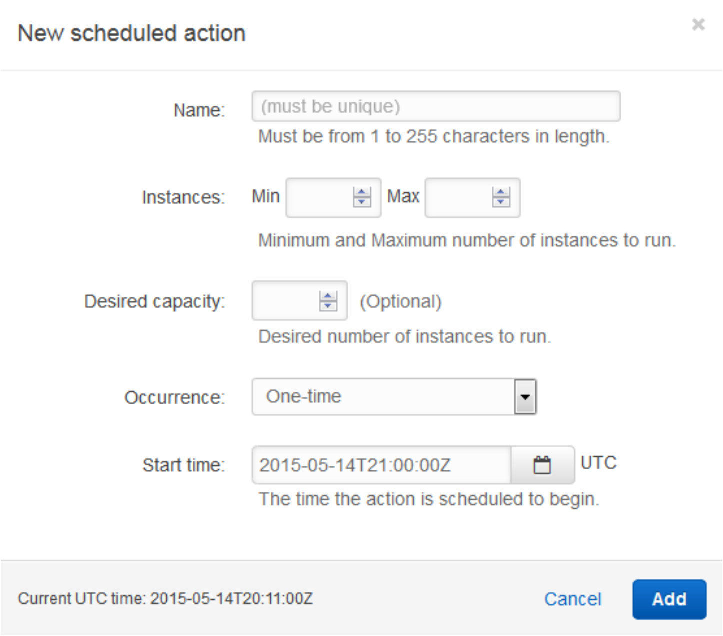
Time-based scaling enables you to plan scaling actions that launch or terminate Amazon EC2 instances in each autoscaling group. You can schedule scaling actions, as needed. To see the list of an environment's scheduled scaling actions, including expired ones, see the **Time-based Scaling** section of the **Scaling** page in the AWS Management Console. An environment can have up to 120 scheduled scaling actions, not including expired actions (actions that ended in the past).

If your Elastic Beanstalk application has predictable needs for scaling out or scaling in at specific times, you can schedule scaling actions on a recurring basis. You can also schedule a scaling action for a planned one-time future event. Instead of creating an entirely new scheduled action, you can reuse an expired scheduled action by changing the start time to a time and date in the future. (Elastic Beanstalk keeps a history of the most recent 150 expired scheduled actions.)

The table in the **Time-based Scaling** section displays the following information.

- The **Name** column shows the name that you assigned the scheduled action when you created it. You cannot change the name of a scaling action after you create it.
- The **Limits** column displays the minimum, maximum, and desired number of instances that you specified for the autoscaling group when you created or most recently changed the scheduled action.
- In the **Next occurrence** column, you can see the next time the scheduled action will take effect. For a recurrent scaling action, the next occurrence is based on the recurrence that you specified with a CRON expression.

In the **Time-based Scaling** section of the page, if you click **Add scheduled action**, the **New scheduled action** window displays the following options:



New scheduled action

Name: (must be unique)
Must be from 1 to 255 characters in length.

Instances: Min Max
Minimum and Maximum number of instances to run.

Desired capacity: (Optional)
Desired number of instances to run.

Occurrence: One-time

Start time: 2015-05-14T21:00:00Z UTC
The time the action is scheduled to begin.

Current UTC time: 2015-05-14T20:11:00Z

- Use **Name** to specify a name for the scheduled scaling action that differentiates it from other scaling actions and that you can use to find it later. You must use alphanumeric characters with no spaces. The name has a limit of 255 characters.
- For **Instances**, indicate the **Min** (minimum) and **Max** (maximum) number of instances for the autoscaling group.
- The **Desired capacity** box lets you specify the number of instances that you want running in the autoscaling group. The desired capacity must be less than or equal to the value you defined as the maximum number of instances and larger than or equal to the value you defined as the minimum.
- For **Occurrence**, choose whether the scaling action should occur only once or on a recurring basis.
  - If you choose **One-time**, configure the following settings:
    - **Start time** – Specify a date and time in the future (in the UTC/GMT time zone) when you want the scheduled action to take effect. The date and time combination must be unique across all scheduled actions. Use ISO 8601 time format to specify the start time. For example, 2015-04-28T04:07:02Z. For more information about ISO 8601 time format, go to Date and Time Formats.
  - If you choose **Recurrent**, configure the following settings:
    - **Recurrence** – Use a CRON expression to specify the frequency with which you want the scheduled action to occur. For example, if you want to schedule a recurrent action that take effect every Tuesday at 6:30 PM UTC time, you can use the CRON expression “30 6 * * 2”. For more information about CRON syntax, see Cron.
    - **Start time** (optional) – Specify a date and time in the future (in the UTC/GMT time zone) when you want to start the scheduled action. The date and time combination must be unique across all scheduled actions. Use ISO 8601 time format to specify the start time. For example, 2015-04-28T04:07:02Z. For more information about ISO 8601 time format, go to Date and Time Formats. (If you don't specify a start time, the recurrence is immediately valid after the environment is updated with the new setting.)
    - **End time** (optional) – Specify a date and time in the future (in the UTC/GMT time zone) when you want to stop the scheduled action. The date and time combination must be unique across all scheduled actions. (If you don’t specify an end time, the scaling action recurs until you configure it with an end date and time.)

You can’t suspend a scheduled action using the Elastic Beanstalk management console, but you can configure option settings with the EB CLI to do so. When you suspend a scheduled scaling action, Elastic

Beanstalk ignores the scheduled action and does nothing. Suspending an action lets you stop the scheduled action without deleting it so that you can resume the action later, if necessary.

Examples of Time-based Scaling Option Settings Files

This section provides examples of option settings files that will configure scheduled scaling actions. You can include this file in your application source bundle. The contents of an option settings file must conform to JSON formatting standards. For more information, go to http://www.json.org.

The following JSON snippet instructs Elastic Beanstalk to scale out from five instances to 10 instances at 2015-12-12T00:00:00Z.

```
[
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleUpSpecificTime",
        "OptionName": "MinSize",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleUpSpecificTime",
        "OptionName": "MaxSize",
        "Value": "10"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleUpSpecificTime",
        "OptionName": "DesiredCapacity",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleUpSpecificTime",
        "OptionName": "StartTime",
        "Value": "2015-12-12T00:00:00Z"
    }
]
```

The following JSON snippet instructs Elastic Beanstalk to scale in at 2015-12-12T07:00:00Z.

```
[
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleDownSpecificTime",
        "OptionName": "MinSize",
        "Value": "1"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleDownSpecificTime",
        "OptionName": "MaxSize",
        "Value": "1"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleDownSpecificTime",
        "OptionName": "DesiredCapacity",
        "Value": "4"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledScaleDownSpecificTime",
        "OptionName": "StartTime",
        "Value": "2015-12-12T07:00:00Z"
    }
]
```

The following JSON snippet instructs Elastic Beanstalk to scale out every day at 9AM. The action is scheduled to begin May 14, 2015 and end January 12, 2016.

```
[
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "MinSize",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "MaxSize",
        "Value": "10"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "DesiredCapacity",
        "Value": "5"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "StartTime",
        "Value": "2015-05-14T07:00:00Z"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "EndTime",
        "Value": "2016-01-12T07:00:00Z"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledPeriodicScaleup",
        "OptionName": "Recurrence",
        "Value": "0 9 * * *"
    }
]
```

The following JSON snippet instructs Elastic Beanstalk to scale in every Friday at 6PM. If you know that your application doesn’t receive as much traffic over the weekend, you can create a similar scheduled action.

```
[
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "MinSize",
        "Value": "1"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "MaxSize",
        "Value": "4"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "DesiredCapacity",
        "Value": "1"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "StartTime",
        "Value": "2015-12-12T07:00:00Z"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "EndTime",
        "Value": "2016-01-12T07:00:00Z"
    },
    {
        "Namespace": "aws:autoscaling:scheduledaction",
        "ResourceName": "ScheduledWeekendScaleDown",
        "OptionName": "Recurrence",
        "Value": "0 18 * * 5"
    }
]
```

Command Line Interface (CLI)

You can use EB CLI to edit environment settings for autoscaling.

To edit an application's environment settings using EB CLI

1. Update an application's environment settings.

   ```
   $ eb config my-env
   ```

2. Add the following to the option settings file that opens in your text editor.

   ```
     AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
       LowerBreachScaleIncrement: '-1'
     AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
       UpperBreachScaleIncrement: '1'
   ```

```
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
  AWSEBCloudwatchAlarmLow.aws:autoscaling:trigger:
    BreachDuration: '5'
    EvaluationPeriods: '1'
    LowerThreshold: '2000000'
    MeasureName: NetworkOut
    Period: '5'
    Statistic: Average
    Unit: Bytes
  aws:autoscaling:asg:
    Availability Zones: Any
    Cooldown: '360'
    Custom Availability Zones: null
    MaxSize: '4'
    MinSize: '1'
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: null
    EC2KeyName: null
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    ImageId: ami-13de5b0e
    InstanceType: t1.micro
    MonitoringInterval: 5 minute
    RootVolumeIOPS: null
    RootVolumeSize: null
    RootVolumeType: null
    SSHSourceRestriction: tcp,22,22,0.0.0.0/0
    SecurityGroups: awseb-e-um3yfrzq22-stack-AWSEBSecurityGroup-10MV688E4994W
```

You can also specify custom Availability Zones using EB CLI.

To update an application's environment settings with custom Availability Zones using EB CLI

1. Update an application's environment settings with custom Availability Zones for autoscaling.

```
$ eb config
```

2. Add the following to the option settings file that opens in your text editor.

```
  aws:autoscaling:asg:
    Availability Zones: Any
    Cooldown: '360'
    Custom Availability Zones: 'us-west-2a,us-west-2b'
    MaxSize: '4'
    MinSize: '1'
```

You can use EB CLI to create a scheduled action to scale out or scale in the number of instances in each autoscaling group in your environment.

To create a new scheduled scaling action using EB CLI

1. Run `eb config`

```
$ eb config
```

2. Add the following to the option settings file that opens in your text editor to create a one-time scheduled scaling action.

   Note
   In the following examples, **`ScheduledScaleUpSpecificTime`** and **`ScheduledScaleUpRecurring`** are the names you want to give each respective scheduled action. Settings are grouped according to the scheduled scaling action name.

```
ScheduledScaleUpSpecificTime.aws:autoscaling:scheduledaction:
  DesiredCapacity: '5'
  EndTime: null
  MaxSize: '10'
  MinSize: '5'
  Recurrence: null
  StartTime: '2015-12-12T00:00:00Z'
  Suspend: 'false'
```

   Add the following to the option settings file that opens in your text editor to create a recurring scheduled scaling action.

```
ScheduledScaleUpRecurring.aws:autoscaling:scheduledaction:
  DesiredCapacity: '5'
  EndTime: '2016-01-12T07:00:00Z'
  MaxSize: '10'
  MinSize: '5'
  Recurrence: '0 9 * * *'
  StartTime: '2015-12-12T00:00:00Z'
  Suspend: 'false'
```

Auto Scaling Health Check Setting

By default, the Auto Scaling created for your environment uses Amazon EC2 status checks. If an instance in your environment fails an EC2 status check, it is taken down and replaced by Auto Scaling.

EC2 status checks only cover an instance's health, not the health of your application, server, or any Docker containers running on the instance. If your application crashes, but the instance that it runs on is still healthy, it may be kicked out of the load balancer, but it won't be replaced automatically by Auto Scaling.

Note
Configuring a health check URL (p. 190) in the Load Balancing configuration section of the Elastic Beanstalk console does not affect the health check behavior of an environment's Auto Scaling group.

You may want instances in your environment to stay up when the application crashes for troubleshooting and recovery purposes. If Auto Scaling replaced the instance as soon as the application crashed, you may not realize that anything went wrong, even if it crashed quickly after starting up.

If you would like Auto Scaling to restart instances whose application has stopped responding, you can configure the Auto Scaling group to use Elastic Load Balancing health checks with an `.ebextensions` configuration file.

> **Note**
> Enabling this option can cause your environment to continually terminate and replace instances if the Elastic Load Balancing health check fails soon after an instance starts. If this happens, change the health check type back to EC2 in the Auto Scaling management console, or reverse the configuration change and redeploy your environment.

To set the Auto Scaling health check to ELB during environment launch

1. Create a `.ebextensions` directory in your project folder.
2. Create a config file in the `.ebextensions` folder with the following content:

   `.ebextensions/autoscaling.config`

   ```
   Resources:
     AWSEBAutoScalingGroup:
       Type: "AWS::AutoScaling::AutoScalingGroup"
       Properties:
         HealthCheckType: ELB
         HealthCheckGracePeriod: 300
   ```

   The grace period setting sets the number of seconds that an instance can fail the health check without being terminated and replaced. Instances may recover even after being kicked out of the load balancer, so give the instance an amount of time that is appropriate for your application.
3. Deploy or update your environment.

By default, the Elastic Load Balancing health check is configured to attempt a TCP connection to your instance over port 80. This confirms that the web server running on the instance is accepting connections, but you may want to customize the check to make sure that your application is in a good state, not just the web server.

You can tell Elastic Load Balancing to use a different port or to make an HTTP GET request to a path that your application uses by changing the Elastic Load Balancing configuration for your environment. See Configuring Elastic Load Balancing with Elastic Beanstalk (p. 186) for details.

Configuring Elastic Load Balancing with Elastic Beanstalk

Elastic Load Balancing is an Amazon web service that helps you improve the availability and scalability of your application. This service makes it easy for you to distribute application loads among Amazon EC2 instances. Elastic Load Balancing increases availability through redundancy and supports traffic growth for your application.

Elastic Load Balancing lets you automatically distribute and balance the incoming application traffic among all the instances you are running. The service also makes it easy to add and remove instances when you need to increase or decrease the capacity of your application.

AWS Management Console

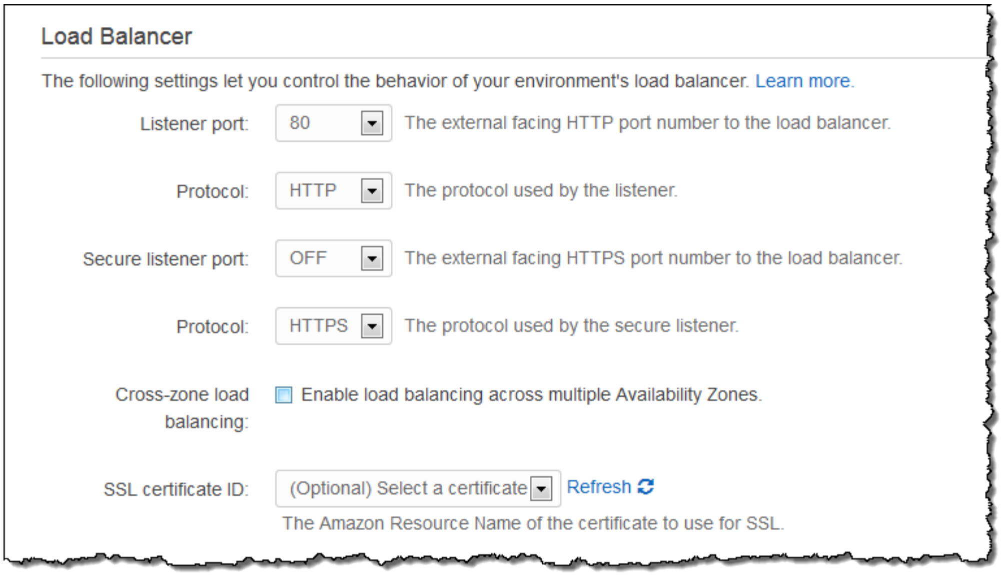
You can modify an environment's load balancing configuration by editing **Load Balancing** in the **Network Tier** section of the environment's **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).

In the AWS Management Console, Elastic Beanstalk groups the parameters that you can configure for Elastic Load Balancing into the following categories:

- **Load Balancer** – For information about these settings, see Ports and Cross-zone Load Balancing (p. 187).
- **Connection Draining** – For information about these settings, see Connection Draining (p. 189).
- **Sessions** – For information about these settings, see Sessions (p. 189).
- **EC2 Instance Health Check** – For information about these settings, see Health Checks (p. 190).

The following sections describe the Elastic Load Balancing settings you can configure for your application.

Ports and Cross-zone Load Balancing



Load Balancer

The following settings let you control the behavior of your environment's load balancer. Learn more.

Listener port: 80 ▾ The external facing HTTP port number to the load balancer.

Protocol: HTTP ▾ The protocol used by the listener.

Secure listener port: OFF ▾ The external facing HTTPS port number to the load balancer.

Protocol: HTTPS ▾ The protocol used by the secure listener.

Cross-zone load balancing: □ Enable load balancing across multiple Availability Zones.

SSL certificate ID: (Optional) Select a certificate ▾ Refresh
The Amazon Resource Name of the certificate to use for SSL.

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on the specified ports and route requests to the Amazon EC2 instances in your Elastic Beanstalk application. By default, the load balancer handles requests on port 80. At least one of the ports must be turned on.

Important
Note the following:

- If you are deploying an application using a legacy container type, make sure the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application. To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).
- If you are deploying an application using a nonlegacy container type, and you want to access your application directly on the EC2 instance using your web browser, modify your HTTP rule in your EC2 security group. For instructions, go to Amazon EC2 Security Groups (p. 201). For

a list of supported nonlegacy container types, see Why are some container types marked legacy? (p. 92).

Controlling the listener port

To turn off the listener port, you select **OFF** for **Listener Port**. To turn on the listener port, you select a port (for example, **80**).

Note
If you want to access your environment using a different port other than the default port 80 (e.g., port 8080), you can add a listener to the existing load balancer and configure the new listener to listen on that port. For example, using the Elastic Load Balancing API Tools, type the following command replacing <yourloadbalancername> with the name of your load balancer for Elastic Beanstalk.

```
elb-create-lb-listeners --lb <yourloadbalancername> --listener "pro
tocol=http, lb-port=8080, instance-port=80"
```

If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the listener protocol

If you turn on the listener port, you can specify the protocol to use. Select **HTTP** or **TCP** from the **Listener protocol** list.

Note
This option is available for nonlegacy containers only. For instructions on migrating from a legacy container, see Migrating Your Application from a Legacy Container Type (p. 91).

Controlling the secure listener port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances are done using plaintext. By default, the secure listener port is turned off.

To turn on the secure listener port

1. Create and upload a certificate and key to the AWS Access and Identity Management (IAM) service. The IAM service will store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate you've uploaded. For more information about creating and uploading certificates, see the Managing Server Certificates section of *Using AWS Identity and Access Management*.
2. Specify the secure listener port by selecting a port from the **Secure Listener Port** list.
3. In the **SSL Certificate ID** box, enter the Amazon Resources Name (ARN) of your SSL certificate (e.g., `arn:aws:iam::123456789012:server-certificate/abc/certs/build`). Use the SSL certificate that you created and uploaded in step 1. For information about viewing the certificate's ARN, see Creating, Uploading, and Deleting Server Certificates topic in the *Creating and Uploading Server Certificates* section of the *Using IAM* guide.

To turn off the secure listener port, select OFF from the **Secure Listener Port** drop-down list.

Controlling the secure protocol

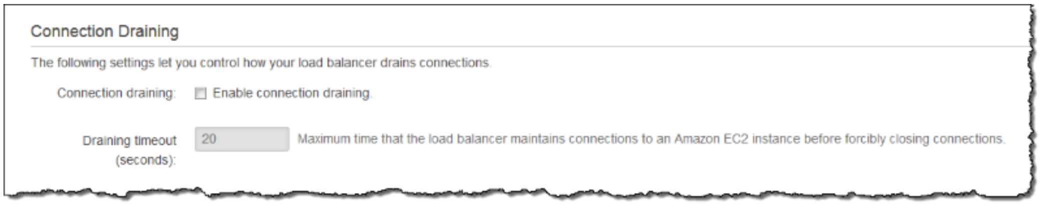
If you turn on the secure listener port, you can specify the protocol to use. Select **HTTPS** or **SSL** from the **Secure Listener Protocol** list.

> **Note**
> This option is available for nonlegacy containers only. For instructions on migrating from a legacy container, see Migrating Your Application from a Legacy Container Type (p. 91).

Enabling cross-zone load balancing

By default, the load balancer node routes traffic to Amazon EC2 instances within the same Availability Zone. When you enable cross-zone load balancing, traffic is routed evenly across all instances, regardless of which Availability Zone the instances are in. This ensures that all Availability Zones receive an equal amount of request traffic. Cross-zone load balancing reduces the need to maintain equivalent numbers of instances in each zone and improves the application's ability to fail over if you lose one or more instances.

Connection Draining



Connection Draining

The following settings let you control how your load balancer drains connections.

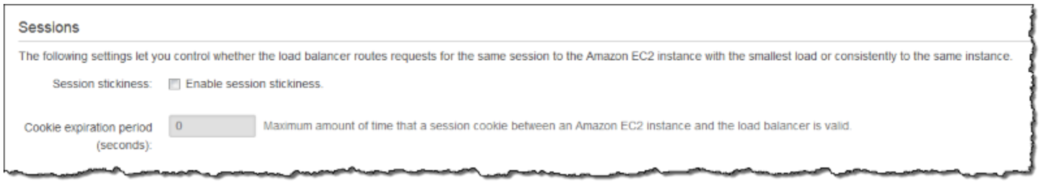
Connection draining: □ Enable connection draining

Draining timeout (seconds): 20 Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections.

You can enable connection draining if you want to keep existing load balancer connections open to unhealthy or deregistering instances to complete in-progress requests even as the load balancer stops sending new requests. Otherwise, when a load balancer detects an unhealthy or deregistering instance, it closes the connections without regard for requests that are in progress. When you enable connection draining, you can specify a maximum time for the load balancer to keep connections alive before forcibly closing connections and reporting the instance as deregistered. If you do not specify the maximum timeout period, by default, the load balancer will close connections to the deregistering instance after 20 seconds. You can specify a draining timeout value up to `3600`.

If your instances are part of an Auto Scaling group and if connection draining is enabled for your load balancer, Auto Scaling will wait for the requests that are in progress to be completed or for the maximum timeout to expire, whichever comes first, before terminating instances.

Sessions



Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance with the smallest load or consistently to the same instance.

Session stickiness: □ Enable session stickiness

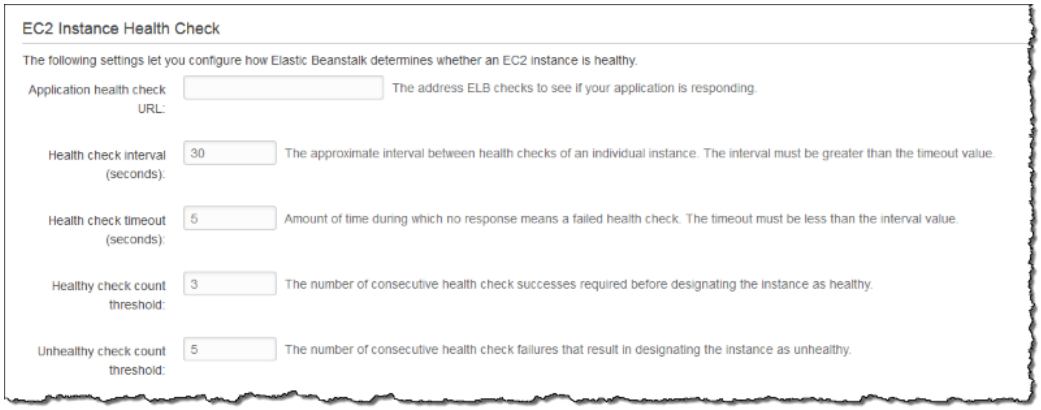
Cookie expiration period (seconds): 0 Maximum amount of time that a session cookie between an Amazon EC2 instance and the load balancer is valid.

By default a load balancer routes each request independently to the server instance with the smallest load. By comparison, enabling session stickiness binds a user's session to a specific server instance so that all requests coming from the user during the session will be sent to the same server instance.

Elastic Beanstalk uses load balancer-generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer–generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If there is no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie. You can configure the cookie expiration period as a value between **`0`** and **`1000000`** seconds.

For more information about Elastic Load Balancing, go to the Elastic Load Balancing Developer Guide.

Health Checks



EC2 Instance Health Check

The following settings let you configure how Elastic Beanstalk determines whether an EC2 instance is healthy.

Application health check URL:		The address ELB checks to see if your application is responding.
Health check interval (seconds):	30	The approximate interval between health checks of an individual instance. The interval must be greater than the timeout value.
Health check timeout (seconds):	5	Amount of time during which no response means a failed health check. The timeout must be less than the interval value.
Healthy check count threshold:	3	The number of consecutive health check successes required before designating the instance as healthy.
Unhealthy check count threshold:	5	The number of consecutive health check failures that result in designating the instance as unhealthy.

Elastic Load Balancing uses a *health check* to determine whether the Amazon EC2 instances running your application are healthy. The health check determines an instance's health status by probing a specified URL at a set interval; if the URL returns an error message, or fails to return within a specified timeout period, the health check fails. Elastic Beanstalk uses Elastic Load Balancing health checks to set the status of your application in the AWS Management Console.

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** configuration page.

The health check definition includes a URL to be queried for instance health. By default, Elastic Beanstalk uses TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override the default URL to match an existing resource in your application (e.g., **`/myapp/index.jsp`**) by entering it in the **Application health check URL** box. If you override the default URL, then Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).

> **Note**
> Configuring a health check URL does not affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check will not automatically be replaced by Auto Scaling unless you configure Auto Scaling to do so manually. See Auto Scaling Health Check Setting (p. 185) for details.

The following list describes the health check parameters you can set for your application.

- For **Health check interval (seconds)**, enter the number of seconds for Elastic Load Balancing to wait between each check of your application's Amazon EC2 instances.
- For **Health check timeout (seconds)**, specify the number of seconds Elastic Load Balancing will wait for a response before it considers the instance unresponsive.
- For **Healthy check count threshold** and **Unhealthy check count threshold**, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying 5 in the **Unhealthy Check Count Threshold** box means that

the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check failed.

Command Line Interface (CLI)

To edit an application's environment settings

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "LoadBalancerHTTPPort",
        "Value": "80"
    },
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "LoadBalancerPortProtocol",
        "Value": "HTTP"
    },
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "LoadBalancerHTTPSPort",
        "Value": "443"
    },
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "LoadBalancerSSLPortProtocol",
        "Value": "HTTPS"
    },
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "SSLCertificateId",
       "Value": "arn:aws:iam::123456789012:server-certificate/abc/certs/build"

    },
    {
        "Namespace": "aws:elasticbeanstalk:application",
        "OptionName": "Application Healthcheck URL",
        "Value": "/"
    },
    {
        "Namespace": "aws:elb:healthcheck",
        "OptionName": "Interval",
        "Value": "30"
    },
    {
        "Namespace": "aws:elb:healthcheck",
        "OptionName": "Timeout",
        "Value": "5"
    },
    {
        "Namespace": "aws:elb:healthcheck",
        "OptionName": "HealthyThreshold",
        "Value": "3"
    },
    {
        "Namespace": "aws:elb:healthcheck",
        "OptionName": "UnhealthyThreshold",
        "Value": "5"
    },
    {
        "Namespace": "aws:elb:policies",
        "OptionName": "Stickiness Policy",
        "Value": "false"
    },
    {
        "Namespace": "aws:elb:policies",
        "OptionName": "Stickiness Cookie Expiration",
        "Value": "0"
    },
    {
        "Namespace": "aws:elb:policies",
        "OptionName": "ConnectionDrainingEnabled",
        "Value": "true"
    },
    {
        "Namespace": "aws:elb:policies",
        "OptionName": "ConnectionDrainingTimeout",
        "Value": "60"
    },
    {
        "Namespace": "aws:elb:loadbalancer",
        "OptionName": "CrossZone",
        "Value": "true"
    }
]
```

API

To edit an application's environment settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elb:loadbalancer`
  - *`OptionSettings.member.1.OptionName`* = `LoadBalancerHTTPPort`
  - *`OptionSettings.member.1.Value`* = `80`
  - *`OptionSettings.member.2.Namespace`* = `aws:elb:loadbalancer`
  - *`OptionSettings.member.2.OptionName`* = `LoadBalancerHTTPSPort`
  - *`OptionSettings.member.2.Value`* = `443`

- *OptionSettings.member.3.Namespace* = aws:elb:loadbalancer
- *OptionSettings.member.3.OptionName* = SSLCertificateId
- *OptionSettings.member.3.Value* = arn:aws:iam::123456789012:server-certificate/abc/certs/build
- *OptionSettings.member.4.Namespace* = aws:elasticbeanstalk:application
- *OptionSettings.member.4.OptionName* = Application Healthcheck URL
- *OptionSettings.member.4.Value* = /
- *OptionSettings.member.5.Namespace* = aws:elb:healthcheck
- *OptionSettings.member.5.OptionName* = Interval
- *OptionSettings.member.5.Value* = 30
- *OptionSettings.member.6.Namespace* = aws:elb:healthcheck
- *OptionSettings.member.6.OptionName* = Timeout
- *OptionSettings.member.6.Value* = 5
- *OptionSettings.member.7.Namespace* = aws:elb:healthcheck
- *OptionSettings.member.7.OptionName* = HealthyThreshold
- *OptionSettings.member.7.Value* = 3
- *OptionSettings.member.8.Namespace* = aws:elb:healthcheck
- *OptionSettings.member.8.OptionName* = UnhealthyThreshold
- *OptionSettings.member.8.Value* = 5
- *OptionSettings.member.9.Namespace* = aws:elb:policies
- *OptionSettings.member.9.OptionName* = Stickiness Policy
- *OptionSettings.member.9.Value* = false
- *OptionSettings.member.10.Namespace* = aws:elb:policies
- *OptionSettings.member.10.OptionName* = Stickiness Cookie Expiration
- *OptionSettings.member.10.Value* = 0
- *OptionSettings.member.11.Namespace* = aws:elb:loadbalancer
- *OptionSettings.member.11.OptionName* = LoadBalancerPortProtocol
- *OptionSettings.member.11.Value* = HTTP
- *OptionSettings.member.12.Namespace* = aws:elb:loadbalancer
- *OptionSettings.member.12.OptionName* = LoadBalancerSSLPortProtocol
- *OptionSettings.member.12.Value* = HTTPS
- *OptionSettings.member.13.Namespace* = aws:elb:policies
- *OptionSettings.member.13.OptionName* = ConnectionDrainingEnabled
- *OptionSettings.member.13.Value* = true
- *OptionSettings.member.14.Namespace* = aws:elb:policies
- *OptionSettings.member.14.OptionName* = ConnectionDrainingTimeout
- *OptionSettings.member.14.Value* = 60
- *OptionSettings.member.15.Namespace* = aws:elb:loadbalancer
- *OptionSettings.member.15.OptionName* = CrossZone
- *OptionSettings.member.15.Value* = true

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.1.OptionName=LoadBalancerHTTPPort
&OptionSettings.member.1.Value=80
&OptionSettings.member.2.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.2.OptionName=LoadBalancerHTTPSPort
&OptionSettings.member.2.Value=443
&OptionSettings.member.3.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.3.OptionName=SSLCertificateIdSSLCertificateId
&OptionSettings.member.3.Value=arn%3Aaws%3Aiam%3A%3A123456789012%3Aserver-
certificate%2Fabc%2Fcerts%2Fbuild
&OptionSettings.member.4.Namespace=aws%3Aelb%3Aapplication
&OptionSettings.member.4.OptionName=Application%20Healthcheck%20URL
&OptionSettings.member.4.Value=/
&OptionSettings.member.5.Namespace=aws%3Aelb%3Ahealthcheck
&OptionSettings.member.5.OptionName=Interval
&OptionSettings.member.5.Value=30
&OptionSettings.member.6.Namespace=aws%3Aelb%3Ahealthcheck
&OptionSettings.member.6.OptionName=Timeout
&OptionSettings.member.6.Value=5
&OptionSettings.member.7.Namespace=aws%3Aelb%3Ahealthcheck
&OptionSettings.member.7.OptionName=HealthyThreshold
&OptionSettings.member.7.Value=3
&OptionSettings.member.8.Namespace=aws%3Aelb%3Ahealthcheck
&OptionSettings.member.8.OptionName=UnhealthyThreshold
&OptionSettings.member.8.Value=5
&OptionSettings.member.9.Namespace=aws%3Aelb%3Apolicies
&OptionSettings.member.9.OptionName=Stickiness%20Policy
&OptionSettings.member.9.Value=false
&OptionSettings.member.10.Namespace=aws%3Aelb%3Apolicies
&OptionSettings.member.10.OptionName=Stickiness%20Cookie%20Expiration
&OptionSettings.member.10.Value=0
&OptionSettings.member.11.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.11.OptionName=LoadBalancerPortProtocol
&OptionSettings.member.11.Value=HTTP
&OptionSettings.member.12.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.12.OptionName=LoadBalancerSSLPortProtocol
&OptionSettings.member.12.Value=HTTPS
&OptionSettings.member.13.Namespace=aws%3Aelb%3Apolicies
&OptionSettings.member.13.OptionName=ConnectionDrainingEnabled
&OptionSettings.member.13.Value=true
&OptionSettings.member.14.Namespace=aws%3Aelb%3Apolicies
&OptionSettings.member.14.OptionName=ConnectionDrainingTimeout
&OptionSettings.member.14.Value=60
&OptionSettings.member.15.Namespace=aws%3Aelb%3Aloadbalancer
&OptionSettings.member.15.OptionName=CrossZone
&OptionSettings.member.15.Value=true
&Operation=UpdateEnvironment
&AuthParams
```

Configuring Databases with Elastic Beanstalk

Amazon Web Services offers a number of database options that you can leverage for your application such as Amazon Relational Database Service (Amazon RDS), Amazon DynamoDB, and Amazon ElastiCache.

Amazon RDS is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you to focus on your applications and business.

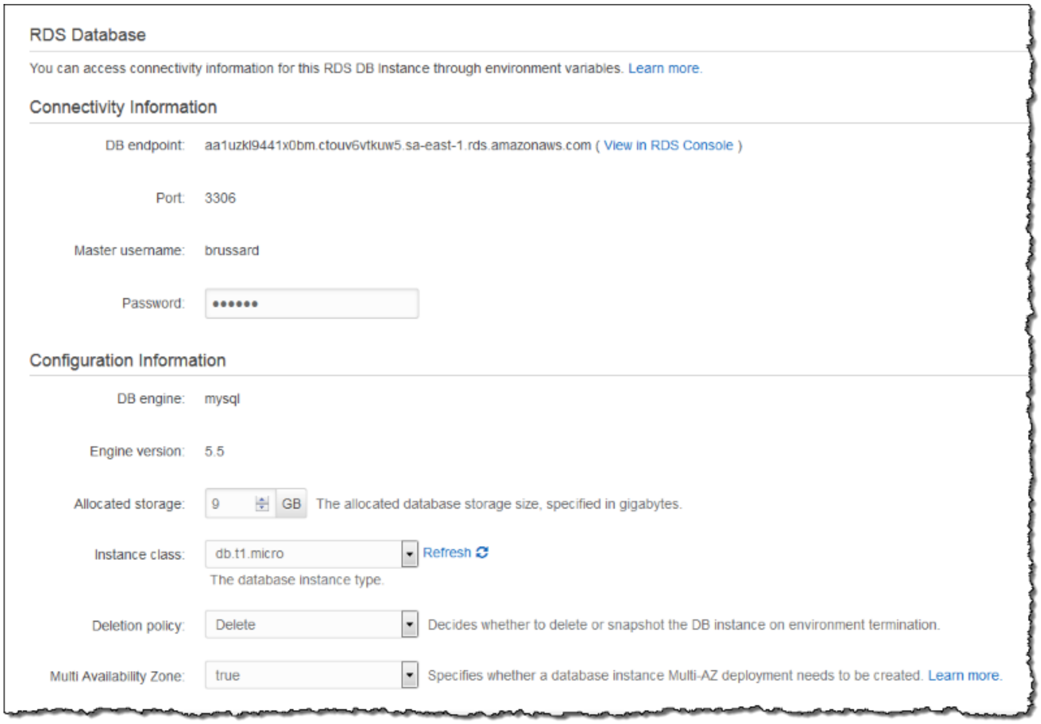
You can create a new RDS database instance for an existing environment and view its settings. For information about using an existing RDS database instance with Elastic Beanstalk, go to Using Elastic Beanstalk with Amazon RDS (p. 294). See the appropriate topic for your programming language.

If you didn't use the **Create New Application** wizard to add an RDS DB instance to your environment, you can use an application's **Configuration** page to do so.

For information about using AWS Elastic Beanstalk with Amazon DynamoDB or Amazon RDS, see Using Elastic Beanstalk with DynamoDB (p. 293) or Using Elastic Beanstalk with Amazon RDS (p. 294)

AWS Management Console

If you have a database associated with your environment, you can view the configuration settings by viewing the settings in the **Data Tier** section on the environment's **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).



RDS Database

You can access connectivity information for this RDS DB instance through environment variables. Learn more.

Connectivity Information

DB endpoint: aa1uzkl9441x0bm.ctouv6vtkuw5.sa-east-1.rds.amazonaws.com (View in RDS Console)

Port: 3306

Master username: brussard

Password: ••••••

Configuration Information

DB engine: mysql

Engine version: 5.5

Allocated storage: 9 GB The allocated database storage size, specified in gigabytes.

Instance class: db.t1.micro Refresh
The database instance type.

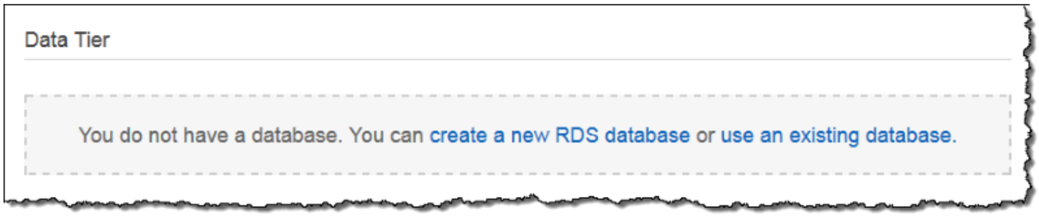
Deletion policy: Delete Decides whether to delete or snapshot the DB instance on environment termination.

Multi Availability Zone: true Specifies whether a database instance Multi-AZ deployment needs to be created. Learn more.

If you don't have an Amazon RDS database associated with your environment, you can associate one by clicking **create a new RDS database** on the **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).

To create an Amazon RDS database and associate it with your existing environment

1. On the **Configuration** page, under **Data Tier**, click **create a new RDS database**.

   Data Tier

   You do not have a database. You can create a new RDS database or use an existing database.

2. Configure the following settings for your database:

   - (Optional) For **Snapshot**, select whether to create an Amazon RDS DB from an existing snapshot.
   - (Optional) For **DB engine**, select a database engine.
   - (Optional) For **Instance Class**, select a database instance class. For information about the DB instance classes, go to http://aws.amazon.com/rds/.
   - For **Allocated Storage**, type the space needed for your database. You can allocate between 5 GB and 1024 GB. You cannot update the allocated storage for a database to a lower amount after you set it. In some cases, allocating a larger amount of storage for your DB instance than the size of your database can improve IO performance. For information about storage allocation, go to Features.
   - For **Master Username**, type a name using alphanumeric characters that you will use to log in to your DB instance with all database privileges.
   - For **Master Password**, type a password containing 8 to 16 printable ASCII characters (excluding /, \, and @).
   - For **Deletion Policy**, select **Create snapshot** to create a snapshot that you can use later to create another Amazon RDS database. Select **Delete** to delete the DB instance when you terminate the environment. If you select **Delete**, you lose your DB instance and all the data in it when you terminate the Elastic Beanstalk instance associated with it. By default, Elastic Beanstalk creates and saves a snapshot. You can use a snapshot to restore data to use in a new environment, but cannot otherwise recover lost data.

     Note
     You may incur charges for storing database snapshots. For more information, see the "Backup Storage" section of Amazon RDS Pricing.

   - For **Availability**, select one of the following:
     - To configure your database in one Availability Zone, select **Single Availability Zone**. A database instance launched in one Availability Zone does not have protection from the failure of a single location.
     - To configure your database across multiple availability zones, select **Multiple Availability Zones**. Running your database instance in multiple Availability Zones helps safeguard your data in the unlikely event of a database instance component failure or service health disruption in one Availability Zone.

3. Click **Save**.

   Elastic Beanstalk updates the environment and creates the Amazon RDS database. After the update is complete, you can view the databases by going to the **Configuration** page.

Use the connectivity information to connect to your DB from inside your application through environment variables. For more information about using Amazon RDS with your applications, see the following topics.

- Java — Using a New Amazon RDS DB Instance with Java (p. 592)
- Node.js — Using a New Amazon RDS DB Instance with Node.js (p. 703)
- .NET — Using a New Amazon RDS DB Instance with .NET (p. 647)
- PHP — Using a New Amazon RDS DB Instance with PHP (p. 729)
- Python — Using a New Amazon RDS DB Instance with Python (p. 762)
- Ruby — Using a New Amazon RDS DB Instance with Ruby (p. 789)

You can use the Elastic Beanstalk Management Console to edit some settings for the Amazon RDS database associated with your environment.

To edit the Amazon RDS database instance associated with your environment

1. On the **Configuration** page, under **Data Tier**, click ⚙ for the **RDS** settings.
2. Update the configuration for any of the following Amazon RDS database settings:

   - For **Master Password**, type a password containing 8 to 16 printable ASCII characters (excluding /, \, and @).
   - For **Allocated Storage**, type the space needed for your database. You can allocate between 5 GB and 1024 GB. You cannot update the allocated storage for a database to a lower amount after you set it. In some cases, allocating a larger amount of storage for your DB instance than the size of your database can improve IO performance. For information about storage allocation, go to Features.
   - (Optional) For **Instance Class**, select a database instance class. For information about the DB instance classes, go to http://aws.amazon.com/rds/.
   - For **Deletion Policy**, select **Create snapshot** to create a snapshot that you can use later to create another Amazon RDS database. Select **Delete** to delete the DB instance when you terminate the environment. If you select **Delete**, you lose your DB instance and all the data in it when you terminate the Elastic Beanstalk instance associated with it. By default, Elastic Beanstalk creates and saves a snapshot. You can use a snapshot to restore data to use in a new environment, but cannot otherwise recover lost data.

     Note
     You may incur charges for storing database snapshots. For more information, see the "Backup Storage" section of Amazon RDS Pricing.
   - For **Availability**, select one of the following:
     - To configure your database in one Availability Zone, select **Single Availability Zone**. A database instance launched in one Availability Zone does not have protection from the failure of a single location.
     - To configure your database across multiple availability zones, select **Multiple Availability Zones**. Running your database instance in multiple Availability Zones helps safeguard your data in the unlikely event of a database instance component failure or service health disruption in one Availability Zone.

3. Click **Save**.

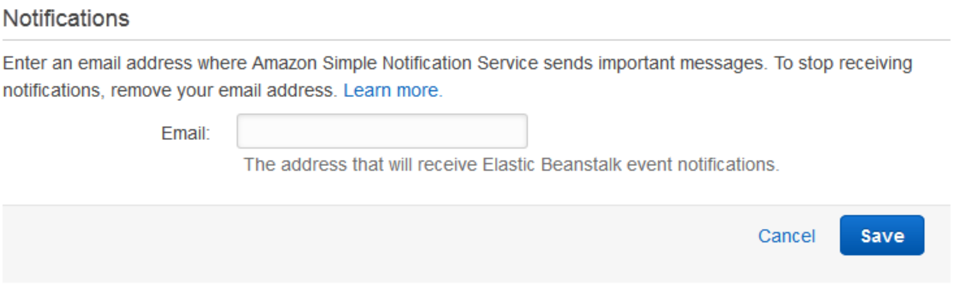
   Elastic Beanstalk updates the environment, replacing the Amazon RDS database if you changed the database instance class.

Configuring Notifications with Elastic Beanstalk

Elastic Beanstalk can use Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application.

AWS Management Console

You can enable Amazon SNS notifications by editing the **Notifications** settings on the environment's **Configuration** page. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).



Notifications

Enter an email address where Amazon Simple Notification Service sends important messages. To stop receiving notifications, remove your email address. Learn more.

Email:

The address that will receive Elastic Beanstalk event notifications.

Command Line Interface (CLI)

To edit an application's environment settings

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:sns:topics",
        "OptionName": "Notification Endpoint",
        "Value": "someone@example.com"
    }
]
```

API

To edit an application's environment settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:sns:topics`
  - *`OptionSettings.member.1.OptionName`* = `Notification Endpoint`

- *`OptionSettings.member.1.Value`* = `someone@example.com`

Example

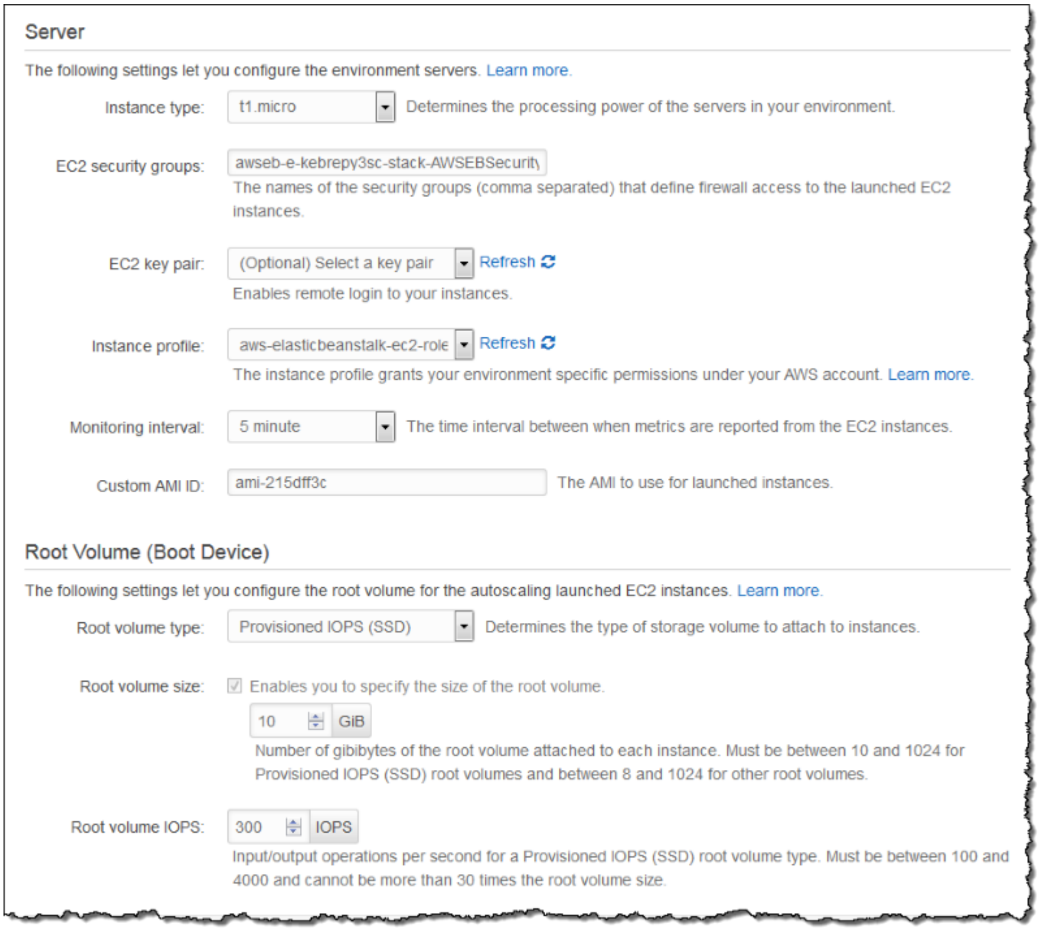
```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Asns%3Atopics
&OptionSettings.member.1.OptionName=Notification%20Endpoint
&OptionSettings.member.1.Value=janedoe%40example.com
&Operation=UpdateEnvironment
&AuthParams
```

Configuring Amazon EC2 Server Instances with Elastic Beanstalk

Amazon EC2 is a web service that enables you to launch and manage server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to the Amazon EC2 product page.

AWS Management Console

You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration by editing **Instances** on the **Configuration** page for that environment. For information about getting to the **Configuration** page, see Changing Environment Configuration Settings (p. 100).



Server

The following settings let you configure the environment servers. Learn more.

Instance type: t1.micro — Determines the processing power of the servers in your environment.

EC2 security groups: awseb-e-kebrepy3sc-stack-AWSEBSecurity
The names of the security groups (comma separated) that define firewall access to the launched EC2 instances.

EC2 key pair: (Optional) Select a key pair — Refresh
Enables remote login to your instances.

Instance profile: aws-elasticbeanstalk-ec2-role — Refresh
The instance profile grants your environment specific permissions under your AWS account. Learn more.

Monitoring interval: 5 minute — The time interval between when metrics are reported from the EC2 instances.

Custom AMI ID: ami-215dff3c — The AMI to use for launched instances.

Root Volume (Boot Device)

The following settings let you configure the root volume for the autoscaling launched EC2 instances. Learn more.

Root volume type: Provisioned IOPS (SSD) — Determines the type of storage volume to attach to instances.

Root volume size: ☑ Enables you to specify the size of the root volume.
10 GiB
Number of gibibytes of the root volume attached to each instance. Must be between 10 and 1024 for Provisioned IOPS (SSD) root volumes and between 8 and 1024 for other root volumes.

Root volume IOPS: 300 IOPS
Input/output operations per second for a Provisioned IOPS (SSD) root volume type. Must be between 100 and 4000 and cannot be more than 30 times the root volume size.

Topics

Amazon EC2 Instance Types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations may require more CPU or memory. Elastic Beanstalk regularly checks your running instances to ensure they are healthy. If your application consumes 95 percent or greater of the CPU, Elastic Beanstalk will trigger an event. For more information about this event, see CPU Utilization Exceeds 95.00% (p. 532).

Note
You cannot change between 32-bit and 64-bit instance types. For example, if your application is built on a 32-bit platform, only 32-bit instance types appear in the list.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see Instance Types in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Security Groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 security group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the Amazon EC2 console. You can specify which Amazon EC2 security groups control access to your Elastic Beanstalk application by entering one or more Amazon EC2 security group names (delimited by commas) into the **EC2 security groups** text box. For more information on Amazon EC2 security groups, see Amazon EC2 Security Groups in the *Amazon Elastic Compute Cloud User Guide*.

Elastic Beanstalk creates a default security group for you. If you are using a legacy container, the security group is **elasticbeanstalk-default**. If you are using a non-legacy container, then Elastic Beanstalk dynamically creates a security group. You can view the security group name in the **EC2 security group** box.

> **Note**
> If you are running your application using a legacy container type, make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see Health Checks (p. 190). To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).

To modify your Amazon EC2 security group

1. Add a new rule for 80 (HTTP) for your EC2 security group with a new source. For instructions, see Adding Rules to a Security Group in the *Amazon Elastic Compute Cloud User Guide*.
2. Type the public DNS address of your EC2 instance in address bar your web browser to verify you can see your application. For instructions on determining your DNS address, see Determining Your Public, Private, and Elastic IP Addresses in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Key Pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

> **Important**
> You must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances. You can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the Amazon Elastic Compute Cloud Getting Started Guide.

The **EC2 key pair** text box lets you specify the name of an Amazon EC2 key pair you use to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.

For more information on Amazon EC2 key pairs, see Network and Security in the *Amazon Elastic Compute Cloud User Guide*. For more information on connecting to Amazon EC2 instances, see Connect to Your Instance and Connecting to Linux/UNIX Instances from Windows using PuTTY in the *Amazon Elastic Compute Cloud User Guide*.

Monitoring Interval

By default, only basic Amazon CloudWatch metrics are enabled; they return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

> **Note**
> Amazon CloudWatch service charges can apply for one-minute interval metrics. See Amazon CloudWatch for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

> **Important**
> Using your own AMI is an advanced task and should be done with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Instance Profiles

If you are using a nonlegacy container, you can select an instance profile. If you are using a legacy container, this option does not appear. Instance profiles provide applications and services access to AWS resources. For example, your application may require access to DynamoDB. Every API request made to AWS services must be signed using AWS security credentials. One way to grant applications access to AWS resources is to distribute your credentials to each instance; however, distributing long-term credentials to each instance is challenging to manage and a potential security risk. Instead, Elastic Beanstalk requires an IAM role with the permissions that applications must have when an application makes calls to other AWS resources. When Elastic Beanstalk launches the Amazon EC2 instances, it uses the instance profile associated with an IAM role. All applications that run on the instances can use the role credentials to sign requests. Because role credentials are temporary and rotated automatically, you don't have to worry about long-term security risks.

In addition,

The **Instance profile** list displays the profiles available for your Elastic Beanstalk environment. If you do not have any instance profiles, Elastic Beanstalk creates one for you. Elastic Beanstalk creates a default instance profile and updates the Amazon S3 bucket policy to allow log rotation. If you choose to not use the default instance profile, you need to grant permissions for Elastic Beanstalk to rotate logs. For more information about this policy, see Example: Granting Elastic Beanstalk Permission to Rotate Logs to Amazon S3 (p. 340). For a list of supported container types, see Why are some container types marked legacy? (p. 92).

> **Note**
> Users must have permission to create a default profile. For more information, see Granting IAM Users Permissions to Create and Pass IAM Roles (p. 332).

Root Volume (Boot Device)

You can configure a root volume (otherwise known as a boot device) to attach to Amazon EC2 instances in your Elastic Beanstalk environment. An Amazon EBS volume is a durable, block-level storage device that you can attach to a single Amazon EC2 instance. After a volume is attached to an instance, you can use it like any other physical hard drive. The **Root volume type** list includes **Magnetic**, **General Purpose**

(SSD), and **Provisioned IOPS (SSD)** volume types. Select the volume type that meets your performance and price requirements. For more information, see Amazon EBS Volume Types and Amazon EBS Product Details.

With **Root volume size**, you can specify the size of the storage volume that you selected. You must specify your desired root volume size if you choose Provisioned IOPS (SSD) as the root volume type that your instances will use. For other root volumes, if you do not specify your own value, Elastic Beanstalk will use the default volume size for the storage volume type. The default volume size varies according to the AMI of the solution stack on which your environment is based. For Provisioned IOPS (SSD) root volumes, the minimum number of gibibytes is 10 and the maximum is 1024. For other root volumes, the minimum number of gibibytes is 8 and the maximum is 1024.

If you selected Provisioned IOPS (SSD) as your root volume type, you must specify your desired input/output operations per second (IOPS). The minimum is 100 and the maximum is 4000. The maximum ratio of IOPS to your volume size is 30 to 1. For example, a volume with 3000 IOPS must be at least 100 GiB.

Block Device Mappings

> **Note**
> You cannot configure this option using the AWS Management Console. Use the EB CLI config command (p. 405) or EB Extensions (p. 99) to change the required configuration options in the aws:autoscaling:launchconfiguration (p. 105) namespace.

Although each Amazon Elastic Compute Cloud instance has an associated root device volume upon launch, you can use block device mappings to specify additional Amazon Elastic Block Store volumes or instance store volumes to attach to all the instances in the autoscaling group. For more information about block device mappings, see Block Device Mapping in the *Amazon Elastic Cloud Computer User Guide*. For more information about instance storage, see Amazon EC2 Instance Store in the *Amazon Elastic Cloud Computer User Guide*.

Command Line Interface (CLI)

To edit an application's environment settings

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "InstanceType",
        "Value": "m1.small"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "SecurityGroups",
        "Value": "awseb-e-98pjjgr9cs-stack-AWSEBSecurityGroup-D1FOQASTKD12"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "EC2KeyName",
        "Value": "mykeypair"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "MonitoringInterval",
        "Value": "5 minute"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "ImageId",
        "Value": "ami-cbab67a2"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "ElasticBeanstalkProfile"
    },
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "BlockDeviceMappings",
        "Value": "/dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0"
    }
]
```

API

For information about all the option values you can pass, see Option Values.

To edit an application's environment settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.1.OptionName`* = `InstanceType`
  - *`OptionSettings.member.1.Value`* = `m1.small`
  - *`OptionSettings.member.2.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.2.OptionName`* = `SecurityGroups`
  - *`OptionSettings.member.2.Value`* = `mysecuritygroup`
  - *`OptionSettings.member.3.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.3.OptionName`* = `EC2KeyName`
  - *`OptionSettings.member.3.Value`* = `mykeypair`
  - *`OptionSettings.member.4.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.4.OptionName`* = `MonitoringInterval`
  - *`OptionSettings.member.4.Value`* = `1 minute`
  - *`OptionSettings.member.5.Namespace`* = `aws:autoscaling:launchconfiguration`
  - *`OptionSettings.member.5.OptionName`* = `ImageId`
  - *`OptionSettings.member.5.Value`* = `ami-cbab67a2`

- *OptionSettings.member.6.Namespace* = aws:autoscaling:launchconfiguration
- *OptionSettings.member.6.OptionName* = IamInstanceProfile
- *OptionSettings.member.6.Value* = ElasticBeanstalkProfile
- *OptionSettings.member.7.Namespace* = aws:autoscaling:launchconfiguration
- *OptionSettings.member.7.OptionName* = BlockDeviceMappings
- *OptionSettings.member.7.Value* = /dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=InstanceType
&OptionSettings.member.1.Value=m1.small
&OptionSettings.member.2.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.2.OptionName=SecurityGroups
&OptionSettings.member.2.Value=awseb-e-98pjjgr9cs-stack-AWSEBSecurityGroup-
D1FOQASTKD12
&OptionSettings.member.3.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.3.OptionName=EC2KeyName
&OptionSettings.member.3.Value=mykeypair
&OptionSettings.member.4.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.4.OptionName=MonitoringInterval
&OptionSettings.member.4.Value=5%20minute
&OptionSettings.member.5.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.5.OptionName=ImageId
&OptionSettings.member.5.Value=ami-cbab67a2
&OptionSettings.member.6.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.6.OptionName=IamInstanceProfile
&OptionSettings.member.6.Value=ElasticBeanstalkProfile
&OptionSettings.member.7.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.7.OptionName=BlockDeviceMappings
&OptionSettings.member.7.Value=/dev/sdj=:100,/dev/sdh=snap-
51eef269,/dev/sdb=ephemeral0
&Operation=UpdateEnvironment
&AuthParams
```

Configuring VPC with Elastic Beanstalk

Amazon Virtual Private Cloud (Amazon VPC) enables you to define a virtual network in your own isolated section within the Amazon Web Services (AWS) cloud, known as a *virtual private cloud (VPC)*. Using VPC, you can deploy a new class of web applications on Elastic Beanstalk, including internal web applications (such as your recruiting application), web applications that connect to an on-premise database (using a VPN connection), as well as private web service back ends. Elastic Beanstalk launches your AWS resources, such as instances, into your VPC. Your VPC closely resembles a traditional network, with the benefits of using AWS's scalable infrastructure. You have complete control over your VPC; you can select the IP address range, create subnets, and configure routes and network gateways. To protect the resources in each subnet, you can use multiple layers of security, including security groups and network access control lists. For more information about Amazon VPC, go to the Amazon VPC User Guide.

You can view your environment's VPC configuration by viewing the **VPC** settings on the environment's **Configuration** page. If you do not see the **VPC** settings on the **Configuration** page, your current environment is not inside a VPC either because you are using a legacy container or you created the Elastic Beanstalk environment outside a VPC. To learn how to migrate to a nonlegacy container to use Amazon VPC with Elastic Beanstalk, see Migrating Your Application from a Legacy Container Type (p. 91). To learn how to create your VPC and launch your Elastic Beanstalk environment inside your VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).

If you want to update your VPC settings, launch a new environment. For more information, see Launching a New AWS Elastic Beanstalk Environment (p. 55).

This feature is not supported by **legacy** containers. See To check if you are using a legacy container type (p. 92) if you are unsure if you are on a legacy configuration or not.

Configuring SSL for Single-Instance Environments

With AWS Elastic Beanstalk, you can configure SSL for single-instance environments that have applications running on the following:

- Docker
- Java (Apache Tomcat 6, Apache Tomcat 7, and Apache Tomcat 8)
- Node.js
- PHP 5.3, PHP 5.4, and PHP 5.5
- Python
- Ruby

To configure SSL for a single-instance environment, you must create a custom configuration file. (For Ruby containers, an additional JSON file is required.) Currently, you cannot use the Elastic Beanstalk console or API to configure SSL for a single-instance environment.

The process has four steps:

- Step 1: Create an SSL Certificate and Private Key (p. 206)
- Step 2: Create an SSL Configuration File (p. 207)
- Step 3: Open Port 443 (p. 207)
- Step 4: Complete the Configuration File for Your Container Type (p. 208)

Step 1: Create an SSL Certificate and Private Key

Before you can enable SSL, you must have an RSA private key and a certificate. To create a private key and certificate request using OpenSSL, follow the steps for configuring HTTPS beginning with Install and Configure OpenSSL (p. 229) and ending with Submit the CSR to Certificate Authority (p. 231). The contents of the certificate and private key files are added to the configuration file in Step 4: Complete the Configuration File for Your Container Type (p. 208).

Note
For testing purposes, you can use a self-signed certificate, but do not use a self-signed certificate in a production environment. For a production environment, you need a certificate from an external Certification Authority (CA), such as Symantec or Entrust.

Step 2: Create an SSL Configuration File

Using a text editor, create an empty configuration file with the extension `.config` (e.g., `singlessl.config`) and place it in the `.ebextensions` directory at the top level of your source bundle. You will modify this file in the following steps. For information about the format of configuration files, see Using Configuration Files (p. 99). For information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147). For information about source bundles, see Create an Application Source Bundle (p. 43).

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Step 3: Open Port 443

To enable SSL, you must allow incoming traffic on port 443 to the Amazon Elastic Compute Cloud (Amazon EC2) instance that your Elastic Beanstalk application is running on. You do this by using the `Resources` key in the configuration file to add a rule for port 443 to the ingress rules for the AWSEBSecurityGroup security group. When the application is uploaded and deployed, Elastic Beanstalk updates the security group with the new rule. For more information about security groups and ingress rules, see Amazon EC2 Security Groups.

> **Important**
> Using SSL with load-balanced environments involves a different procedure than using SSL in a single instance environment. If at any point you decide to redeploy your application using a load-balanced environment, you risk opening port 443 to all incoming traffic from the Internet. In this case, delete the configuration file from your `.ebextensions` directory, create a load-balanced environment, and set up SSL using the **Load Balancer** section of the **Configuration** page of the Elastic Beanstalk console.

The following snippet adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic for EC2-Classic (non-VPC) security groups. Note the use of the `GroupName` property to identify the security group.

Example .ebextensions Snippet for Opening Port 443 for an EC2-Classic Security Group

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

The following snippet adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic for VPC security groups. Note the use of the `GroupId` property to identify the security group.

Example .ebextensions Snippet for Opening Port 443 for a VPC Security Group

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

You'll find this code in each of the examples in the next step.

Step 4: Complete the Configuration File for Your Container Type

The contents of the configuration file depend on the container type. Use the following examples for guidance for your container type:

- Configuring SSL on Single Instances of Docker (p. 208)
- Configuring SSL on Single Instances of Java/Apache Tomcat 6, Apache Tomcat 7, or Apache Tomcat 8 (p. 211)
- Configuring SSL on Single Instances of Node.js (p. 213)
- Configuring SSL on Single Instances of PHP 5.3, PHP 5.4, and PHP 5.5 (p. 218)
- Configuring SSL on Single Instances of Python (p. 220)
- Configuring SSL on Single Instances of Ruby (p. 224)

Configuring SSL on Single Instances of Docker

For Docker containers, you use a configuration file (p. 207) to enable SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `files` key creates the following files on the instance:

  `/etc/nginx/conf.d/ssl.conf`

  Configures the Nginx server. This file is loaded when the Nginx service starts.

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

  `/etc/pki/tls/certs/server.key`

  Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Docker

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

files:
  /etc/nginx/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      # HTTPS Server

      server {
        listen 443;
        server_name localhost;

        ssl on;
        ssl_certificate /etc/pki/tls/certs/server.crt;
        ssl_certificate_key /etc/pki/tls/certs/server.key;

        ssl_session_timeout 5m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
        ssl_prefer_server_ciphers on;

        location / {
          proxy_pass http://docker;
          proxy_http_version 1.1;

          proxy_set_header Connection "";
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
      }

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----
```

Configuring SSL on Single Instances of Java/Apache Tomcat 6, Apache Tomcat 7, or Apache Tomcat 8

For Java container types, you use a configuration file (p. 207) to enable the Apache HTTP Server to use SSL when acting as the reverse proxy for Apache Tomcat 6, Apache Tomcat 7, and Apache Tomcat 8. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `packages` key uses yum to install `mod_ssl`.
- The `files` key creates the following files on the instance:

  `/etc/httpd/conf.d/ssl.conf`
  Configures the Apache server. It is loaded when the Apache service starts.

  `/etc/pki/tls/certs/server.crt`
  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

  `/etc/pki/tls/certs/server.key`
  Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.
- For some HTTPS clients, such as those running on Android devices, you must configure the Apache HTTP Server with an intermediate certificate authority (CA) bundle and add the following to your SSL configuration file (p. 207):
  - In the `files` key, `SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"`
  - In the `files` key, the contents of the intermediate certificate file
  - In the list of files in the `services` key, the path to the intermediate certificate file
- The `services` key restarts the Apache server after everything is configured so that it will use the new `ssl.conf` file.

Note
Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Java

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

packages:
  yum:
    mod_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
          Order deny,allow
          Allow from all
        </Proxy>

        SSLEngine             on
        SSLCertificateFile    "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
        SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
        SSLProtocol           All -SSLv2 -SSLv3
        SSLHonorCipherOrder   On

       Header always set Strict-Transport-Security "max-age=63072000; include
Subdomains; preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        ProxyPass / http://localhost:8080/ retry=0
        ProxyPassReverse / http://localhost:8080/
        ProxyPreserveHost on

        LogFormat "%h (%{X-Forwarded-For}i) %l %u %t \"%r\" %>s %b \"%{Refer
er}i\" \"%{User-Agent}i\""
        ErrorLog /var/log/httpd/elasticbeanstalk-error_log
        TransferLog /var/log/httpd/elasticbeanstalk-access_log
      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----

services:
  sysvinit:
    httpd:
      enabled: true
      ensureRunning: true
      files : [/etc/httpd/conf.d/ssl.conf,/etc/pki/tls/certs/serv
er.key,/etc/pki/tls/certs/server.crt]
```

Configuring SSL on Single Instances of Node.js

For Node.js container types, you use a configuration file (p. 207) to enable SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `files` key creates the following files on the instance:

  `/etc/nginx/conf.d/ssl.conf`
  Configures the Nginx server. This file is loaded when the Nginx service starts.

  `/etc/pki/tls/certs/server.crt`
  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

  `/etc/pki/tls/certs/server.key`
  Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.

  For some HTTPS clients, such as those running on Android devices, you must include intermediate certificates to ensure that the client to trusts the connection. Concatenate intermediate certificates in order starting with the one immediately above your site's certificate onto your web site's certificate and add the entire bundle to your configuration file (p. 207) in the `files` key.

  For example, the following command from the Linux command line concatenates multiple certificates to create a bundle:

```
$ cat YourWebserverCert.crt FirstIntermediateCert.crt SecondIntermediate
Cert.crt > bundle.crt
```

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Node.js

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

files:
  /etc/nginx/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      # HTTPS server

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
          ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
          ssl_prefer_server_ciphers   on;

          location / {
              proxy_pass  http://nodejs;
              proxy_set_header   Connection "";
              proxy_http_version 1.1;
              proxy_set_header        Host            $host;
              proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header       X-Forwarded-For $proxy_add_x_forwarded_for;

          }
      }
  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----
```

If SSL does not work after you configure it with the preceding configuration file, you might need to include additional information to configure an upstream Node.js server. The following example includes the upstream directive to the Nginx server to proxy requests to port 443 on the Node.js server with IP address 127.0.0.1.

Example .ebextensions Snippet for Configuring SSL for Node.js (with Upstream Directive)

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

files:
  /etc/nginx/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      # HTTPS server

      upstream nodejs {
      server 127.0.0.1:8443;
      keepalive 256;
      }

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
         ssl_ciphers  ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

          ssl_prefer_server_ciphers   on;

          location / {
              proxy_pass  http://nodejs;
              proxy_set_header   Connection "";
              proxy_http_version 1.1;
              proxy_set_header        Host            $host;
              proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;

          }
      }
  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----
```

Configuring SSL on Single Instances of PHP 5.3, PHP 5.4, and PHP 5.5

For PHP container types, you use a configuration file (p. 207) to enable the Apache HTTP Server to use SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `packages` key uses yum to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

  `/etc/httpd/conf.d/ssl.conf`

  Configures the Apache server. This file is loaded when the Apache service starts.

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

  `/etc/pki/tls/certs/server.key`

  Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.
- For some HTTPS clients, such as those running on Android devices, you must configure the Apache HTTP Server with an intermediate certificate authority (CA) bundle and add the following to your SSL configuration file (p. 207) in the files key:
  - `SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"`
  - The contents of the intermediate certificate file

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for PHP

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
          Order deny,allow
          Allow from all
        </Proxy>

        SSLEngine             on
        SSLCertificateFile    "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
        SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
        SSLProtocol           All -SSLv2 -SSLv3
        SSLHonorCipherOrder   On
        SSLSessionTickets     Off

       Header always set Strict-Transport-Security "max-age=63072000; include
Subdomains; preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        ProxyPass / http://localhost:80/ retry=0
        ProxyPassReverse / http://localhost:80/
        ProxyPreserveHost on
        RequestHeader set X-Forwarded-Proto "https" early

        LogFormat "%h (%{X-Forwarded-For}i) %l %u %t \"%r\" %>s %b \"%{Refer
er}i\" \"%{User-Agent}i\""
        ErrorLog /var/log/httpd/elasticbeanstalk-error_log
        TransferLog /var/log/httpd/elasticbeanstalk-access_log
      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----
```

Configuring SSL on Single Instances of Python

For Python container types using Apache HTTP Server with the Web Server Gateway Interface (WSGI), you use a configuration file (p. 207) to enable the Apache HTTP Server to use SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `packages` key uses yum to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

  `/etc/httpd/conf.d/ssl.conf`
  Configures the Apache server. This file is loaded when the Apache service starts.

  `/etc/pki/tls/certs/server.crt`
  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

  `/etc/pki/tls/certs/server.key`
  Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.
- For some HTTPS clients, such as those running on Android devices, you must configure the Apache HTTP Server with an intermediate certificate authority (CA) bundle and add the following to your SSL configuration file (p. 207) in the files key:
  - `SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"`
  - The contents of the intermediate certificate file
- The `container_commands` key stops the httpd service after everything has been configured so that the service uses the new `ssl.conf` file and certificate.

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Python 2.6

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
          Require all granted
        </Proxy>

        SSLEngine             on
        SSLCertificateFile    "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
        SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
        SSLProtocol           All -SSLv2 -SSLv3
        SSLHonorCipherOrder   On
        SSLSessionTickets     Off

       Header always set Strict-Transport-Security "max-age=63072000; include
Subdomains; preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        Alias /static /opt/python/current/app/
        <Directory /opt/python/current/app/>
        Order allow,deny
        Allow from all
        </Directory>

        WSGIScriptAlias / /opt/python/current/app/application.py

        <Directory /opt/python/current/app/>
        Order allow,deny
        Allow from all
        </Directory>

       WSGIDaemonProcess wsgi-ssl processes=1 threads=15 display-name=%{GROUP}
 \
          python-path=/opt/python/current/app:/opt/python/run/venv/lib/py
thon2.6/site-packages user=wsgi group=wsgi \
          home=/opt/python/current/app
        WSGIProcessGroup wsgi
      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----

container_commands:
  01killhttpd:
    command: "killall httpd"
  02waitforhttpddeath:
    command: "sleep 3"
```

Example .ebextensions Snippet for Configuring SSL for Python 2.7

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000755"
    owner: root
    group: root
    content: |
      LoadModule wsgi_module modules/mod_wsgi.so
      WSGIPythonHome /opt/python/run/baselinenv
      WSGISocketPrefix run/wsgi
      WSGIRestrictEmbedded On
      Listen 443
      <VirtualHost *:80>
        ServerName myserver
        Redirect permanent / https://myserver
      </VirtualHost>

      <VirtualHost *:443>
        ServerName myserver


        SSLEngine on
        SSLCertificateFile "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

        Alias /static/ /opt/python/current/app/static/
        <Directory /opt/python/current/app/static>
        Order allow,deny
        Allow from all
        </Directory>

        WSGIScriptAlias / /opt/python/current/app/application.py

        <Directory /opt/python/current/app>
        Require all granted
        </Directory>

       WSGIDaemonProcess wsgi-ssl processes=1 threads=15 display-name=%{GROUP}
 \
          python-path=/opt/python/current/app:/opt/python/run/venv/lib/py
thon2.7/site-packages user=wsgi group=wsgi \
          home=/opt/python/current/app
        WSGIProcessGroup wsgi-ssl
      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----

container_commands:
  01killhttpd:
    command: "killall httpd"
  02waitforhttpddeath:
    command: "sleep 3"
```

Configuring SSL on Single Instances of Ruby

For Ruby container types, the way you enable SSL depends on the type of application server used.

Topics

Configure SSL for Ruby with Puma

For Ruby container types that use Puma as the application server, you use a configuration file (p. 207) to enable SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
- The `files` key creates the following files on the instance:

  `/etc/nginx/conf.d/ssl.conf`

  Configures the Nginx server. This file is loaded when the Nginx service starts.

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

`/etc/pki/tls/certs/server.key`
: Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the Nginx server after everything is configured so that the server uses the new `ssl.conf` file.

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Ruby with Puma

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

files:
  /etc/nginx/conf.d/ssl.conf:
    content: |
      # HTTPS server

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
          ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
          ssl_prefer_server_ciphers   on;

          location / {
              proxy_pass  http://my_app;
              proxy_set_header        Host            $host;
            proxy_set_header       X-Forwarded-For $proxy_add_x_forwarded_for;

          }

          location /assets {
            alias /var/app/current/public/assets;
            gzip_static on;
            gzip on;
            expires max;
            add_header Cache-Control public;
          }

          location /public {
            alias /var/app/current/public;
            gzip_static on;
            gzip on;
            expires max;
            add_header Cache-Control public;
          }
      }

  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Configure SSL for Ruby with Passenger

For Ruby container types that use Passenger as the application server, you use both a configuration file (p. 207) and a JSON file to enable SSL. For more information about the contents of configuration files, see Customizing Software on Linux Servers (p. 147).

To configure SSL for Ruby with Passenger

1. Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

   - The `Resources` key enables port 443 on the security group used by your environment's instance. For security groups that are in a VPC, you must replace the `GroupName` property with `GroupId`.
   - The `files` key creates the following files on the instance:

     `/etc/pki/tls/certs/server.crt`
     Creates the certificate file on the instance. Replace *`<certificate file contents>`* with the contents of your certificate.

     `/etc/pki/tls/certs/server.key`
     Creates the private key file on the instance. Replace *`<private key contents>`* with the contents of the private key used to create the certificate request or self-signed certificate.

   Note
   Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Example .ebextensions Snippet for Configuring SSL for Ruby with Passenger

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      # GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      <certificate file contents>
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      <private key contents>
      -----END RSA PRIVATE KEY-----
```

2. Create a text file and add the following JSON to the file. Save it in your source bundle's root directory with the name `passenger-standalone.json`. This JSON file configures Passenger to use SSL.

   Important
   This JSON file must not contain a byte order mark (BOM). If it does, the Passenger JSON library will not read the file correctly and the Passenger service will not start.

Example JSON File Snippet for Configuring SSL for Ruby with Passenger

```
{
  "ssl" : true,
  "ssl_port" : 443,
  "ssl_certificate" : "/etc/pki/tls/certs/server.crt",
  "ssl_certificate_key" : "/etc/pki/tls/certs/server.key"
}
```

Configuring HTTPS for your Elastic Beanstalk Environment

You can configure your Elastic Beanstalk environment to use HTTPS for your application. Configuring HTTPS ensures traffic encryption for client connections to the load balancer in load-balancing autoscaling environments.

Note
For single-instance environments, see Configuring SSL for Single-Instance Environments (p. 206).

This section walks you through the necessary steps to configure HTTPS for your Elastic Beanstalk application. This section assumes you have already deployed an Elastic Beanstalk application. If you have not already deployed an Elastic Beanstalk application, do this now. For more information, see Getting Started Using Elastic Beanstalk (p. 4).

To configure HTTPS for your Elastic Beanstalk application, you must perform the following tasks:

Tasks

- Step 1: Create a Custom Domain Name (p. 229)
- Step 2: Create an X509 Certificate (p. 229)
- Step 3: Upload the Certificate to IAM (p. 232)
- Step 4: Update Your Elastic Beanstalk Environment to Use HTTPS (p. 233)

Step 1: Create a Custom Domain Name

You must create a custom domain name, such as `www.example.com`, to obtain a digitally signed SSL certificate. When obtaining a signed SSL certificate, the Certificate Authority (CA) checks the domain name to ensure that you are the owner of that domain. Because your default Elastic Beanstalk URL contains `elasticbeanstalk.com`, you cannot obtain a certificate for this domain name.

To create a custom domain name, you can use Amazon Route 53 or a third party DNS provider. For more information about using a custom domain name, see Using Custom Domains with Elastic Beanstalk (p. 240).

Step 2: Create an X509 Certificate

After you have created your custom domain name, you must create an X509 certificate. To create a certificate, perform the following tasks:

Tasks

- Install and Configure OpenSSL (p. 229)
- Create a Private Key (p. 230)
- Create a Certificate Signing Request (p. 230)
- Submit the CSR to Certificate Authority (p. 231)

Install and Configure OpenSSL

To create your private key and certificate signing request, you will need a tool, such as OpenSSL, that supports the SSL and TLS protocols. OpenSSL is an open-source tool that provides the basic cryptographic functions necessary to create an RSA key and certificate signing request.

If you already have OpenSSL or another SSL tool installed, you can proceed to Create a Private Key (p. 230). If you do not already have OpenSSL installed, perform the following steps.

1. Install OpenSSL:

   To install OpenSSL On Linux and UNIX:

   1. Go to OpenSSL: Source, Tarballs.
   2. Download the latest source.
   3. Build the package.

To install OpenSSL On Windows:

1. Go to OpenSSL: Binary Distributions.
2. Choose **OpenSSL for Windows**. A new page displays with links to the Windows downloads.
3. Follow the instructions in the `OpenSSL Setup Wizard`. Install OpenSSL in a folder on your local machine.

2. Set the `OpenSSL_HOME` environment variable.

   To set the `OpenSSL_HOME` variable on Linux and UNIX, open a terminal and enter the following command:

   ```
   & export OpenSSL_HOME=path_to_your_OpenSSL_installation
   ```

   To set the `OpenSSL_HOME` variable on Windows, open a comamnd prompt and enter the following command:

   ```
   > set OpenSSL_HOME=path_to_your_OpenSSL_installation
   ```

   Note
   Any changes you make to the environment variables are valid only for the current command-line session.

Create a Private Key

You need to create an RSA private key that is used to create your certificate signing request (CSR). To create your private key, use the **openssl genrsa** command:

```
PROMPT> openssl genrsa 2048 > privatekey.pem
```

`privatekey.pem`
The name of the file to save the private key to. Normally, the **openssl genrsa** command prints the private key contents to the screen, but this command pipes the output to a file. This can be any file name you choose. You must store this file in a secure place so that you can retrieve it later. If you lose your private key, your certificate will become useless.

Create a Certificate Signing Request

A certificate signing request (CSR) is a file you send to a certificate authority (CA) to apply for a digital server certificate. To create a CSR, use the **openssl req** command:

```
PROMPT> openssl req -new -key privatekey.pem -out csr.pem
```

`privatekey.pem`
Your private key.

`csr.pem`
The file to save the certificate signing request to. This can be any file name you choose. You send this file to the CA to have your certificate created.

The output of the **openssl req** will look similar to the following example:

```
You are about to be asked to enter information that will be incorporated
 into your certificate request.
 What you are about to enter is what is called a Distinguished Name or a DN.
 There are quite a few fields but you can leave some blank
 For some fields there will be a default value,
 If you enter '.', the field will be left blank.
```

The following table can help you create your certificate request.

Name	Description	Example
Country Name	The two-letter ISO abbreviation for your country.	US = United States
State or Province	The name of the state or province where your organization is located. This name cannot be abbreviated.	Washington
Locality Name	The name of the city where your organization is located.	Seattle
Organization Name	The full legal name of your organization. Do not abbreviate your organization name.	CorporationX
Organizational Unit	Optional, for additional organization information.	Marketing
Common Name	The fully qualified domain name for your site. This is your custom domain name created previously. You will receive a certificate name check warning if this is not an exact match.	www.example.com
Email address	The server administrator's email address	someone@example.com

Note
The Common Name field is often misunderstood and completed incorrectly. The common name is typically your domain name. It will look like `www.example.com` or `example.com`. You need to create a CSR using the correct common name.

Submit the CSR to Certificate Authority

Normally, at this stage, you would submit your CSR to a certificate authority (CA) to apply for a digital server certificate. However, you can also generate a self-signed certificate for testing purposes only. For this example, you'll generate a self-signed certificate.

To generate a self-signed certificate, use the **openssl x509** command:

```
PROMPT> openssl x509 -req -days 365 -in csr.pem -signkey privatekey.pem -out
server.crt
```

`365`
The number of days until the certificate expires.

`csr.pem`
The file containing the certificate signing request.

`privatekey.pem`
Your private key.

server.crt
The name of the file to save the certificate to.

The output will look similar to the following example:

```
Loading 'screen' into random state - done
Signature ok
subject=/C=us/ST=washington/L=seattle/O=corporationx/OU=marketing/CN=ex
ample.com/emailAddress=someone@example.com
Getting Private key
```

Step 3: Upload the Certificate to IAM

To enable you to use your certificate with AWS services such as Elastic Beanstalk, you must upload the certificate and private key to IAM. After you upload the certificate to IAM, the certificate is available for other AWS services to use. The IAM console does not provide a way to upload a certificate, so you must use the AWS Command Line Interface (AWS CLI) to upload your certificate.

To upload a signed certificate

- Use the IAM **upload-server-certificate** command to upload your certificate:

```
PROMPT>aws iam upload-server-certificate \
--server-certificate-name <certificate_name> \
--certificate-body file://<certificate_file> \
--private-key file://<private_key_file> \
--certificate-chain file://<certificate_chain_file>
```

<certificate_name>
The name to apply to the certificate object. This can be any name you choose. The name can only contain alphanumeric and hyphen characters.

<certificate_file>
Your certificate file.

<private_key_file>
Your private key.

<certificate_chain_file>
The chain file for the certificate.

Note
If you are uploading a self-signed certificate and it's not important that browsers implicitly accept the certificate, you can omit the **--certificate-chain** option and upload just the server certificate and private key.

The response will look similar to the following:

```
{
  "ServerCertificateMetadata" :
  {
    "ServerCertificateId" : "<certificate_id>",
    "ServerCertificateName" : "<certificate_name>",
    "Expiration" : "<cert_expriation_date>",
    "Path" : "/",
    "Arn" : "arn:aws:iam::<account_id>:server-certificate/<certificate_name>",

    "UploadDate" : "<cert_upload_date>"
  }
}
```

Make note of the Amazon resources name (ARN) for your certificate, which you will use when you update your load balancer configuration settings to use HTTPS.

If you have a certificate that results in an error when you upload it, ensure that it meets the criteria, and then try uploading it again.

To see sample certificates that are valid with IAM, go to Sample Certificates in the *AWS Identity and Access Management Using IAM User Guide*.

Step 4: Update Your Elastic Beanstalk Environment to Use HTTPS

After you upload your certificate to IAM, you need to complete the following tasks to configure your application to use HTTPS.

Tasks

- Update Your Security Group (p. 233)
- Configure Your Load Balancer (p. 234)

Update Your Security Group

You must add a rule to your security group that allows inbound traffic from 0.0.0.0/0 to port 443. You do this by adding a `Resources` key to a configuration file in the `.ebextensions` directory for your application. For information about configuration files, see Using Configuration Files (p. 99).

Alternatively, you can add this rule manually. For more information about manually adding rules to a security group, see Adding Rules to a Security Group in the Amazon EC2 User Guide for Linux Instances.

> **Important**
> If at any point you decide to redeploy your application using a load-balanced environment, you risk opening port 443 to all incoming traffic from the Internet. In that case, delete the configuration file from your `.ebextensions` directory. Then create a load-balanced environment and set up SSL using the **Load Balancer** section of the **Configuration** page of the Elastic Beanstalk management console.

The following configuration file example adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic. This snippet is used for EC2-Classic (non-VPC) security groups. Note the use of `GroupName` to identify the security group.

Example .ebextensions snippet opening port 443 for an EC2-Classic security group

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

The following configuration file example adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic. This snippet is used for VPC security groups. Note the use of `GroupId` to identify the security group.

Example .ebextensions snippet opening port 443 for a VPC security group

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

Configure Your Load Balancer

You need to update the load balancer configuration settings for your Elastic Beanstalk environment to enable HTTPS. To do this, you need to set the following options on the environment:

- Load Balancer HTTP Port
- Load Balancer Port Protocol
- Load Balancer HTTPS Port
- Load Balancer SSL Port Protocol
- SSL Certificate ID

How you set these options varies depending on the method you user to update your environment. There are several methods you can use to update your environment. The following list provides links to the relevant instructions.

- Elastic Beanstalk console (p. 235)
- Elastic Beanstalk API (p. 192). With this method, you need to set the following option values on the environment:

  `LoadBalancerHTTPPort`

    `OFF`

      The load balancer does not listen to HTTP requests.

    `80`

      The load balancer listens to HTTP requests.

  `LoadBalancerPortProtocol`

    `HTTP`

`LoadBalancerHTTPSPort`
: `443` or `8443`.

> **Note**
> If you are using Amazon VPC with Elastic Beanstalk, you must set `LoadBalancerHTTPSPort` to `443`.

`LoadBalancerSSLPortProtocol`
: `HTTPS`

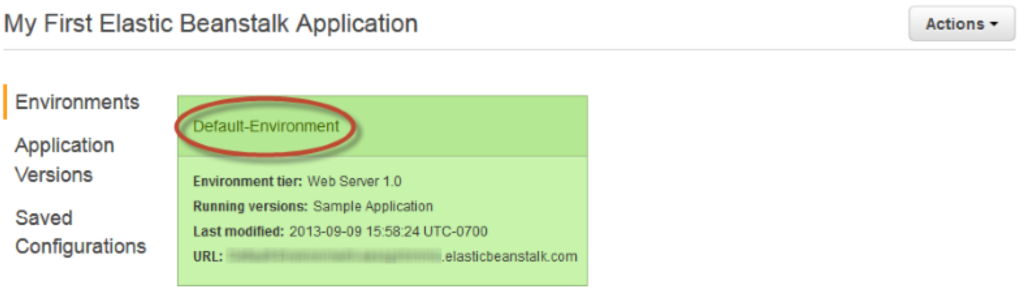
`SSLCertificateId`
: The ARN of the certificate you uploaded to IAM.

- AWS Toolkit for Eclipse
- AWS Toolkit for Visual Studio

Configure HTTPS With the Elastic Beanstalk Console

To configure HTTPS for your load balancer with the Elastic Beanstalk console

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, choose the environment that you want to modify.



My First Elastic Beanstalk Application

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the console navigation pane, choose **Configuration**.
4. In the **Network Tier** section, choose the gear icon next to **Load Balancing**.
5. Update the **Load Balancer** section with the following information and choose **Apply**.

Listener port

`OFF`
: The load balancer does not listen to HTTP requests.

`80`
: The load balancer listens to HTTP requests.

Protocol (Listener port)
: `HTTP`

Secure listener port
: `443` or `8443`.

> **Note**
> If you are using Amazon VPC with Elastic Beanstalk, you must set this to `443`.

Protocol (Secure listener port)
: `HTTPS`

SSL certificate ID
: Select the certificate you uploaded to IAM.

It will take a few minutes to update your Elastic Beanstalk environment. Once your environment is Green and Ready, enter the HTTPS address for your application, such as `https://www.example.com`, in your web browser to verify that it is working with HTTPS. For instructions on how to check your environment status, see Health Colors (p. 247).

> **Note**
> If you used a self-signed certificate, your web browser does not recognize you as a CA, so you will see a warning message asking you if you want to proceed to the website. Choose to proceed, and then you can view your application.

Creating a Custom Amazon Machine Image (AMI)

When you create an AWS Elastic Beanstalk environment, you can specify an Amazon Machine Image (AMI) to use instead of the standard Elastic Beanstalk AMI included in your platform configuration's solution stack. A custom AMI can improve provisioning times when instances are launched in your environment if you need to install a lot of software that isn't included in the standard AMIs.

Using .ebextensions (p. 99) is great for configuring and customizing your environment quickly and consistently. Applying configurations, however, can start to take a long time during environment creation and updates. If you do a lot of server configuration in `.ebextensions`, you can reduce this time by making a custom AMI that already has the software and configuration that you need.

A custom AMI also allows you to make changes to low level components, such as the Linux kernel, that are difficult to implement or take a long time to apply in `.ebextensions`. You can base your custom AMI off of an Elastic Beanstalk image, or make a custom image from standard Amazon Linux AMIs.

To create a custom AMI, launch the base Elastic Beanstalk, Amazon Linux or other community AMI in Amazon EC2, customize the software and configuration to your needs, and then stop the instance and save an AMI from it.

> **Note**
> Do not create an AMI from an instance that was launched in an Elastic Beanstalk environment. Elastic Beanstalk makes changes to instances during provisioning that can cause issues in the saved AMI. Saving an image from an instance in an Elastic Beanstalk environment will also bake in the version of your application that was deployed to the instance.

If you create a custom AMI from an Amazon Linux AMI instead of Elastic Beanstalk's, provisioning time will increase because Elastic Beanstalk must install its own software components to support features such as Enhanced Health Reporting (p. 250). We recommend basing your custom AMI off of an Elastic Beanstalk AMI whenever possible.

To identify the base Elastic Beanstalk AMI

1. Open the Elastic Beanstalk Management Console.
2. Create an Elastic Beanstalk environment running your application. For more information on how to launch an Elastic Beanstalk application, go to the Getting Started Using Elastic Beanstalk (p. 4).
3. Navigate to the management console (p. 51) for your environment.
4. Choose **Configuration**
5. Next to **Instances**, click ⚙.
6. Note the value in the **Custom AMI ID** box.

You can also create a custom AMI from an Amazon Linux AMI or any community AMI.

To create a custom AMI

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Click **Launch Instance**.
3. Click **Community AMIs**
4. Enter the AMI ID of the Elastic Beanstalk or community AMI that you will customize to create a custom AMI and press **Enter**.
5. Click **Select** to select the AMI.
6. Select an instance type and click **Next: Configure Instance Details**.
7. Expand the **Advanced Details** section and paste the following text in the **User Data** field:

   ```
   #cloud-config
   repo_releasever: repository version number
   repo_upgrade: none
   ```

   The *repository version number* is the year and month version in the AMI name. For example, AMIs based on the March 2015 release of Amazon Linux have a repository version number `2015.03`. For Elastic Beanstalk image, this matches the date shown in the solution stack name for your platform configuration (p. 24).

   > **Note**
   > These settings configure the lock-on-launch feature, which causes the AMI to use a fixed, specific repository version when it launches, and disables the automatic installation of security updates. Both are required to use a custom AMI with Elastic Beanstalk.

8. Proceed through the wizard to launch the EC2 instance. When prompted, select an SSH key that you have access to so that you can use SSH to connect to the instance for the next steps.

   > **Note**
   > For additional information on how to launch an Amazon EC2 instance, go to Running an Instance in the *Amazon Elastic Compute Cloud User Guide*.

9. Connect to the instance. For more information on connecting to an Amazon EC2 instance, go to Connecting to Instances in the *Amazon Elastic Compute Cloud User Guide*.
10. After customizing a Windows instance, you need to run the EC2Config service Sysprep. For information about EC2Config, go to Configuring a Windows Instance Using the EC2Config Service.
11. If you are using an AMI with Apache and Tomcat, you will need to perform your customizations.

    Apache and Tomcat are not automatically started when you manually launch the Elastic Beanstalk AMI using the Amazon EC2 tab on the AWS Management Console. Enter the following commands at your Amazon EC2 instance's command prompt to start Apache and Tomcat.

    ```

    sudo -s
    cd /etc/init.d
    ./httpd start
    ./tomcat7 start
    ```

12. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
13. Stop the EC2 instance.
14. Select the stopped instance that you've just modified and choose **Create Image (EBS AMI)** from the **Instance Actions** menu.
15. To avoid incurring additional AWS charges, terminate the Amazon EC2 instance. For instructions on how to terminate an instance, go to Terminate Your Instance in the *Amazon Elastic Compute Cloud User Guide*.

16. To use your custom AMI, specify your custom AMI ID in the **Custom AMI ID** text box in the Elastic Beanstalk **Edit Configuration** dialog box. Existing instances will be replaced with new instances launched from the new custom AMI.

Constructing a Launch Now URL

You can construct a custom uniform resource locator (URL) so that anyone can quickly deploy and run a predetermined web application in Elastic Beanstalk. This URL is called a Launch Now URL. You might need a Launch Now URL, for example, to demonstrate a web application that is built to run on Elastic Beanstalk. With Launch Now URL, you can use parameters to add the required information to the Create Application wizard in advance. When you do, anyone can use the URL link to launch an Elastic Beanstalk environment with your web application source in just a few clicks. This means users don't need to manually upload or specify the location of the application source bundle or provide any additional input to the wizard.

A Launch Now URL gives Elastic Beanstalk the minimum information required to create an application: the application name, solution stack, instance type, and environment type. Elastic Beanstalk uses default values for other configuration details that are not explicitly specified in your custom Launch Now URL.

A Launch Now URL uses standard URL syntax. For more information, see RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax.

URL Parameters

The URL must contain the following parameters, which are case-sensitive:

- **region** – Specify an AWS region. For a list of regions supported by Elastic Beanstalk, see AWS Elastic Beanstalk in the *Amazon Web Services General Reference*.
- **applicationName** – Specify the name of your application. Elastic Beanstalk displays the application name in the AWS Management Console to distinguish it from other applications. By default, the application name also forms the basis of the environment name and environment URL.
- **solutionStackName** – Specify the platform and version that will be used for the environment. For more information, see Supported Platforms (p. 24).

A Launch Now URL can optionally contain the following parameters. If you do not include the optional parameters in your Launch Now URL, Elastic Beanstalk uses default values to create and run your application. When you do not include the **sourceBundleUrl** parameter, Elastic Beanstalk uses the default sample application for the specified **solutionStackName**.

- **sourceBundleUrl** – Specify the location of your web application source bundle in URL format. For example, if you uploaded your source bundle to an Amazon Simple Storage Service bucket, you might specify the value of the **sourceBundleUrl** parameter as `http://s3.amazonaws.com/mybucket/myobject`.

  Note
  You can specify the value of the **sourceBundleUrl** parameter as an HTTP URL, but the user's web browser will convert characters as needed by applying HTML URL encoding.
- **environmentType** – Specify whether the environment is load balancing and autoscaling or just a single instance. For more information, see Environment Types (p. 68). You can specify either `LoadBalancing` or `SingleInstance` as the parameter value.
- **tierName** – Specify whether the environment supports a web application that processes web requests or a web application that runs background jobs. For more information, see Worker Environments (p. 69). You can specify either `WebServer` or `Worker`,

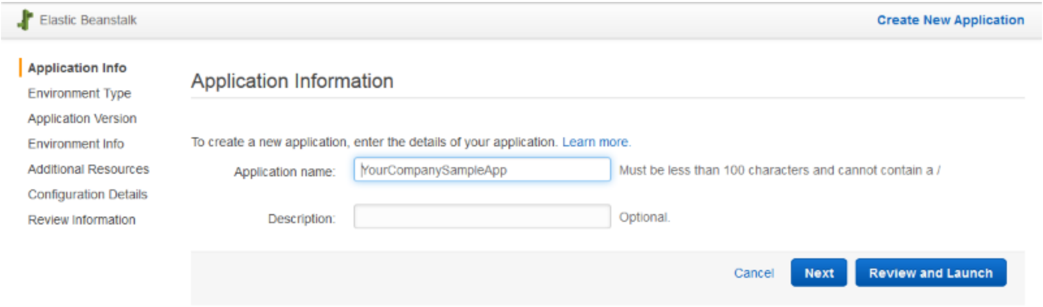
- **instanceType** – Specify a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. To see the instance types that are available in your Elastic Beanstalk region, see InstanceType in the topic Configuration Options (p. 103). To see the detailed specifications for each Amazon EC2 instance type, see Instance Types.
- **withVpc** – Specify whether to create the environment in an Amazon VPC. You can specify either `true` or `false`. For more information about using Elastic Beanstalk with Amazon VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).
- **withRds** – Specify whether to create an Amazon RDS database instance with this environment. For more information, see Using Elastic Beanstalk with Amazon RDS (p. 294). You can specify either `true` or `false`.
- **rdsDBEngine** – Specify the database engine that you want to use for your Amazon EC2 instances in this environment. You can specify `mysql`, `oracle-sel`, `sqlserver-ex`, `sqlserver-web`, or `sqlserver-se`. The default value is `mysql`.
- **rdsDBAllocatedStorage** – Specify the allocated database storage size in gigabytes. You can specify the following values:
  - **MySQL** – `5` to `1024`. The default is `5`.
  - **Oracle** – `10` to `1024`. The default is `10`.
  - **Microsoft SQL Server Express Edition** – `30`.
  - **Microsoft SQL Server Web Edition** – `30`.
  - **Microsoft SQL Server Standard Edition** – `200`.
- **rdsDBInstanceClass** – Specify the database instance type. The default value is `db.t1.micro`. For a list of database instance classes supported by Amazon RDS, see DB Instance Class in the *Amazon Relational Database Service User Guide*.
- **rdsMultiAZDatabase** – Specify whether Elastic Beanstalk needs to create the database instance across multiple Availability Zones. You can specify either `true` or `false`. For more information about multiple Availability Zone deployments with Amazon RDS, go to Regions and Availability Zones in the *Amazon Relational Database Service User Guide*.
- **rdsDBDeletionPolicy** – Specify whether to delete or snapshot the database instance on environment termination. You can specify either `Delete` or `Snapshot`.

Example

The following is an example Launch Now URL. After you construct your own, you can give it to your users. For example, you might want to embed the URL on a web page or in training materials. When users create an application using the Launch Now URL, the Elastic Beanstalk Create an Application wizard requires no additional input.

[illegible]

When users click a Launch Now URL, Elastic Beanstalk displays a page similar to the following.



Application Information

To create a new application, enter the details of your application. Learn more.

Application name: YourCompanySampleApp Must be less than 100 characters and cannot contain a /

Description: Optional.

To use the Launch Now URL

1. Click the Launch Now URL.
2. When the Elastic Beanstalk console opens, on the **Application Info** page, click **Review and Launch** to view the settings Elastic Beanstalk will use to create the application and launch the environment in which the application runs.
3. On the **Review** page, click **Launch** to create the application.

Using Custom Domains with Elastic Beanstalk

You can use a custom domain for your Elastic Beanstalk application, such as example.com.

There are two ways you can use custom domains with Elastic Beanstalk:

- Create a CNAME with your Domain Name System (DNS) provider.
- Use Amazon Route 53 to create an alias record.

Using a Domain Hosted by a Third Party

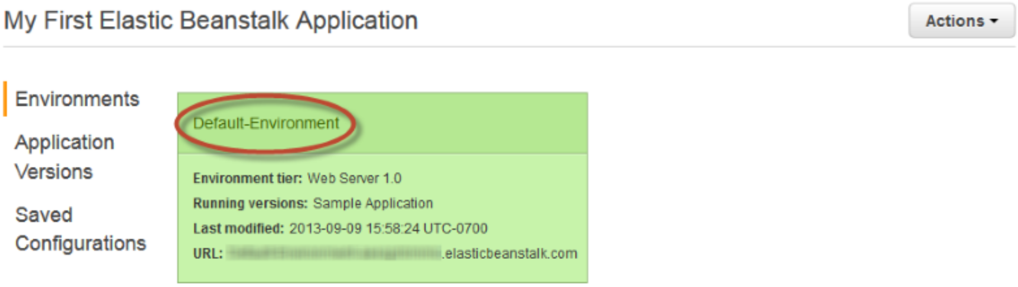
If you host a domain name with a third party, you can use that domain name for your Elastic Beanstalk application. Because the IP address of the elastic load balancer is not fixed, you should not associate your domain name with your load balancer's IP address. Instead, you should create a CNAME with your DNS provider, and then map the CNAME to your Elastic Beanstalk URL. Make sure you also forward your unqualified domain name, (i.e., example.com), to your qualified domain name (i.e., www.example.com) so that when users type **example.com** and **www.example.com**, they both map to your Elastic Beanstalk application. Check your DNS provider's instructions for mapping your domain name to your Elastic Beanstalk URL.

For example, if your Elastic Beanstalk URL is **http://foobar.elasticbeanstalk.com**, then you would do the following high level steps:

1. Create a CNAME for your www record that maps to foobar.elasticbeanstalk.com.
2. Forward example.com to www.example.com.

To view the Elastic Beanstalk URL for your application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select the region that includes the environment that you want to work with.
3. From the Elastic Beanstalk console applications page, click the name of the environment that you want to view the URL for.

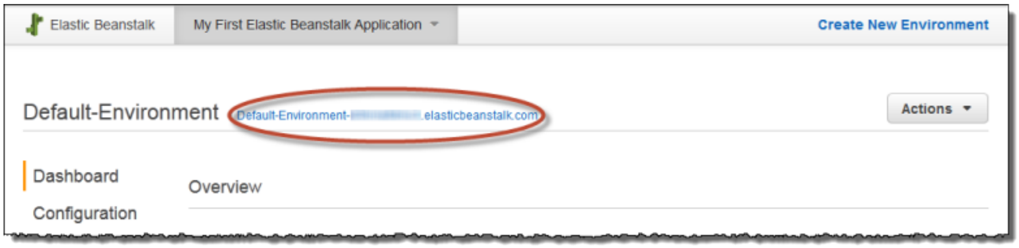


My First Elastic Beanstalk Application

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

In the environment dashboard, the URL for the environment is displayed next to the name of the environment.



Default-Environment Default-Environment-[redacted].elasticbeanstalk.com

Using a Domain Hosted by Amazon Route 53

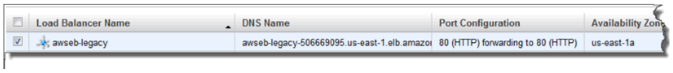
Amazon Route 53 is a highly available and scalable DNS web service. If you host a domain name using Amazon Route 53, you can use that domain name for your Elastic Beanstalk application. Given how the DNS protocol works, there is no way to refer your elastic load balancer or Amazon EC2 instance from the root (also known as the apex) of the domain. For instance, you can create a DNS entry that maps http://www.example.com to an elastic load balancer or EC2 instance, but you cannot do the same for http://example.com. Amazon Route 53 enables you to map the apex (such as example.com) of a hosted zone to your elastic load balancer or EC2 instance using an alias record. When Amazon Route 53 encounters an alias record, it looks up the records associated with the target DNS name in the alias, and returns the IP addresses from that name. The following procedures walk you through mapping your root domain and subdomains to your elastic load balancer or EC2 instance in your Elastic Beanstalk environment.

To map your root domain and subdomains to your Elastic Load Balancing load balancer

1. Follow the Amazon Route 53 Getting Started Guide instructions to sign up for Route 53, create a hosted zone, and then update your name server records with your registrar.
2. Get the value of the hosted zone ID for your load balancer.

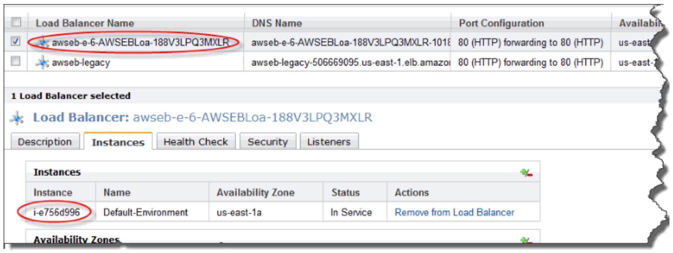
   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
   b. From the region list, select a region.
   c. In the **Navigation** pane, click **Load Balancers** .
   d. Select the load balancer associated with your Elastic Beanstalk application. Depending on your container type, the load balancer will appear in one of the following formats:

      - Legacy container — **awseb-<*`your app environment name`*>**



☐	Load Balancer Name	DNS Name	Port Configuration	Availability Zon
☑	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-1a

- Nonlegacy container — contains **awseb** in the load balancer name. To verify you have the correct load balancer for your environment, check the instance name in the **Instances** tab. The instance name should be your environment name.



☐	Load Balancer Name	DNS Name	Port Configuration	Availabi
☑	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR-101	80 (HTTP) forwarding to 80 (HTTP)	us-east
☐	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-

1 Load Balancer selected

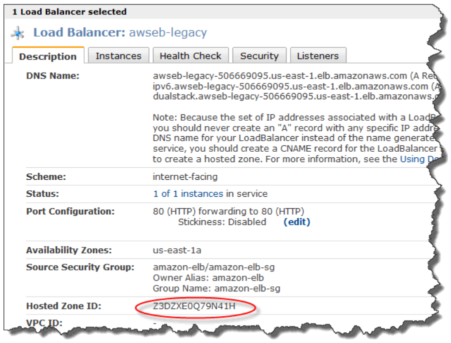
Load Balancer: awseb-e-6-AWSEBLoa-188V3LPQ3MXLR

Instances

Instance	Name	Availability Zone	Status	Actions
i-e756d996	Default-Environment	us-east-1a	In Service	Remove from Load Balancer

Availability Zones

Your hosted ID will appear in the **Load Balancer** details pane on the **Description** tab. Make a note of your hosted ID.



1 Load Balancer selected

Load Balancer: awseb-legacy

DNS Name:	awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A Re ipv6.awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A dualstack.awseb-legacy-506669095.us-east-1.elb.amazonaws. Note: Because the set of IP addresses associated with a LoadB you should never create an "A" record with any specific IP addr DNS name for your LoadBalancer instead of the name generate service, you should create a CNAME record for the LoadBalancer to create a hosted zone. For more information, see the Using D
Scheme:	internet-facing
Status:	1 of 1 instances in service
Port Configuration:	80 (HTTP) forwarding to 80 (HTTP) Stickiness: Disabled (edit)
Availability Zones:	us-east-1a
Source Security Group:	amazon-elb/amazon-elb-sg Owner Alias: amazon-elb Group Name: amazon-elb-sg
Hosted Zone ID:	Z3DZXE0Q79N41H
VPC ID:	-

3. Create alias resource record sets in your hosted zone for your root domain and subdomain. For instructions, go to How to Create an Alias Resource Record Set in the *Amazon Route 53 Developer Guide*.
4. Your root domain and subdomain are now mapped to your Elastic Beanstalk elastic load balancer. Type your domain name in your web browser to verify it worked.

If you rebuild your environment, launch a new environment, or swap your environment URL, you will need to map your root domain to the load balancer in your new environment.

To map your root domain and subdomains to a single Amazon EC2 instance

1. Follow the instructions to sign up for Route 53 and create a hosted zone in Getting Started: Creating a Domain that Uses Route 53 in the *Amazon Route 53 Developer Guide*.
2. Get the Elastic IP for your EC2 instance.
    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
    b. From the region list, select a region.
    c. In the **Navigation** column, click **Instances**.
    d. Select the instance associated with your Elastic Beanstalk application.

Your Elastic IP address will appear in the details pane on the **Description** tab. Make a note of your Elastic IP address.

3. Create resource record sets in your hosted zone for your root domain and subdomains. For a single EC2 instance, create an A record by specifying the Elastic IP address of the instance. For instructions, see Step 4: Create Resource Record Sets in your Route 53 Hosted Zone in the *Amazon Route 53 Developer Guide*.
4. Update your registrar's name server records. For instructions, see Step 5: Update the Registrar's Name Server Records in the *Amazon Route 53 Developer Guide*.
5. Your root domain and subdomain are now mapped to the Elastic IP of your EC2 instance in Elastic Beanstalk. Type your domain name in your web browser to verify it worked.

Monitoring an Environment

When you are running a production website, it is important to know that your application is available and responding to requests. To assist with monitoring your application’s responsiveness, Elastic Beanstalk provides features that monitor statistics about your application and create alerts that trigger when thresholds are exceeded.

Topics

Monitoring Environment Health in the AWS Management Console

You can access operational information about your application from the AWS Management Console at http://console.aws.amazon.com/elasticbeanstalk.

The AWS Management Console displays your environment's status and application health at a glance. In the Elastic Beanstalk console applications page, each environment is color-coded to indicate an environment's status.

To monitor an environment in the AWS Management Console

1. Navigate to the Environment Management Console (p. 51) for your environment
2. In the left navigation, click **Monitoring**.

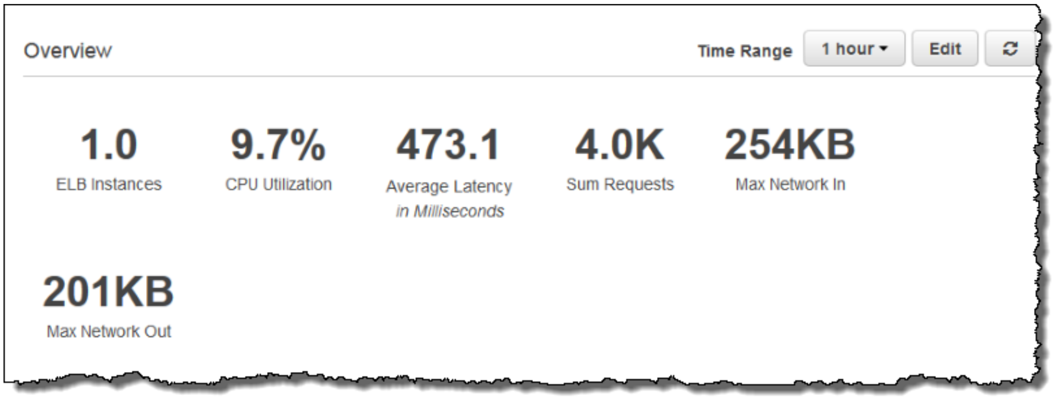
   The Monitoring page shows you overall statistics about your environment, such as CPU utilization and average latency. In addition to the overall statistics, you can view monitoring graphs that show resource usage over time. You can click any of the graphs to view more detailed information.

Note

By default, only basic CloudWatch metrics are enabled, which return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by editing your environment's configuration settings.

Overview

An overview of the environment's health is shown near the top of the screen.

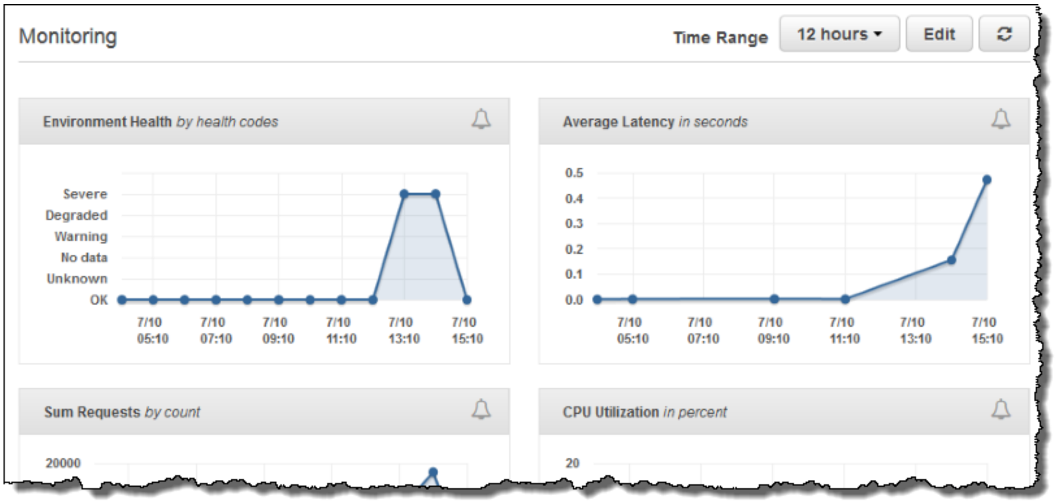


1.0	9.7%	473.1	4.0K	254KB
ELB Instances	CPU Utilization	Average Latency *in Milliseconds*	Sum Requests	Max Network In
201KB				
Max Network Out				

The overview panel shows a customizable summary of the activity in your environment over the last hour. Click the **Time Range** drop-down and select a different length of time to view information for a time period of five minutes to one day.

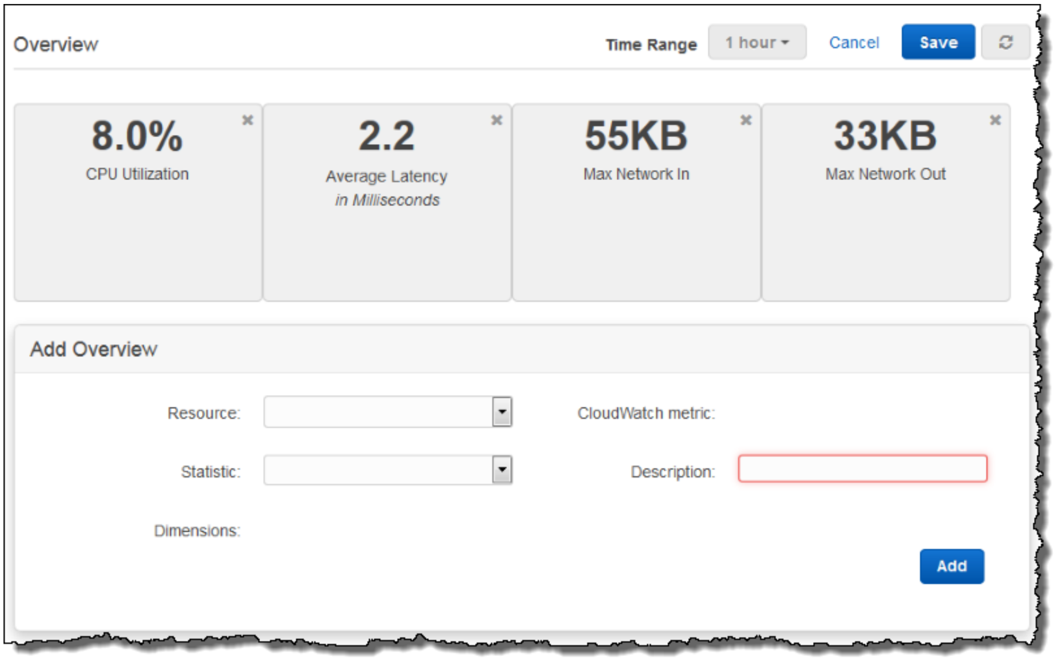
Monitoring Graphs

Below the overview are graphs that show data about overall environment health over the last twelve hours. Click the **Time Range** drop-down and select a different length of time to view information for a time period of three hours and two weeks.



Customizing the Monitoring Console

Click **Edit** next to either monitoring section to customize the information shown.



8.0% CPU Utilization

2.2 Average Latency *in Milliseconds*

55KB Max Network In

33KB Max Network Out

To remove any of the existing items, click the ✖ in the top right corner.

To add an overview or graph

1. Click **Edit** in the **Overview** or **Monitoring** section.
2. Select a **Resource**. The supported resources are your environment's Auto Scaling group, Elastic Load Balancing load balancer, and the environment itself.
3. Select a **CloudWatch metric** for the resource. See Publishing Amazon CloudWatch Custom Metrics for an Environment (p. 263) for a full list of supported metrics.
4. Select a **Statistic**. The default statistic is the average value of the selected cloudwatch metric during the time range (overview) or between plot points (graph).
5. Enter a **Description**. The description is the label for the item shown in the monitoring console.
6. Click **Add**.
7. Repeat the previous steps to add more items or click **Save** to finish modifying the panel.

For more information about metrics and dimensions for each resource, see Amazon CloudWatch Metrics, Namespaces, and Dimensions Reference in the *Amazon CloudWatch Developer Guide*.

Elastic Load Balancing and Amazon EC2 metrics are enabled for all environments.

With enhanced health, the EnvironmentHealth metric is enabled and a graph is added to the monitoring console automatically. Additional metrics become available for use in the monitoring console when you enabled them in the environment configuration.

> **Note**
> When you enable additional CloudWatch metrics for your environment, it takes a few minutes for them to start being reported and appear in the list of metrics that you use to add graphs and overview stats.

See Publishing Amazon CloudWatch Custom Metrics for an Environment (p. 263) for a list of available enhanced health metrics.

Basic Health Reporting

AWS Elastic Beanstalk uses information from multiple sources to determine if your environment is available and processing requests from the Internet. An environment's health is represented by one of four colors, which is displayed in the environment dashboard (p. 51), and is also available from the DescribeEnvironments API and by calling `eb status` with the EB CLI (p. 384).

Prior to version 2 Linux platform configurations, the only health reporting system was basic health. The basic health reporting system provides information about the health of instances in an Elastic Beanstalk environment based on health checks performed by Elastic Load Balancing for load balanced environments or Amazon Elastic Compute Cloud for single instance environments.

In addition to checking the health of your EC2 instances, Elastic Beanstalk also monitors the other resources in your environment and reports missing or incorrectly configured resources that can cause your environment to become unavailable to users.

Metrics gathered by the resources in your environment is published to Amazon CloudWatch in five minute intervals. This includes operating system metrics from EC2, request metrics from Elastic Load Balancing. You can view graphs based on these CloudWatch metrics on the Monitoring page (p. 244) of the environment console. For basic health, these metrics are not used to determine an environment's health.

Topics

Health Colors

Elastic Beanstalk reports the health of a web server environment depending on how the application running in it responds to the health check. Elastic Beanstalk uses one of four colors to describe status, as shown in the following table:

Color	Description
Gray	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Some requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consistently failing.

These descriptions only apply to environments using basic health reporting. See Health Colors and Statuses (p. 260) for details related to enhanced health.

Elastic Load Balancing Health Check

In a load balanced environment, Elastic Load Balancing sends a request to each instance in an environment every 30 seconds to confirm that instances are healthy. By default, the load balancer is configured to open a TCP connection on port 80. If the instance acknowledges the connection, it is considered healthy.

You can choose to override this setting by specifying an existing resource in your application. If you specify a path, such as `/health`, the health check URL is set to `http:80/health`. The health check URL should be set to a path that is always served by your application. If it is set to a static page that is served or cached by the web server in front of your application, health checks will not reveal issues with the application server or web container. For instructions on modifying your health check URL, see Health Checks (p. 190).

If a health check URL is configured, Elastic Load Balancing expects a GET request that it sends to return a response of `200 OK`. The application fails the health check if it fails to respond within 5 seconds or if it responds with any other HTTP status code. After 5 consecutive health check failures, Elastic Load Balancing takes the instance out of service.

For more information about Elastic Load Balancing health checks, see Health Check in the *Elastic Load Balancing Developer Guide*.

> **Note**
> Congfiguring a health check URL does not change the health check behavior of an environment's Auto Scaling group. An unhealthy instance is removed from the load balancer, but is not automatically replaced by Auto Scaling unless you configure Auto Scaling to use the Elastic Load Balancing health check as a basis for replacing instances. To configure Auto Scaling to replace instances that fail an Elastic Load Balancing health check, see Auto Scaling Health Check Setting (p. 185).

Single Instance Environment Health Check

In a single instance environment, Elastic Beanstalk determines the instance's health by monitoring its Amazon EC2 instance status. Elastic Load Balancing health settings, including HTTP health check URLs, cannot be used in a single instance environment.

For more information on Amazon EC2 instance status checks, see Monitoring Instances with Status Checks in the *Amazon EC2 User Guide for Linux Instances*.

Additional Checks

In addition to Elastic Load Balancing health checks, Elastic Beanstalk monitors resources in your environment and changes health status to red if they fail to deploy, are not configured correctly, or become unavailable. These checks confirm that:

- The environment's Auto Scalinggroup is available and has a minimum of at least one instance.
- The environment's security group is available and is configured to allow incoming traffic on port 80.
- The environment CNAME exists and is pointing to the right load balancer.
- In a worker environment, the Amazon Simple Queue Service (Amazon SQS) queue is being polled at least once every three minutes.

Amazon CloudWatch Metrics

With basic health reporting, the Elastic Beanstalk service does not publish any metrics to Amazon CloudWatch. The CloudWatch metrics used to produce graphs on the Monitoring page (p. 244) of the environment console are published by the resources in your environment.

For example, EC2 publishes the following metrics for the instances in your environment's Auto Scaling group:

`CPUUtilization`
: Percentage of compute units currently in use.

`DiskReadBytes`, `DiskReadOps`, `DiskWriteBytes`, `DiskWriteOps`
: Number of bytes read and written, and number of read and write operations.

`NetworkIn`, `NetworkOut`
: Number of bytes sent and received.

Elastic Load Balancing publishes the following metrics for your environment's load balancer:

`BackendConnectionErrors`
: Number of connection failures between the load balancer and environment instances.

`HTTPCode_Backend_2XX`, `HTTPCode_Backend_4XX`
: Number of successful (2XX) and client error (4XX) response codes generated by instances in your environment.

`Latency`
: Number of seconds between when the load balancer relays a request to an instance and when the response is received.

`RequestCount`
: Number of completed requests.

These lists are not comprehensive. For a full list of metrics that can be reported for these resources, see the following topics in the Amazon CloudWatch Developer Guide:

Metrics

Namespace	Topic
AWS::ElasticLoadBalancing::LoadBalancer	Elastic Load Balancing Metrics and Resources
AWS::AutoScaling::AutoScalingGroup	Amazon Elastic Compute Cloud Metrics and Resources
AWS::SQS::Queue	Amazon SQS Metrics and Resources
AWS::RDS::DBInstance	Amazon RDS Dimensions and Metrics

Worker Environment Health Metric

For worker environments only, the SQS daemon publishes a custom metric for environment health to CloudWatch, where a value of 1 is Green. You can review the CloudWatch health metric data in your account using the `ElasticBeanstalk/SQSD` namespace. The metric dimension is `EnvironmentName`, and the metric name is `Health`. All instances publish their metrics to the same namespace.

To enable the daemon to publish metrics, you must grant the appropriate permissions to the environment's instance profile. For more information, see Granting IAM Role Permissions for Worker Environment Tiers (p. 333).

Troubleshooting Health Issues

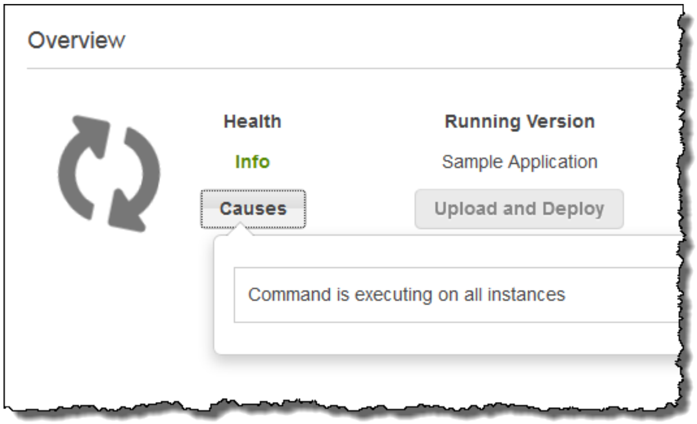
If the health of your environment changes to red, try the following:

- Review recent environment events (p. 531). Messages from Elastic Beanstalk about deployment, load, and configuration issues often appear here.
- Pull logs (p. 282) to view recent log file entries. Web server logs contain information about incoming requests and errors.
- Connect to an instance (p. 280) and check system resources.
- Roll back (p. 78) to a previous, working version of the application.
- Undo recent configuration changes or restore a saved configuration (p. 101).
- Deploy a new environment. If it appears healthy, perform a CNAME swap (p. 83) to route traffic to the new environment and continue to debug the old one.

Enhanced Health Reporting and Monitoring

Enhanced health reporting is a feature that you can enable on your environment to allow AWS Elastic Beanstalk to gather additional information about resources in your environment. Elastic Beanstalk analyzes the information gathered to provide a better picture of overall environment health and aid in the identification of issues that can cause your application to become unavailable.

In addition to changes in how health color works, enhanced health adds a *status* descriptor that provides an indicator of the severity of issues observed when an environment is yellow or red. When additional information about an environment's health is available, Elastic Beanstalk provides a message indicating the *cause*.



Overview

Health: Info

Running Version: Sample Application

Command is executing on all instances

Cause shown in the environment dashboard for a healthy (green) environment that is being updated.

To provide detailed health information about the EC2 instances running in your environment, Elastic Beanstalk includes a health agent in the Amazon Machine Image (AMI) for each platform configuration that supports enhanced health. The health agent monitors web server logs and system metrics and relays them to the Elastic Beanstalk service. Elastic Beanstalk analyzes these metrics along with data from Elastic Load Balancing and Auto Scaling to provide an overall picture of an environment's health.

In addition to collecting and presenting information about your environment's resources, Elastic Beanstalk monitors the resources in your environment for several error conditions and provides notifications to help you avoid failures and resolve configuration issues.

You can view health status in real time by using the environment dashboard (p. 51) in the AWS Management Console or the `eb health` command in the EB command line interface (p. 384) (CLI). To record and track environment and instance health over time, you can configure your environment to publish the information gathered by Elastic Beanstalk for enhanced health reporting to Amazon CloudWatch as custom metrics. CloudWatch charges for custom metrics apply to all metrics other than `EnvironmentHealth`, which is free of charge.

Enhanced health reporting requires a version 2 or newer platform configuration (p. 24) and is supported by all platforms except Windows Server with IIS. In order to monitor resources and publish metrics, your environment must have both an instance profile and service (p. 250) role. The Multicontainer Docker configuration does not include a web server by default but can be used with enhanced health reporting if you configure your web server to provide logs in the proper format (p. 269).

The first time you create an environment with a version 2 platform configuration in the AWS Management Console, Elastic Beanstalk prompts you to create the required roles and enables enhanced health reporting by default. Continue reading for details on how enhanced health reporting works, or go to Enabling and Disabling AWS Elastic Beanstalk Enhanced Health Reporting (p. 251) to get started using it right away.

Enabling and Disabling AWS Elastic Beanstalk Enhanced Health Reporting

New environments created with the latest platform versions (p. 24) include the AWS Elastic Beanstalk health agent (p. 273), which supports enhanced health reporting. If you create your environment in the AWS Management Console or with the EB CLI, you are prompted to enable enhanced health reporting and to create the required AWS Identity and Access Management (IAM) roles. You can also set your health reporting preference in your application's source code using `.ebextensions`.

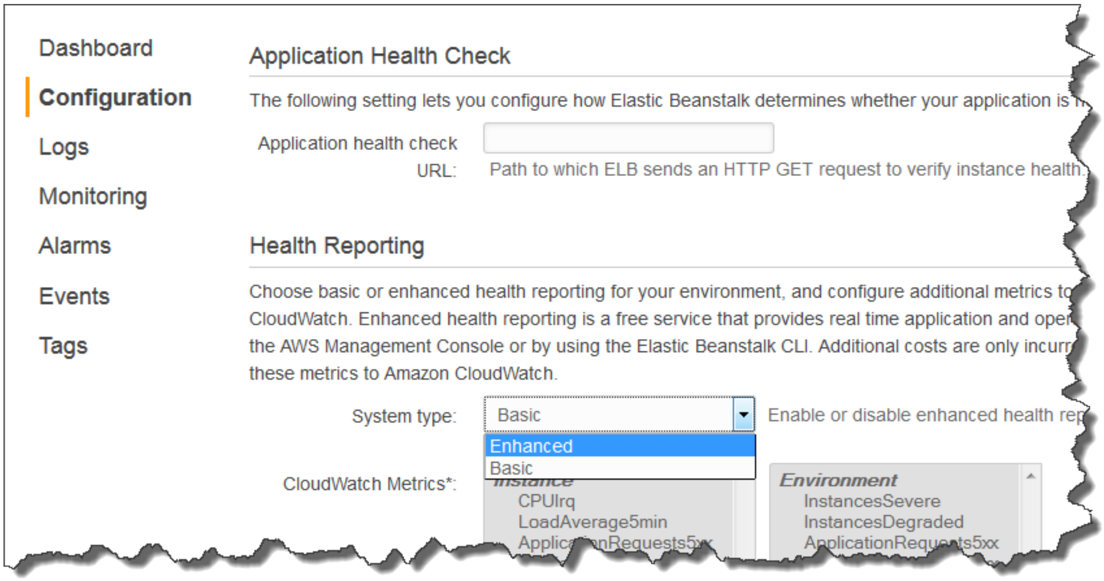
See Launching an Environment with a Sample Application in the AWS Management Console (p. 50) for instructions on creating your first environment.

Topics

Enabling and Disabling Enhanced Health Reporting with the AWS Management Console

To enable or disable enhanced health reporting in a running environment with the AWS Management Console

1. Navigate to the management console (p. 51) for your environment.
2. In the left navigation panel, choose **Configuration**.
3. On the **Health** panel under **Web Tier**, choose ⚙ (edit).
4. Under **Health Reporting**, for **System type**, choose **Enhanced** to enable enhanced health reporting or **Basic** to disable enhanced health reporting.



Application Health Check

The following setting lets you configure how Elastic Beanstalk determines whether your application is h

Application health check URL: Path to which ELB sends an HTTP GET request to verify instance health.

Health Reporting

Choose basic or enhanced health reporting for your environment, and configure additional metrics to CloudWatch. Enhanced health reporting is a free service that provides real time application and oper the AWS Management Console or by using the Elastic Beanstalk CLI. Additional costs are only incurr these metrics to Amazon CloudWatch.

System type: Basic — Enable or disable enhanced health rep

CloudWatch Metrics*:

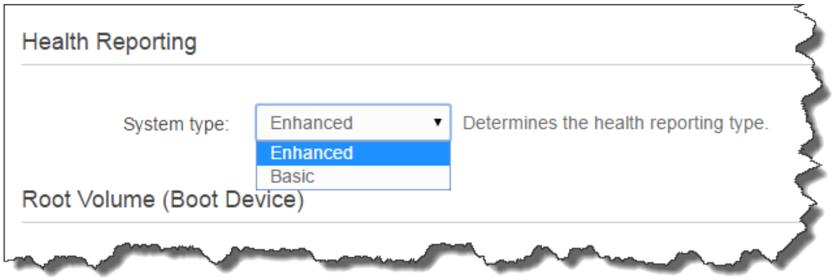
Health reporting options in the environment management console.

5. Choose **Apply**.

The Elastic Beanstalk console defaults to enhanced health reporting when you create a new environment with a version 2 platform configuration. You can disable enhanced health reporting by changing the health reporting option during environment creation.

To disable enhanced health reporting when creating an environment using the AWS Management Console

1. Open the Elastic Beanstalk Management Console.
2. Create an application (p. 34) or select an existing one.
3. Create an environment (p. 55). On the **Configuration Details** page, choose **Basic** for the health reporting type under **Health Reporting**.


Health Reporting

System type: Enhanced — Determines the health reporting type.

Root Volume (Boot Device)

Enabling Enhanced Health Reporting with the EB CLI

When you create a new environment with the `eb create` command, the EB CLI guides you through the necessary configuration and enables enhanced health reporting by default.

When creating an environment with the EB CLI, you must assign a service role and instance profile to the environment. Unless you specify an instance profile (p. 20) with the `--instance-profile` option, the EB CLI will create one named `aws-elasticbeanstalk-ec2-role`, give it the appropriate permissions, and assign it to your environment. If you don't already have a service role (p. 19) named `aws-elasticbeanstalk-service-role`, the EB CLI will prompt you to create one, as shown in the following example:

```
$ eb create --cname my-env --instance_type t2.micro
Enter Environment Name
(default is MyFirstElasticBeanstal): my-env

2.0+ Platforms require a service role. We will attempt to create one for you.
You can specify your own role using the --service-role option.
Type "view" to see the policy, or just press ENTER to continue: Enter
```

If you already have the default service role, the EB CLI applies it to your environment automatically. You can override this behavior by specifying a different service role by name with the `--service-role` option.

If you have an environment running with basic health reporting on a version 2 platform configuration and want to switch to enhanced health, follow these steps:

To enable enhanced health on a running environment using the EB CLI (p. 384)

1. Use the `eb config` command to open the configuration file in the default text editor:

   ```
   ~/project$ eb config
   ```

2. Locate the `aws:elasticbeanstalk:environment` namespace in the settings section. Ensure that the value of `ServiceRole` is not null and that it matches the name of your service role (p. 19).

   ```
     aws:elasticbeanstalk:environment:
       EnvironmentType: LoadBalanced
       ServiceRole: aws-elasticbeanstalk-service-role
   ```

3. Under the `aws:elasticbeanstalk:healthreporting:system:` namespace, change the value of `SystemType` to **enhanced**.

   ```
     aws:elasticbeanstalk:healthreporting:system:
       SystemType: enhanced
   ```

4. Save the configuration file and close the text editor.
5. The EB CLI starts an environment update to apply your configuration changes. Wait for the operation to complete or press **Ctrl**+**C** to exit safely.

   ```
   ~/project$ eb config
   Printing Status:
   INFO: Environment update is starting.
   INFO: Health reporting type changed to ENHANCED.
   INFO: Updating environment no-role-test's configuration settings.
   ```

Enabling Enhanced Health Reporting with .ebextensions

To enable enahanced health reporting in a configuration file in your source bundle, use `.ebextensions`.

To enable enhanced health reporting with .ebextensions

1. In your project folder, create a configuration file named `health.config` in the `.ebextensions` directory.
2. Add the following text to the configuration file:

   ```
   OptionSettings:
     aws:elasticbeanstalk:healthreporting:system:
       SystemType: enhanced
     aws:elasticbeanstalk:environment:
       ServiceRole: aws-elasticbeanstalk-service-role
   ```

   For the service role, use the name of a role configured to allow Elastic Beanstalk to monitor health. See Elastic Beanstalk Service Role (p. 19) for details.
3. Package the `.ebextensions` folder into your source bundle (p. 43) or deploy it with the EB CLI (p. 384).

To disable enhanced health reporting with .ebextensions

1. In your project folder, create a configuration file with an appropriate name, for example, `health.config`, in the .ebextensions directory.
2. Add the following text:

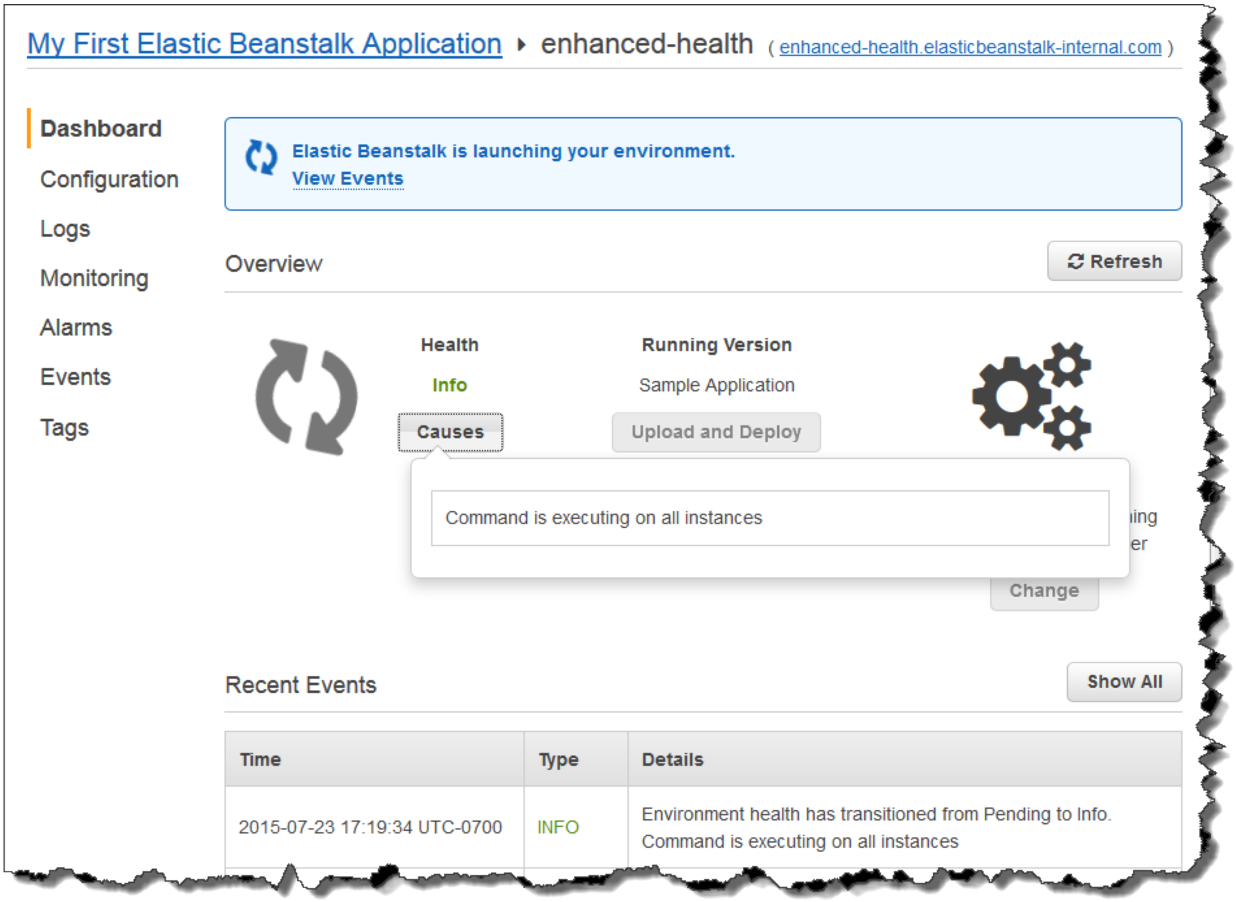
   ```
   OptionSettings:
     aws:elasticbeanstalk:healthreporting:system:
       SystemType: basic
   ```

3. Package the `.ebextensions` folder into your source bundle (p. 43) or deploy it with the EB CLI (p. 385).

Enhanced Health Monitoring with the Environment Management Console

When you have enabled enhanced health reporting, you can use the environment management console (p. 51) to find health information and options.

The **Overview** section of the **Dashboard** displays the health status (p. 260) of the environment and includes a **Causes** button. For information about the current health status, choose **Causes** .



Elastic Beanstalk is launching your environment.

Running Version: Sample Application

Command is executing on all instances

Time	Type	Details
2015-07-23 17:19:34 UTC-0700	INFO	Environment health has transitioned from Pending to Info. Command is executing on all instances

The **Monitoring** page of the console displays summary statistics and graphs for the custom Amazon CloudWatch metrics generated by the enhanced health reporting system. See Monitoring Environment Health in the AWS Management Console (p. 244) for instructions on adding graphs and stats to this page. For a complete list of the metrics generated by Elastic Beanstalk enhanced health reporting , see Publishing Amazon CloudWatch Custom Metrics for an Environment (p. 263).

Using the EB CLI to Monitor Environment Health

The EB command line interface (p. 384) (CLI) is a command line tool for managing AWS Elastic Beanstalk environments. You also can use the EB CLI to monitor your environment's health in real time and with more granularity than is currently available in the AWS Management Console

After installing (p. 385) and configuring (p. 390) the EB CLI, you can launch a new environment (p. 394) and deploy your code to it with the `eb create` command. If you already have an environment that you created in the AWS Management Console, you can attach the EB CLI to it by running `eb init` in a project folder and following the prompts (the project folder can be empty).

Important

Ensure that you are using the latest version of the EB CLI by running `pip install` with the `--upgrade` option:

```
$ sudo pip install --upgrade awsebcli
```

For complete EB CLI installation instructions, see Install the EB Command Line Interface (CLI) (p. 385).

To use the EB CLI to monitor your environment's health, you must first configure a local project folder by running `eb init` and following the prompts. For complete instructions, see Configure the EB CLI (p. 390).

If you already have an environment running in Elastic Beanstalk and want to use the EB CLI to monitor its health, attach it to use the existing environment by following these steps.

To attach the EB CLI to an existing environment

1. Open a command line terminal and navigate to your user folder.
2. Create and open a new folder for your environment.
3. Run the `eb init` command, and then choose the application and environment whose health you want to monitor. If you have only one environment running the application you choose, the EB CLI will select it automatically and you will not need to choose the environment, as shown in the following example:

```
~/project$ eb init
Select an application to use
1) elastic-beanstalk-example
2) [ Create new Application ]
(default is 2): 1
Select the default environment.
You can change this later by typing "eb use [environment_name]".
1) elasticBeanstalkEx2-env
2) elasticBeanstalkExa-env
(default is 1): 1
```

To monitor health by using the EB CLI

Topics

1. Open a command line and navigate to your project folder.
2. Run the `eb health` command to display the health status of the instances in your environment. In this example, there are five instances running in the environment:

```
~/project $ eb health
 elasticBeanstalkExa-env                                  Ok
         2015-07-08 23:13:20
WebServer
             Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  id             status     cause
    Overall      Ok
  i-d581497d     Ok
  i-d481497c     Ok
  i-136e00c0     Ok
  i-126e00c1     Ok
  i-8b2cf575     Ok

  id             r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90
 p75     p50     p10
    Overall      0.0         -      -      -      -         -        -
   -       -       -
  i-d581497d     0.0         -      -      -      -         -        -
   -       -       -
  i-d481497c     0.0         -      -      -      -         -        -
   -       -       -
  i-136e00c0     0.0         -      -      -      -         -        -
   -       -       -
  i-126e00c1     0.0         -      -      -      -         -        -
   -       -       -
  i-8b2cf575     0.0         -      -      -      -         -        -
   -       -       -

  id             az            running     load 1  load 5      user%  nice%
  system%  idle%   iowait%
  i-d581497d     us-east-1a    12 mins        0.0    0.03        0.2    0.0
      0.0   99.7       0.1
  i-d481497c     us-east-1a    12 mins        0.0    0.03        0.3    0.0
      0.0   99.7       0.0
  i-136e00c0     us-east-1b    12 mins        0.0    0.04        0.1    0.0
      0.0   99.9       0.0
  i-126e00c1     us-east-1b    12 mins       0.01    0.04        0.2    0.0
      0.0   99.7       0.1
  i-8b2cf575     us-east-1c    1 hour         0.0    0.01        0.2    0.0
      0.1   99.6       0.1
```

Reading the Output

The output displays the name of the environment, the environment's overall health, and the current date at the top of the screen:

```
elasticBeanstalkExa-env                                  Ok
    2015-07-08 23:13:20
```

The next three lines display the type of environment ("WebServer" in this case), the configuration (Ruby 2.1 with Puma), and a breakdown of how many instances are in each of the seven states:

```
WebServer
         Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0
```

The rest of the output is split into three sections. The first displays the *status* and the *cause* of the status for the environment overall, and then for each instance. The following example shows two instances in the environment with a status of `Info` and a cause indicating that a deployment has started:

```
  id             status     cause
    Overall      Ok
  i-d581497d     Info       Performing application deployment (running for 3
seconds)
  i-d481497c     Info       Performing application deployment (running for 3
seconds)
```

```
  i-136e00c0     Ok
  i-126e00c1     Ok
  i-8b2cf575     Ok
```

For information about health statuses and colors, see Health Colors and Statuses (p. 260).

The next section displays information from the web server logs on each instance. In this example, each instance is taking requests normally and there are no errors:

```
  id             r/sec    %2xx   %3xx   %4xx   %5xx      p99    p90      p75
     p50     p10
    Overall      13.7    100.0    0.0    0.0    0.0    1.403    0.970    0.710
   0.413   0.079
  i-d581497d     2.4     100.0    0.0    0.0    0.0    1.102*   0.865    0.601
   0.413   0.091
  i-d481497c     2.7     100.0    0.0    0.0    0.0    0.842*   0.788    0.480
   0.305   0.062
  i-136e00c0     4.1     100.0    0.0    0.0    0.0    1.520*   1.088    0.883
   0.524   0.104
  i-126e00c1     2.2     100.0    0.0    0.0    0.0    1.334*   0.791    0.760
   0.344   0.197
  i-8b2cf575     2.3     100.0    0.0    0.0    0.0    1.162*   0.867    0.698
   0.477   0.076
```

The final section shows operating system metrics for each instance:

```
  id             az            running     load 1  load 5      user%  nice%
system%  idle%   iowait%
  i-d581497d     us-east-1a    1 hour        0.06    0.05        3.5    0.0
   0.7   95.2      0.5
  i-d481497c     us-east-1a    1 hour         0.0    0.01        3.2    0.0
   0.8   95.8      0.1
  i-136e00c0     us-east-1b    1 hour         0.0    0.01        3.4    0.0
   0.7   95.8      0.0
  i-126e00c1     us-east-1b    1 hour        0.06    0.04        3.2    0.0
   0.7   95.9      0.1
  i-8b2cf575     us-east-1c    2 hours        0.0    0.03        1.8    0.0
   0.5   97.5      0.1
```

For information about the server and operating system metrics shown, see Instance Metrics (p. 262).

Interactive Health View

The `eb health` command displays a snapshot of your environment's health. To refresh the displayed information every 10 seconds, use the `--refresh` option:

```
$ eb health --refresh
 elasticBeanstalkExa-env                             Ok
     2015-07-09 22:10:04 (1 secs)
WebServer
                   Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  id             status     cause
```

```
    Overall      Ok
  i-bb65c145     Ok        Application deployment completed 35 seconds ago and
 took 26 seconds
  i-ba65c144     Ok        Application deployment completed 17 seconds ago and
 took 25 seconds
  i-f6a2d525     Ok        Application deployment completed 53 seconds ago and
 took 26 seconds
  i-e8a2d53b     Ok        Application deployment completed 32 seconds ago and
 took 31 seconds
  i-e81cca40     Ok

  id             r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
     p50     p10
    Overall      0.0         -      -      -      -        -        -        -
       -       -
  i-bb65c145     0.0         -      -      -      -        -        -        -
       -       -
  i-ba65c144     0.0         -      -      -      -        -        -        -
       -       -
  i-f6a2d525     0.0         -      -      -      -        -        -        -
       -       -
  i-e8a2d53b     0.0         -      -      -      -        -        -        -
       -       -
  i-e81cca40     0.0         -      -      -      -        -        -        -
       -       -

  id             az            running     load 1  load 5      user%  nice%
system%  idle%   iowait%
  i-bb65c145     us-east-1c    3 mins        0.33    0.16        0.6    0.0
   0.2   98.0       1.2
  i-ba65c144     us-east-1c    3 mins        0.51    0.16        3.7    0.0
   2.6   90.7       2.9
  i-f6a2d525     us-east-1b    3 mins        0.33    0.18        0.6    0.0
   0.1   99.3       0.0
  i-e8a2d53b     us-east-1b    3 mins        0.45     0.2        0.4    0.0
   0.0   98.5       1.1
  i-e81cca40     us-east-1a    3 hours        0.0    0.01        0.3    0.0
   0.0   99.6       0.1

 (Commands: Help,Quit,        )
```

This example shows an environment that has recently been scaled up from one to five instances. The scaling operation succeeded, and all instances are now passing health checks and are ready to take requests. In interactive mode, the health status updates every 10 seconds. In the upper right corner, a timer ticks down to the next update.

In the lower left corner, the report displays a list of options. To exit interactive mode, press **Q**. To scroll, press the arrow keys. To see a list of additional commands, press **H**.

Interactive Health View Options

When viewing environment health interactively, you can use keyboard keys to adjust the view and tell Elastic Beanstalk to replace or reboot individual instances. To see a list of available commands while viewing the health report in interactive mode, press **H** :

```
up,down,home,end   Scroll vertically
left,right         Scroll horizontally
F                  Freeze/unfreeze data
X                  Replace instance
B                  Reboot instance
<,>                Move sort column left/right
-,+                Sort order descending/ascending
P                  Save health snapshot data file
Z                  Toggle color/mono mode

Views
1                  All tables/split view
2                  Status Table
3                  Request Summary Table
4                  CPU%/Load Table
H                  This help menu
```

Health Colors and Statuses

Enhanced health reporting represents instance and overall environment health with four colors, similar ito basic health reporting (p. 247). Enhanced health reporting also provides seven health statuses, single word descriptors that provide a better indication of the state of your environment.

Instance Status and Environment Status

Every time Elastic Beanstalk runs a health check on your environment, enhanced health reporting checks the health of each instance in your environment by analyzing all of the data (p. 273) available. If any of the lower-level checks fails, Elastic Beanstalk downgrades the health of the instance.

Elastic Beanstalk displays the health information for the overall environment (color, status, and cause) in the environment management console (p. 51). This information is also available in the EB CLI. Health status and cause messages for individual instances are updated every ten seconds and are available from the EB CLI (p. 384) when you view health status with `eb health` (p. 255).

Elastic Beanstalk uses changes in instance health to evaluate environment health, but does not immediately change environment health status. When an instance fails health checks at least three times in any one-minute period, Elastic Beanstalk may downgrade the health of the environment. Depending on the number of instances in the environment and the issue identified, one unhealthy instance may cause Elastic Beanstalk to display an informational message or to change the environment's health status from green (OK) to yellow (Warning) or red (Degraded or Severe).

OK (Green)

An **instance** is passing health checks and the health agent is not reporting any problems.

Most instances in the **environment** are passing health checks and and the health agent is not reporting major issues.

An **instance** is passing health checks and is completing requests normally.

Example: Your environment was recently deployed and is taking requests normally. Five percent of requests are returning 400 series errors. Deployment completed normally on each **instance**.

Message (Instance): Application deployment completed 23 seconds ago and took 26 seconds.

Warning (Yellow)

The health agent is reporting a moderate number of request failures or other issues for an **instance** or **environment**.

An operation in progress on an **instance** and is taking a very long time.

Example: One instance in the **environment** has a status of Severe.

Message (Environment): Impaired services on 1 out of 5 instances

Degraded (Red)

The health agent is reporting a high number of request failures or other issues for an **instance** or **environment**.

Example: **Environment** is in the process of scaling up to 5 instances.

Message (Environment): 4 active instances is below Auto Scaling group minimum size 5

Severe (Red)

The health agent is reporting a very high number of request failures or other issues for an **instance** or **environment**.

Example: Elastic Beanstalk is unable to contact the load balancer to get instance health.

Message (Environment): *ELB health is failing or not available for all instances. None of the instances are sending data. Unable to assume role "arn:aws:iam::0123456789012:role/aws-elasticbeanstalk-service-role". Verify that the role exists and is configured correctly.*

Message (Instances): Instance ELB health has not been available for 37 minutes. No data. Last seen 37 minutes ago.

Info (Green)

An operation is in progress on an **instance**.

An operation is in progress on several instances in an **environment**.

Example: A new application version is being deployed to running instances.

Message (Environment): Command is executing on 3 out of 5 instances

Message (Instance): Performing application deployment (running for 3 seconds)

Pending (Grey)

An operation is in progress on an **instance** within the command timeout (p. 274).

Example: You have recently created the environment and **instances** are being bootstrapped.

Message: Performing initialization (running for 12 seconds)

Unknown (Grey)

Elastic Beanstalk and the health agent are reporting an insufficient amount of data on an **instance**.

Example: No data is being received.

Instance Metrics

Instance metrics provide information about the health of instances in your environment.The AWS Elastic Beanstalk Elastic Beanstalk health agent (p. 273) gathers and relays metrics about instances to Elastic Beanstalk, which analyzes the metrics to determine the health of the instances in your environments.

The Elastic Beanstalk health agent gathers metrics about instances from web server logs and the operating system. Web server logs provide information about incoming HTTP requests: how many requests came in, how many resulted in errors, and how long they took to resolve. The operating system provides snapshot information about the state of the instances' resources: the CPU usage, I/O bandwidth, and memory usage. These metrics are a subset of the information that you would see if you ran `top` on a Linux server.

The health agent gathers web server and operating system metrics and relays them to Elastic Beanstalk every ten seconds. Elastic Beanstalk analyzes the data and uses the results to update the health status for each instance and the environment.

Web Server Metrics

The Elastic Beanstalk health agent reads web server metrics fromlogs generated by the web container or server that processes requests on each instance in your environment. Elastic Beanstalk platforms are configured to generate two logs: one in human readable format and one in machine readable format. The health agent relays machine-readable logs to Elastic Beanstalk every ten seconds.

For more information on the log format used by Elastic Beanstalk, see Enhanced Health Log Format (p. 269).

Web Server Metrics

`r/sec`
: Average number of requests handled by the web server per second over the last 10 seconds.

`%2xx`, `%3xx`, `%4xx`, `%5xx`
: Percentage of requests that resulted in each type of status code over the last 10 seconds. For example, successful requests return a 200 OK, redirects are a 301, and a 404 is returned if the URL entered does not match any resources in the application.

`p99`, `p90`, `p75`, `p50`, `p10`
: Average latency for the slowest *x* percent of requests over the last 10 seconds.

Operating System Metrics

The Elastic Beanstalk health agent reports the following operating system metrics. Elastic Beanstalk uses these metrics to identify instances that are under sustained heavy load:

Operating System Metrics

`Running`
: The amount of time that has passed since the instance was launched.

`Load 1`, `Load 5`
: Load average in the last 1-minute and 5-minute periods. Shown as a decimal value indicating the average number of processes running during that time. If the number shown is higher than the number of vCPUs (threads) available, then the remainder is the average number of processes that were waiting.

For example, if your instance type has 4 vCPUs, and the load is 4.5, there was an average of .5 processes in wait during that time period, equivalent to one process was waiting 50 percent of the time.

`User %`, `Nice %`, `System %`, `Idle %`, `I/O Wait %`
Percentage of time that the CPU has spent in each state over the last 10 seconds.

Publishing Amazon CloudWatch Custom Metrics for an Environment

You can publish the data gathered by AWS Elastic Beanstalk enhanced health reporting to Amazon CloudWatch as custom metrics. Publishing metrics to CloudWatch lets you monitor changes in application performance over time and identify potential issues by tracking how resource usage and request latency scale with load.

By publishing metrics to CloudWatch you also make them available for use with monitoring graphs (p. 245) and alarms (p. 276). One free metric, *EnvironmentHealth* is enabled automatically when you use enhanced health reporting. Custom metrics other than *EnvironmentHealth* incur standard CloudWatch charges.

To publish CloudWatch custom metrics for an environment, you must first enable enhanced health reporting on the environment. See Enabling and Disabling AWS Elastic Beanstalk Enhanced Health Reporting (p. 251) for instructions.

Topics

Enhanced Health Reporting Metrics

When you enabled enhanced health reporting in your environment, the enhanced health reporting system automatically publishes one CloudWatch custom metric, *EnvironmentHealth*. To publish additional metrics to CloudWatch, configure your environment with the metrics that you want to publish using the AWS Management Console (p. 264), the EB CLI (p. 264), or .ebextensions (p. 103).

`EnvironmentHealth`
Environment only. This is the only CloudWatch metric that is published by the enhanced health system unless you configure additional metrics. Environment health is represented by one of seven statuses (p. 260).

`InstancesSevere`, `InstancesDegraded`, `InstancesWarning`, `InstancesInfo`, `InstancesOk`, `InstancesPending`, `InstancesUnknown`, `InstancesNoData`
Environment only. These metrics indicate the number of instances in the environment with each health status. `InstancesNoData` indicates the number of instances for which no data is being received for (if any).

`ApplicationRequestsTotal`, `ApplicationRequests5xx`, `ApplicationRequests4xx`, `ApplicationRequests3xx`, `ApplicationRequests2xx`
Instance and environment. Indicates the total number of requests completed by the instance or environment, and the number of requests that completed with each status code category.

`ApplicationLatencyP10`, `ApplicationLatencyP50`, `ApplicationLatencyP75`, `ApplicationLatencyP85`, `ApplicationLatencyP90`, `ApplicationLatencyP95`, `ApplicationLatencyP99`, `ApplicationLatencyP99.9`
Instance and environment. Indicates the average amount of time it takes to complete the fastest *x* percent of requests.

`LoadAverage1min`
: Instance only. The average CPU load of the instance over the last minute.

`InstanceHealth`
: Instance only. Indicates the current health status of the instance.

`RootFilesystemUtil`
: Instance only. Indicates the percentage of disk space in use.

`CPUIrq`, `CPUUser`, `CPUIdle`, `CPUSystem`, `CPUSoftirq`, `CPUIowait`, `CPUNice`
: Instance only. Indicates the percentage of time that the CPU has spent in each state over the last minute.

Configuring CloudWatch Metrics in the AWS Management Console

Use the AWS Management Console to configure your environment to publish enhanced health reporting metrics to CloudWatch and make them available for use with monitoring graphs and alarms.

To configure CloudWatch custom metrics in the AWS Management Console

1. Open the Elastic Beanstalk Management Console.
2. Navigate to the management console (p. 51) for your environment.
3. Choose **Configuration**.
4. On the **Health** panel, under **Web Tier**,choose ⚙ (edit).
5. Under **Health Reporting**, select the instance and environment metrics that you want to publish to CloudWatch. To select multiple metrics, hold the **Ctrl** key while choosing.

Enabling CloudWatch custom metrics adds them to the list of metrics available in the Monitoring Console (p. 244).

Configuring CloudWatch Custom Metrics with the EB CLI

You can use the EB CLI to configure custom metrics by saving your environment's configuration locally, adding an entry that defines the metrics to publish, and then uploading the configuration to Elastic Beanstalk. The saved configuration can be applied to an environment during or after creation.

To configure CloudWatch custom metrics with the EB CLI and saved configurations

1. Initialize your project folder with `eb init` (p. 390).
2. Create an environment by running the `eb create` (p. 394) command.
3. Save a configuration template locally by running the `eb config save` command. The following example uses the `--cfg` option to specify the name of the configuration.

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-
state.cfg.yml
```

4. Open the saved configuration file in a text editor.
5. Under `OptionSettings` > `aws:elasticbeanstalk:healthreporting:system:`, add a `ConfigDocument` key to enable each of the CloudWatch metrics that you want to enable. For example, the following `ConfigDocument` publishes `ApplicationRequests5xx` and `ApplicationRequests4xx` metrics at the environment level, and `ApplicationRequestsTotal` metrics at the instance level:

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      CloudWatchMetrics:
        Environment
          ApplicationRequests5xx: 60
          ApplicationRequests4xx: 60
        Instance:
          ApplicationRequestsTotal: 60
      Version: 1
    SystemType: enhanced
...
```

> **Note**
> In the example, 60 indicates the number of seconds between measurements. This is the only currently supported value.

6. Save the configuration file and close the text editor. In this example, the updated configuration file is saved with a different name (`02-cloudwatch-enabled.cfg.yml`) than the downloaded configuration file. This will create a separate saved configuration when the file is uploaded. You can use the same name as the downloaded file to overwrite the existing configuration without creating a new one.
7. Upload the updated configuration file to Elastic Beanstalk with the `eb config put` command:

   ```
   $ eb config put 02-cloudwatch-enabled
   ```

   When using the `eb config` `get` and `put` commands with saved configurations, do not include the file extension..
8. Apply the saved configuration to your running environment:

   ```
   $ eb config --cfg 02-cloudwatch-enabled
   ```

   The `--cfg` option specifies a named configuration file that is applied to the environment. The configuration can be saved locally or in Elastic Beanstalk. If a configuration file with the specified name exists in both locations, the EB CLI uses the local file.

Providing Custom Metric Config Documents

The config document for Amazon CloudWatch custom metrics is a JSON document that lists the metrics to publish at an environment and instance level. The following example shows a config document that enables all available custom metrics:

```

{
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
```

```
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
  },
  "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

For the AWS CLI, you pass the document as a value for the Value key in an option settings argument, which itself is a JSON object. In this case, quotes in the embedded document must be escaped:

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-
app --environment-name my-env --option-settings '[
    {
        "Namespace": "aws:elasticbeanstalk:healthreporting:system",
        "OptionName": "ConfigDocument",
        "Value": "{\"CloudWatchMetrics\": {\"Environment\": {\"Application
LatencyP99.9\": 60,\"InstancesSevere\": 60,\"ApplicationLatencyP90\": 60,\"Ap
plicationLatencyP99\": 60,\"ApplicationLatencyP95\": 60,\"InstancesUnknown\":
```

```
60,\"ApplicationLatencyP85\": 60,\"InstancesInfo\": 60,\"ApplicationRe
quests2xx\": 60,\"InstancesDegraded\": 60,\"InstancesWarning\": 60,\"Applica
tionLatencyP50\": 60,\"ApplicationRequestsTotal\": 60,\"InstancesNoData\":
60,\"InstancesPending\": 60,\"ApplicationLatencyP10\": 60,\"ApplicationRe
quests5xx\": 60,\"ApplicationLatencyP75\": 60,\"InstancesOk\": 60,\"Application
Requests3xx\": 60,\"ApplicationRequests4xx\": 60},\"Instance\": {\"Application
LatencyP99.9\": 60,\"ApplicationLatencyP90\": 60,\"ApplicationLatencyP99\":
60,\"ApplicationLatencyP95\": 60,\"ApplicationLatencyP85\": 60,\"CPUUser\":
60,\"ApplicationRequests2xx\": 60,\"CPUIdle\": 60,\"ApplicationLatencyP50\":
60,\"ApplicationRequestsTotal\": 60,\"RootFilesystemUtil\": 60,\"LoadAver
age1min\": 60,\"CPUIrq\": 60,\"CPUNice\": 60,\"CPUIowait\": 60,\"Application
LatencyP10\": 60,\"LoadAverage5min\": 60,\"ApplicationRequests5xx\": 60,\"Ap
plicationLatencyP75\": 60,\"CPUSystem\": 60,\"ApplicationRequests3xx\":
60,\"ApplicationRequests4xx\": 60,\"InstanceHealth\": 60,\"CPUSoftirq\":
60}},\"Version\": 1}"}
]'
```

For an `.ebextensions` configuration file in YAML, you can provide the JSON document as is:

```
  option_settings:
    - namespace: aws:elasticbeanstalk:healthreporting:system
      option_name: ConfigDocument
      value: {
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
  },
  "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

Using Enhanced Health Reporting with the AWS Elastic Beanstalk API

Because AWS Elastic Beanstalk enhanced health reporting has role and solution stack requirements, you must update scripts and code that you used prior to the release of enhanced health reporting before you can use it. To maintain backward compatibility, enhanced health reporting is not enabled by default when you create an environment using the Elastic Beanstalk API.

You configure enhanced health reporting by setting the service role, the instance profile, and Amazon CloudWatch configuration options for your environment. You can do this in three ways: by setting the configuration options in the `.ebextensions` folder, with saved configurations, or by configuring them directly in the `create-environment` call's `option-settings` parameter.

To use the API, SDKs, or AWS command line interface (CLI) to create an environment that supports enhanced health, you must:

- Create a service role and instance profile with the appropriate permissions (p. 19)
- Create a new environment with a solution stack for a new version of the platform configuration (p. 24)
- Set the health system type, instance profile, and service role configuration options (p. 103)

Use the following configuration options in the `aws:elasticbeanstalk:healthreporting:system`, `aws:autoscaling:launchconfiguration`, and `aws:elasticbeanstalk:environment` namespaces to configure your environment for enhanced health reporting.

Enhanced Health Configuration Options

SystemType

Namespace: `aws:elasticbeanstalk:healthreporting:system`

To enable enhanced health reporting, set to **enhanced**.

IamInstanceProfile

Namespace: `aws:autoscaling:launchconfiguration`

Set to the name of an instance profile configured for use with Elastic Beanstalk.

ServiceRole

Namespace: `aws:elasticbeanstalk:environment`

Set to the name of a service role configured for use with Elastic Beanstalk.

ConfigDocument (optional)

Namespace: `aws:elasticbeanstalk:healthreporting:system`

A JSON document that defines the and instance and environment metrics to publish to CloudWatch. For example:

```
{
  "CloudWatchMetrics":
    {
    "Environment":
      {
      "ApplicationLatencyP99.9":60,
      "InstancesSevere":60
      }
    "Instance":
      {
      "ApplicationLatencyP85":60,
      "CPUUser": 60
      }
    }
  "Version":1
}
```

Note
Config documents may require special formatting, such as escaping quotes, depending on how you provide them to Elastic Beanstalk. See Providing Custom Metric Config Documents (p. 265) for examples.

Enhanced Health Log Format

AWS Elastic Beanstalk platforms use a custom web server log format to efficiently relay information about HTTP requests to the enhanced health reporting system, which analyzes the logs, identifies issues, and sets the instance and environment health accordingly. If you disable the web server proxy on your environment and serve requests directly from the web container, you can still make full use of enhanced health reporting by configuring your server to output logs in the location and format that the Elastic Beanstalk health agent (p. 273) uses.

Web Server Log Configuration

Elastic Beanstalk platforms are configured to output two logs with information about HTTP requests. The first is in verbose format and provides detailed information about the request, including the requester's user agent information and a human-readable timestamp.

The following examples are from an Nginx proxy running on a Ruby web server environment, but the format is similar for Apache:

`/var/log/nginx/access.log`

```
172.31.24.3 - - [23/Jul/2015:00:21:20 +0000] "GET / HTTP/1.1" 200 11 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4
libidn/1.23 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:21 +0000] "GET / HTTP/1.1" 200 11 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4
libidn/1.23 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4
libidn/1.23 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4
libidn/1.23 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4
libidn/1.23 librtmp/2.3" "177.72.242.17"
```

The second log is in terse format. It includes information relevant only to enhanced health reporting. This log is output to a subfolder named `healthd` and rotates hourly. Old logs are deleted immediately after rotating out.

The following example shows a log in the machine-readable format.

`/var/log/nginx/healthd/application.log.2015-07-23-00`

```
1437609879.311"/"200"0.083"0.083"177.72.242.17
1437609879.874"/"200"0.347"0.347"177.72.242.17
1437609880.006"/bad/path"404"0.001"0.001"177.72.242.17
1437609880.058"/"200"0.530"0.530"177.72.242.17
1437609880.928"/bad/path"404"0.001"0.001"177.72.242.17
```

The enhanced health log format includes the following information:

- The time of the request, in Unix time. For Nginx this is in floating-point seconds, with three decimal places. For Apache, it is in whole milliseconds.
- The path of the request.
- The HTTP status code for the result.
- The request time in floating-point seconds.
- The upstream time in floating-point seconds.
- The `X-Forwarded-For` HTTP header.

The following example shows the log configuration for Nginx with the `healthd` log format highlighted:

`/etc/nginx/conf.d/webapp_healthd.conf`

```
upstream my_app {
  server unix:///var/run/puma/my_app.sock;
}

log_format healthd '$msec"$uri"'
                '$status"$request_time"$upstream_response_time"'
                '$http_x_forwarded_for';

server {
  listen 80;
  server_name _ localhost; # need to listen to localhost for worker tier

  if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
    set $year $1;
    set $month $2;
    set $day $3;
    set $hour $4;
  }

  access_log  /var/log/nginx/access.log  main;
  access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour 
healthd;

  location / {
    proxy_pass http://my_app; # match the name of upstream directive which is 
defined above
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }

  location /assets {
    alias /var/app/current/public/assets;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
  }

  location /public {
    alias /var/app/current/public;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
  }
}
```

Generating Logs for Enhanced Health Reporting

To provide logs to the health agent, you must do the following:

- Output logs in the correct format, as shown in the previous section
- Output logs to `/var/log/nginx/healthd/`
- Name logs using the following format: `application.log.$year-$month-$day-$hour`
- Rotate logs once per hour
- Do not truncate logs

Notifications and Troubleshooting

This page lists example cause messages for common issues and links to more information. Cause messages appear in the environment dashboard (p. 244) and are recorded in events (p. 278) when health issues persist across several checks.

Application Server

15% of requests are erroring with HTTP 4xx

A high percentage of HTTP requests to an **instance** or **environment** are failing with 4xx errors.

A 400 series status code indicates that the user made a bad request, such as requesting a page that doesn't exist (404 File Not Found) or that the user doesn't have access to (403 Forbidden). A low number of 404s is not unusual but a large number could mean that there are internal or external links to unavailable pages. These issues can be resolved by fixing bad internal links and adding redirects for bad external links.

5% of the requests are failing with HTTP 5xx

A high percentage of HTTP requests to an **instance** or **environment** are failing with 500 series status codes.

A 500 series status code indicates that the application server encountered an internal error. These issues indicate that there is an error in your application code and should be identified and fixed quickly.

95% of CPU is in use

On an **instance**, the health agent is reporting an extremely high percentage of CPU usage and sets the instance health to **Warning** or **Degraded**.

Scale your environment to take load off of instances.

Worker Instance

20 messages waiting in the queue (25 seconds ago)

Requests are being added to your worker environment's queue faster than they can be processed. Scale your environment to increase capacity.

5 messages in Dead Letter Queue (15 seconds ago)

Worker requests are failing repeatedly and being added to the Dead Letter Queue (p. 71). Check the requests in the dead letter queue to see why they are failing.

Other Resources

4 active instances is below Auto Scaling group minimum size 5

The number of instances running in your environment is fewer than the minimum configured for the Auto Scaling group.

Auto Scaling group (groupname) notifications have been deleted or modified

The notifications configured for your Auto Scaling group have been modified outside of Elastic Beanstalk.

Topics

The Elastic Beanstalk Health Agent

The Elastic Beanstalk health agent is a daemon process that runs on each EC2 instance in your environment, monitoring operating system and application-level health metrics and reporting issues to Elastic Beanstalk. The health agent is included in all Linux platform solution stacks starting with version 2.0 of each configuration.

The health agent reports similar metrics to those published to CloudWatch (p. 248) by Auto Scaling and Elastic Load Balancing as part of basic health reporting (p. 247), including CPU load, HTTP codes, and latency. The health agent, however, reports directly to Elastic Beanstalk, with greater granularity and frequency than basic health reporting.

For basic health, these metrics are published every five minutes and can be monitored with graphs in the environment management console. With enhanced health, the Elastic Beanstalk health agent reports metrics to Elastic Beanstalk every ten seconds. Elastic Beanstalk uses the metrics provided by the health agent to determine the health status of each instance in the environment, and, combined with other factors (p. 273), to determine the overall health of the environment.

The overall health of the environment can be viewed in real-time in the environment dashboard and is published to CloudWatch by Elastic Beanstalk every sixty seconds. Detailed metrics reported by the health agent can be viewed in real time with the `eb health` (p. 255) command in the EB CLI (p. 384).

For an additional charge, you can choose to publish individual instance and environment level metrics to CloudWatch every sixty seconds. Metrics published to CloudWatch can then be used to create monitoring graphs (p. 245) in the environment management console (p. 51).

Enhanced health reporting only incurs a charge if you choose to publish enhanced health metrics to CloudWatch. When you use enhanced health, you still get the basic health metrics published for free, even if you don't choose to publish enhanced health metrics.

See Instance Metrics (p. 262) for details on the metrics reported by the health agent. For details on publishing enhanced health metrics to CloudWatch, see Publishing Amazon CloudWatch Custom Metrics for an Environment (p. 263).

Factors in Determining Instance and Environment Health

In addition to the basic health reporting system checks, including Elastic Load Balancing Health Check (p. 248) and resource monitoring (p. 248), Elastic Beanstalk enhanced health reporting gathers additional data about the state of the instances in your environment, including operating system metrics, server logs, and the state of ongoing environment operations such as deployments and updates. The Elastic Beanstalk health reporting service combines information from all available sources and analyzes it to determine the overall health of the environment.

Operations and Commands

When you perform an operation on your environment, such as deploying a new version of an application, Elastic Beanstalk makes several changes that cause the health status of the environment to change.

For example, when you deploy a new version of an application to an environment that is running multiple instances, you might see messages similar the following as you monitor the environment's health with the EB CLI (p. 255):

```
id             status     cause
  Overall      Info       Command is executing on 3 out of 5 instances
  i-bb65c145     Pending    91 % of CPU is in use. 24 % in I/O wait
                            Performing application deployment (running for 31 
seconds)
  i-ba65c144     Pending    Performing initialization (running for 12 seconds)

  i-f6a2d525     Ok        Application deployment completed 23 seconds ago and
 took 26 seconds
  i-e8a2d53b     Pending    94 % of CPU is in use. 52 % in I/O wait
                            Performing application deployment (running for 33 
seconds)
  i-e81cca40     Ok
```

In this example, the overall status of the environment is `Ok` and the cause of this status is that the *Command is executing on 3 out of 5 instances*. Three of the instances in the environment have the status *Pending*, indicating that an operation is in progress.

When an operation completes, Elastic Beanstalk reports additional informationabout the operation. For the example, Elastic Beanstalk displays the following information about an instance that has already been updated with the new version of the application:

```
i-f6a2d525     Ok          Application deployment completed 23 seconds ago and 
took 26 seconds
```

In the cause column, Elastic Beanstalk includes informational messages about successful operations and other healthy states across multiple health checks, but they do not persist indefinitely. Causes for unhealthy environment statuses persist until the environment returns to a healthy status.

Command Timeout

Elastic Beanstalk applies a command timeout from the time an operation begins to allow an instance to transition into a healthy state. This command timeout is set in your environment's update and deployment configuration (in the aws:elasticbeanstalk:command (p. 115) namespace) and defaults to 10 minutes.

During rolling updates, Elastic Beanstalk applies a separate timeout to each batch in the operation. This timeout is set as part of the environment's rolling update configuration (in the aws:autoscaling:updatepolicy:rollingupdate (p. 111) namespace). If all instances in the batch are healthy within the command timeout, the operation continues to the next batch. If not, the operation fails.

For a web server environment to be considered healthy, each instance in the environment or batch must pass 12 consecutive health checks over the course of two minutes. For worker tier, each instance must pass 18 health checks. Prior to command timeout, Elastic Beanstalk does not lower an environment's health status when health checks fail. As long as the instances in the environment and become healthy within the command timeout, the operation succeeds.

HTTP Requests

When no operation is in progress on an environment, the primary source of information about instance and environment health is the web server logs for each instance. To determine the health of an instance and the overall health of the environment, Elastic Beanstalk considers the number of requests, the result of each request, and the speed at which each request was resolved.

If you use Multicontainer Docker, which does not include a web server, or disable the web server (Nginx or Apache) that is included in other Elastic Beanstalk platforms, additional configuration is required to get the Elastic Beanstalk health agent (p. 273) logs in the format that it needs to relay health information to the Elastic Beanstalk service. See Enhanced Health Log Format (p. 269) for details.

Operating System Metrics

Elastic Beanstalk monitors operating system metrics reported by the health agent to identify instances that are consistently low on system resources.

See Instance Metrics (p. 262) for details on the metrics reported by the health agent.

Enhanced Health Roles

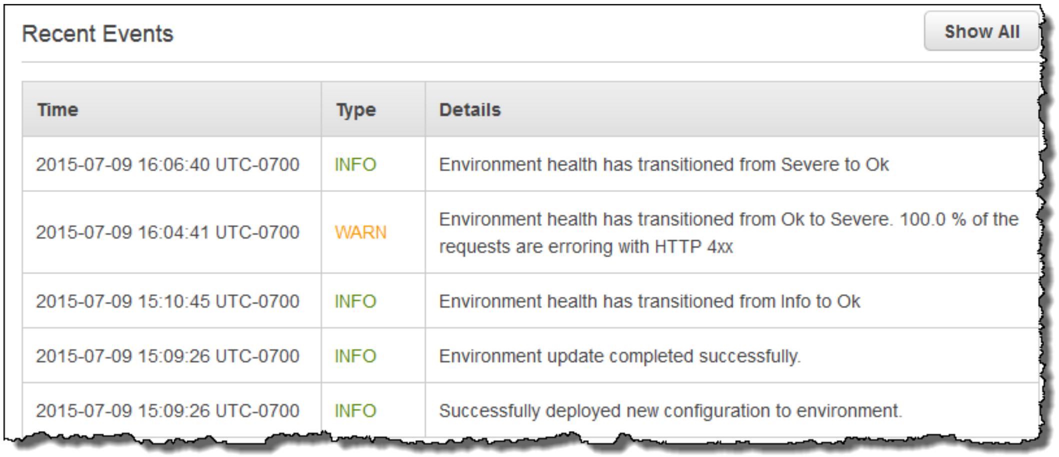
Enhanced health reporting requires two roles—a service role for Elastic Beanstalk and an instance profile for the environment. The service role allows Elastic Beanstalk to interact with other AWS services on your behalf in order to gather information about the resources in your environment. The instance profile allows the instances in your environment to write logs to Amazon S3.

When you create an Elastic Beanstalk environment in the AWS Management Console, the console prompts you to create an instance profile and service role with appropriate permissions. The EB CLI also assists you in creating these roles when you call `eb create` to create an environment.

If you use the API, an SDK, or the AWS CLI to create environments, you must create these roles beforehand and specify them during environment creation to use enhanced health. For instructions on creating appropriate roles for your environments, see Service Roles, Instance Profiles, and User Policies (p. 19).

Enhanced Health Events

The enhanced health system generates events when an environment transitions between states. The following example shows events output by an environment transitioning between Info, OK and Severe states:



Time	Type	Details
2015-07-09 16:06:40 UTC-0700	INFO	Environment health has transitioned from Severe to Ok
2015-07-09 16:04:41 UTC-0700	WARN	Environment health has transitioned from Ok to Severe. 100.0 % of the requests are erroring with HTTP 4xx
2015-07-09 15:10:45 UTC-0700	INFO	Environment health has transitioned from Info to Ok
2015-07-09 15:09:26 UTC-0700	INFO	Environment update completed successfully.
2015-07-09 15:09:26 UTC-0700	INFO	Successfully deployed new configuration to environment.

When transitioning to a worse state, Elastic Beanstalk includes a message indicating the cause in the event.

Not all changes in status at an instance level will cause Elastic Beanstalk to emit an event. To prevent false alarms, Elastic Beanstalk only generates a health related event if an issue persists across multiple checks.

Real time environment level health information, including status, color and cause, is available in the environment dashboard (p. 52) and the EB CLI (p. 384). By attaching the EB CLI to your environment and running the `eb health` (p. 255) command, you can also view real time statuses from each of the *instances* in your environment.

Enhanced Health Reporting Behavior during Updates, Deployments, and Scaling

Enabling enhanced health reporting can affect how your environment behaves during configuration updates and deployments. Elastic Beanstalk won't complete a batch of updates until all of the instances pass health checks consistently, and since enhanced health reporting applies a higher standard for health and monitors more factors, instances that pass basic health reporting's ELB health check (p. 248) won't necessarily pass muster with enhanced health reporting. See the topics on rolling configuration updates (p. 169) and rolling deployments (p. 174) for details on how health checks affect the update process.

Enhanced health reporting can also highlight the need to set a proper health check URL (p. 190) for Elastic Load Balancing. When your environment scales up to meet demand, new instances will start taking requests as soon as they pass enough ELB health checks. If a health check URL is not configured, this can be as little as 20 seconds after a new instance is able to accept a TCP connection.

If your application hasn't finished starting up by the time the load balancer declares it healthy enough to receive traffic, you will see a flood of failed requests, and your environment will start to fail health checks. A health check URL that hits a path served by your application can prevent this issue; ELB health checks won't pass until a GET request to the health check URL returns a 200 status code.

Manage Alarms

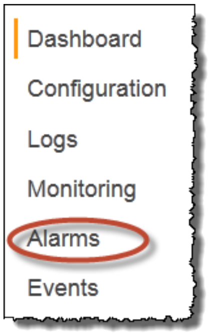
You can create alarms for metrics that you are monitoring by using the AWS Management Console. Alarms help you monitor changes to your environment so that you can easily identify and mitigate problems before they occur. For example, you can set an alarm that notifies you when CPU utilization in an environment exceeds a certain threshold, ensuring that you are notified before a potential problem occurs. For more information, see Using Elastic Beanstalk with Amazon CloudWatch (p. 290).

> **Note**
> Elastic Beanstalk uses CloudWatch for monitoring and alarms, meaning CloudWatch costs are applied to your AWS account for any alarms that you use.

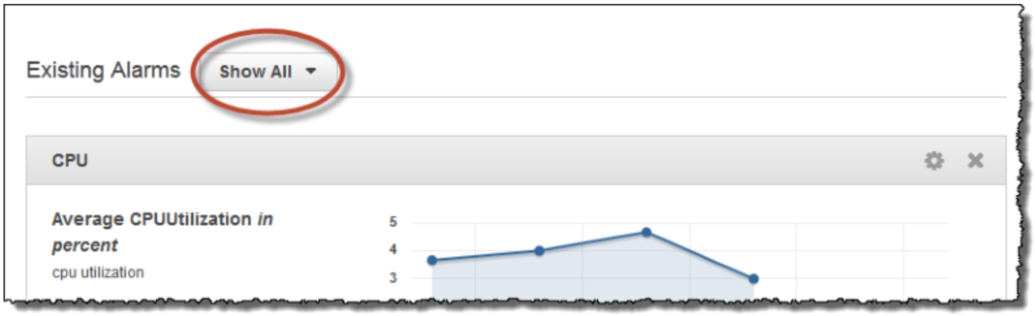
For more information about monitoring specific metrics, see Basic Health Reporting (p. 247).

To check the state of your alarms

1. From the Elastic Beanstalk console applications page, click the environment name that you want to manage alarms for.
2. From the navigation menu, click **Alarms** to see a list of alarms.

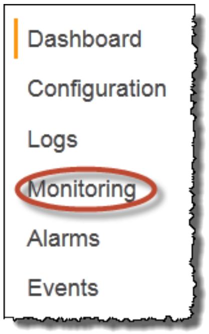
   If any alarms is in the alarm state, they are flagged with ⚠ (warning).
3. To filter alarms, click the drop-down filter and select the filter that you want.



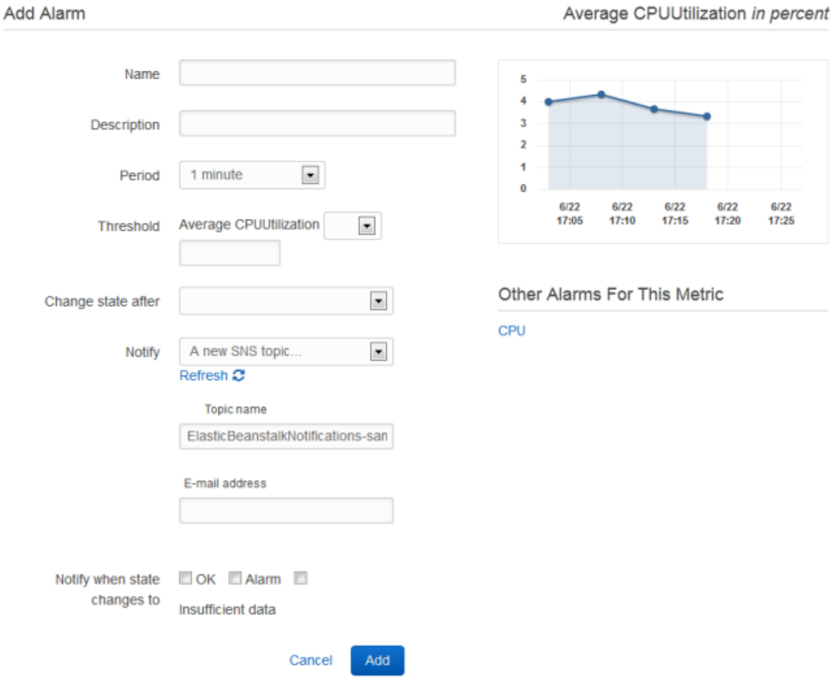
4. To edit or delete an alarm, click ⚙ (edit) or ✖ (delete).

To create an alarm

1. From the Elastic Beanstalk console applications page, click the environment name that you want to add alarms to.
2. From the navigation menu, click **Monitoring**.



3. For the metric that you want to create an alarm for, click 🔔. You are directed to the **Alarms** page.



Name

Description

Period: 1 minute

Threshold: Average CPUUtilization

Change state after

Notify: A new SNS topic...

Topic name: ElasticBeanstalkNotifications-san

E-mail address

Notify when state changes to: □ OK □ Alarm □ Insufficient data

4. Enter details about the alarm:

   - **Name**: A name for this alarm.
   - **Description** (optional): A short description of what this alarm is.
   - **Period**: The time interval between readings.
   - **Threshold**: Describes the behavior and value that the metric must exceed in order to trigger an alarm.
   - **Change state after**: The amount a time after a threshold has been exceed that triggers a change in state of the alarm.
   - **Notify**: The Amazon SNS topic that is notified when an alarm changes state.
   - **Notify when state changes to**:
     - **OK**: The metric is within the defined threshold.
     - **Alarm**: The metric exceeded the defined threshold.
     - **Insufficient data**: The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

5. Click **Add**. The environment status changes to gray while the environment updates. You can view the alarm that you created by going to the **Alarms** page.

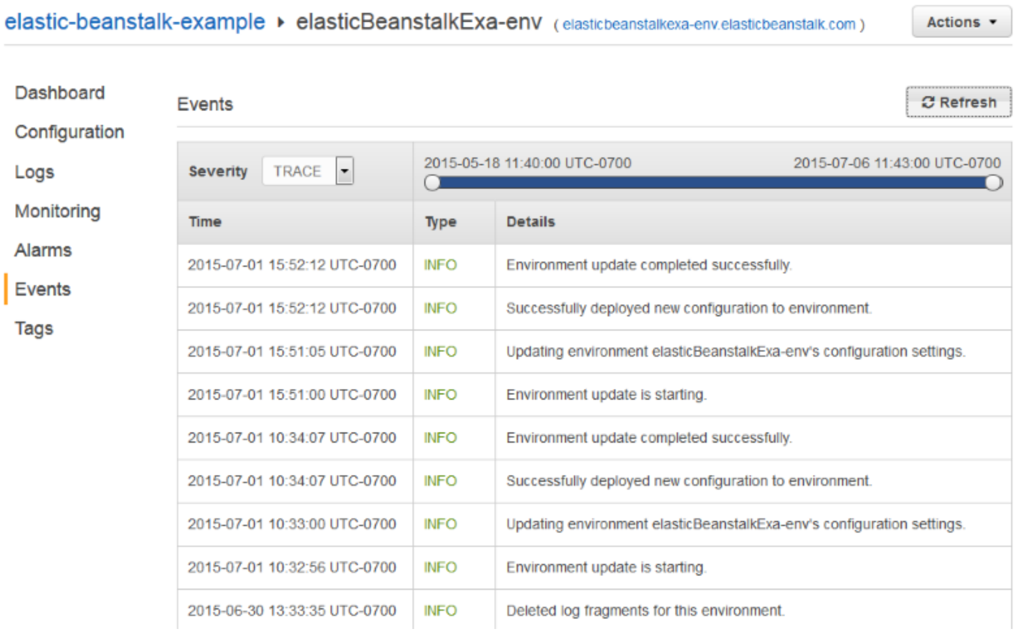
View Events

You can use the AWS Management Console to access events and notifications associated with your application. For more details on the most common events, see Understanding Environment Launch Events (p. 531). For information on how to view events using the AWS Toolkit for Eclipse, see Viewing Events (p. 598).

AWS Management Console

To view environment and application events

1. Navigate to the management console (p. 51) for your environment.
2. From the navigation menu, click **Events**.


elastic-beanstalk-example ▸ elasticBeanstalkExa-env (elasticbeanstalkexa-env.elasticbeanstalk.com)
Actions ▾
Dashboard
Configuration
Logs
Monitoring
Alarms
Events
Tags
Events
Refresh
Severity
TRACE

2015-05-18 11:40:00 UTC-0700 2015-07-06 11:43:00 UTC-0700

Time	Type	Details
2015-07-01 15:52:12 UTC-0700	INFO	Environment update completed successfully.
2015-07-01 15:52:12 UTC-0700	INFO	Successfully deployed new configuration to environment.
2015-07-01 15:51:05 UTC-0700	INFO	Updating environment elasticBeanstalkExa-env's configuration settings.
2015-07-01 15:51:00 UTC-0700	INFO	Environment update is starting.
2015-07-01 10:34:07 UTC-0700	INFO	Environment update completed successfully.
2015-07-01 10:34:07 UTC-0700	INFO	Successfully deployed new configuration to environment.
2015-07-01 10:33:00 UTC-0700	INFO	Updating environment elasticBeanstalkExa-env's configuration settings.
2015-07-01 10:32:56 UTC-0700	INFO	Environment update is starting.
2015-06-30 13:33:35 UTC-0700	INFO	Deleted log fragments for this environment.

The Events page shows you a list of all events that have been recorded for the environment and application version. You can filter on the type of events by using the **Severity** drop-down list. You can also filter when the events occurred by using the time slider.

Command Line

The EB CLI (p. 384) and AWS CLI (p. 523) both provide commands for retrieving events. If you are managing your environment using the EB CLI, use `eb events` (p. 417) to print a list of events. This command also has a `--follow` option that continues to show new events until you press **Ctrl**+**C** to stop output.

To pull events using the AWS CLI, use the `describe-events` command and specify the environment by name or ID:

```
$ aws elasticbeanstalk describe-events --environment-id e-gbjzqccra3
{
    "Events": [
        {
            "ApplicationName": "elastic-beanstalk-example",
            "EnvironmentName": "elasticBeanstalkExa-env",
            "Severity": "INFO",
            "RequestId": "a4c7bfd6-2043-11e5-91e2-9114455c358a",
            "Message": "Environment update completed successfully.",
            "EventDate": "2015-07-01T22:52:12.639Z"
        },
...
```

For more information on the command line tools, see Tools (p. 384).

API

You can use the Elastic Beanstalk APIs to view all events for your application. In this example, we use the APIs to get a list of all events for an application named My First Elastic Beanstalk Application.

To view all application events

- Call `DescribeEvents` with the following parameter:
  - *`ApplicationName`* = `My First Elastic Beanstalk Application`

  Example

  ```
  https://elasticbeanstalk.us-west-2.amazon.com/?Application
  Name=My%20First%20Elastic%20Beanstalk%20Application
  &Operation=DescribeEvents
  &AuthParams
  ```

Listing and Connecting to Server Instances

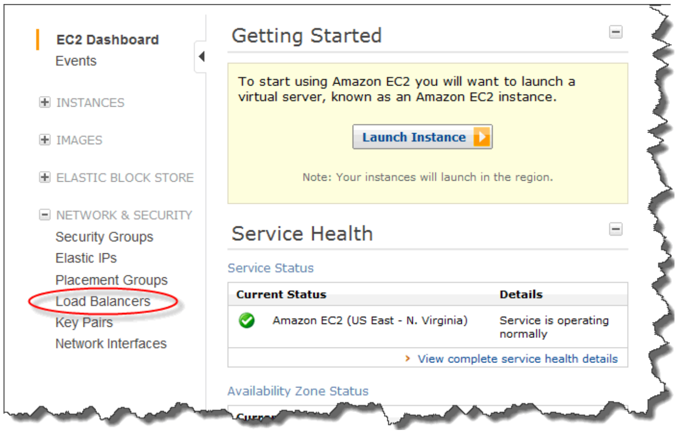
You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Management Console. You can connect to the instances using any SSH client. For more information about listing and connecting to Server Instances using the AWS Toolkit for Eclipse, see Listing and Connecting to Server Instances (p. 610). You can connect to the instances running Windows using Remote Desktop. For more information about listing and connecting to Server Instances using the AWS Toolkit for Visual Studio, see Listing and Connecting to Server Instances (p. 660).

> **Important**
> You must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances. You can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, go to the *Amazon EC2 Getting Started Guide*. For more information on how to configure your Amazon EC2 instances to use an Amazon EC2 key pair, see Amazon EC2 Key Pairs (p. 201).
> Elastic Beanstalk does not enable remote connections to EC2 instances in a Windows container by default except for legacy Windows containers. (Beanstalk configures EC2 instances in legacy Windows containers to use port 3389 for RDP connections.) You can enable remote connections to your EC2 instances running Windows by adding a rule to a security group that authorizes inbound traffic to the instances. We strongly recommend that you remove the rule when you end your remote connection. You can add the rule again the next time you need to log in remotely. For more information, see Adding a Rule for Inbound RDP Traffic to a Windows Instance and Connect to Your Windows Instance in the *Amazon Elastic Compute Cloud User Guide for Microsoft Windows*.

To view and connect to Amazon EC2 instances for an environment

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the region list, select a region.

3. In the navigation (left) pane of the console, click **Load Balancers**.



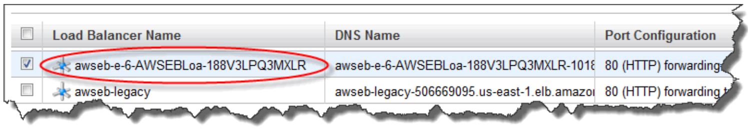
To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the region.

Current Status	Details
Amazon EC2 (US East - N. Virginia)	Service is operating normally

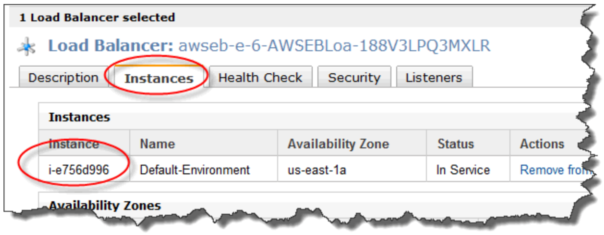
› View complete service health details

4. Load balancers created by Elastic Beanstalk will have a **awseb** in the name. Find the load balancer for your environment and click it.



□	Load Balancer Name	DNS Name	Port Configuration
☑	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR-1018	80 (HTTP) forwarding
□	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding t

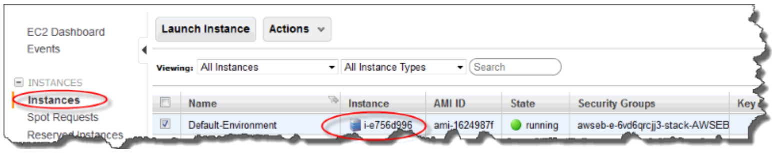
5. Click the **Instances** tab in the bottom pane of the console window.



Instance	Name	Availability Zone	Status	Actions
i-e756d996	Default-Environment	us-east-1a	In Service	Remove fro

A list of the instances that the load balancer for your Elastic Beanstalk environment uses is displayed. Make a note of an instance ID that you want to connect to.

6. Click the **Instances** link in the left side of the Amazon EC2 console, and find your instance ID in the list.



□	Name	Instance	AMI ID	State	Security Groups	Key
☑	Default-Environment	i-e756d996	ami-1624987f	● running	awseb-e-6vd6qrcjj3-stack-AWSEB	

7. Right-click the instance ID for the Amazon EC2 instance running in your environment's load balancer, and then select **Connect** from the context menu.
8. Make a note of the instance's public DNS address on the **Description** tab.
9. To connect to an instance running Linux, use the SSH client of your choice to connect to your instance and type **ssh -i .ec2/mykeypair.pem ec2-user@<public-DNS-of-the-instance>** . For instructions on how to connect to an instance running Windows, see Connect to your Windows Instance in the *Amazon Elastic Compute Cloud Microsoft Windows Guide*.

For more information on connecting to an Amazon EC2 instance, see the Amazon Elastic Compute Cloud Getting Started Guide.

Instance Logs

You can access logs from the Amazon EC2 instances running your applications. There are several ways to do this:

- View a snapshot of the last 100 lines of logs (also known as tail logs) in the Elastic Beanstalk console.
- Download all logs (also known as bundle logs) from the Elastic Beanstalk console.

  Note
  Legacy container types and Windows container types do not support bundle logs.
- Configure your environment to automatically publish logs to an Amazon S3 bucket.

This topic explains how to access your logs using each method.

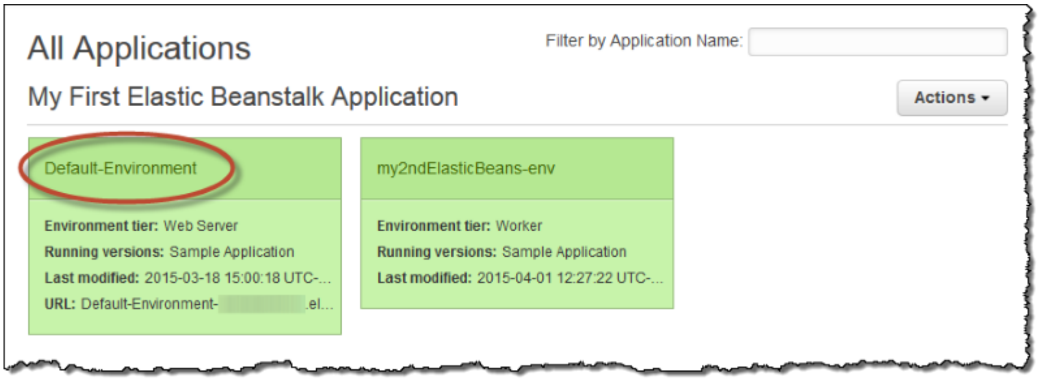
> **Note**
> In the eu-central-1 region, accessing logs from the instances that run your applications requires a custom IAM role with permission to rotate logs. If you use a custom IAM role to deploy and manage your application, you must attach a policy to the role that grants Elastic Beanstalk permission to rotate logs. For more information, see Using a Custom Instance Profile (p. 341). If you use a default instance profile, no additional configuration of the IAM role is required.

Viewing Tail Log Snapshots in the Elastic Beanstalk Console

To take a snapshot and view tail logs in the console

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the name of the environment for which you want to view logs.

   Note
   If you don't see any applications or environments listed on the **All Applications** page, ensure that you have selected the correct region and that you have created an environment for your application.



All Applications

Filter by Application Name:

My First Elastic Beanstalk Application

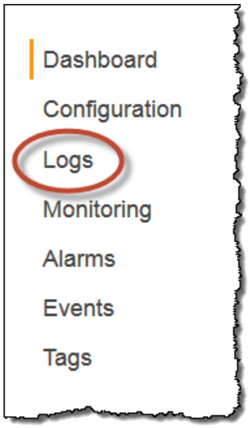
Default-Environment

Environment tier: Web Server
Running versions: Sample Application
Last modified: 2015-03-18 15:00:18 UTC-...
URL: Default-Environment-[redacted].el...

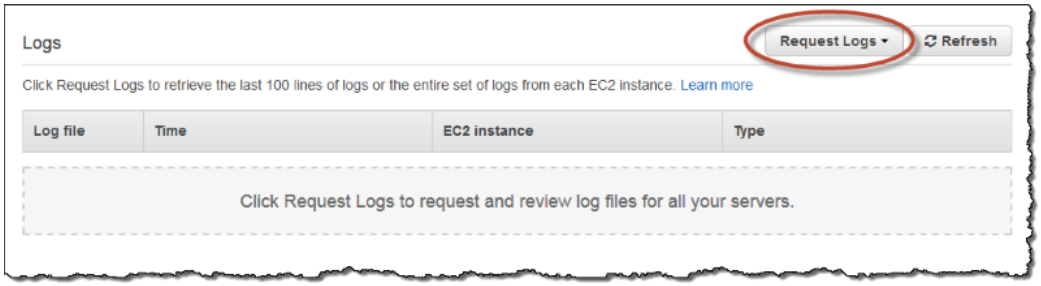
my2ndElasticBeans-env

Environment tier: Worker
Running versions: Sample Application
Last modified: 2015-04-01 12:27:22 UTC-...

3. In the navigation menu, click **Logs**. The **Logs** page lists logs you recently requested for your environment.



4. To get the latest snapshot of the logs for your Elastic Beanstalk application, click **Request Logs**, and then click **Last 100 Lines**.



Logs

Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. Learn more

Log file	Time	EC2 instance	Type

Click Request Logs to request and review log files for all your servers.

   Note

   It takes several seconds to retrieve the log files. You might need to click the **Refresh** button to see the contents of the log files.

5. To view the contents of the logs you requested, in the **Log file** column, click **Download**.

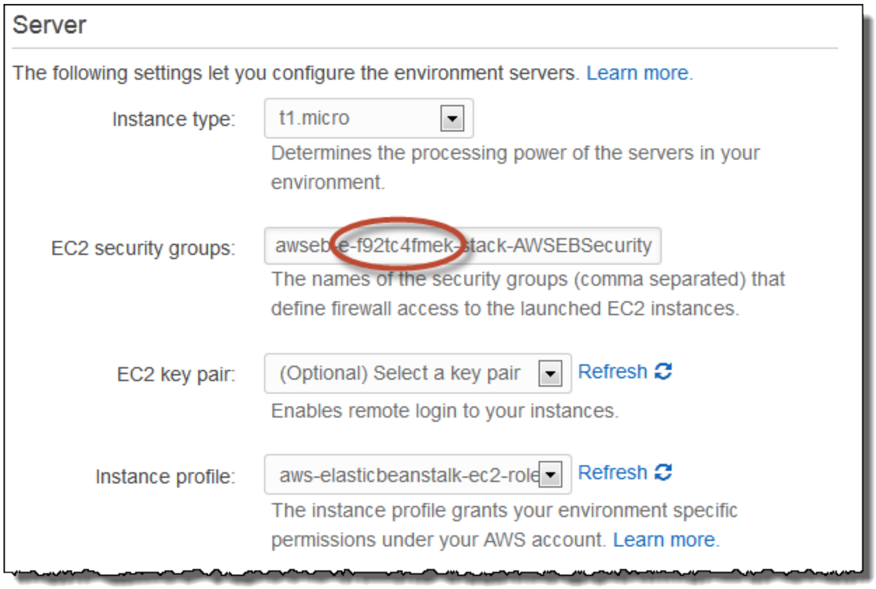
   A web page displays the text output of the log file snapshot.

A copy of the logs is placed in the Amazon S3 bucket associated with your application for 15 minutes and then they are deleted. You can request new logs later if necessary. Depending on whether you are using a legacy or non-legacy container, you can access these logs in one of the following locations. If you are not sure if you are using a legacy or non-legacy container, see To check if you are using a legacy container type (p. 92).

- **non-legacy** — `elasticbeanstalk-`*`region`*`-`*`account id`*`/resources/environments/logs/tail/`*`environment ID`*`/`*`instance ID`*`/`

  You can find your instance ID in the **EC2 instance** column on the **Logs** page shown in the preceding diagram.

  You can find your environment ID in the **Server** section of the **Instances** configuration page.


Server

The following settings let you configure the environment servers. Learn more.

Instance type: t1.micro
Determines the processing power of the servers in your environment.

EC2 security groups: awseb-e-f92tc4fmek-stack-AWSEBSecurity
The names of the security groups (comma separated) that define firewall access to the launched EC2 instances.

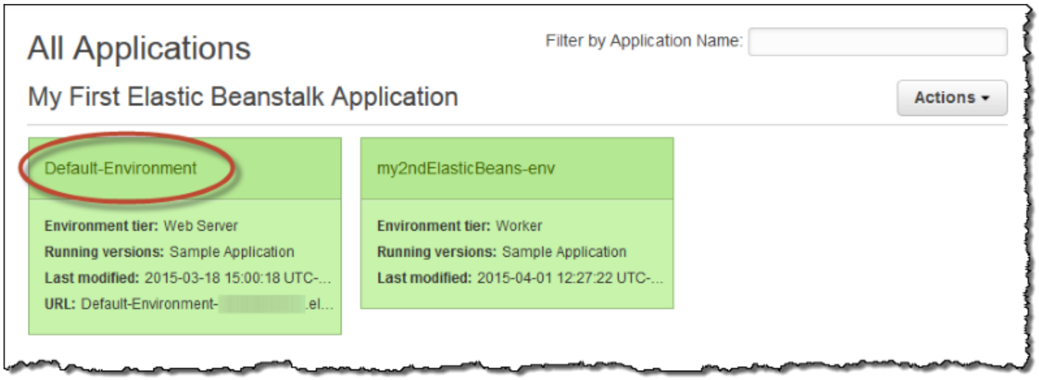
EC2 key pair: (Optional) Select a key pair
Enables remote login to your instances.

Instance profile: aws-elasticbeanstalk-ec2-role
The instance profile grants your environment specific permissions under your AWS account. Learn more.

- **legacy** — Use the `RetrieveEnvironmentInfo` API to retrieve the location for the tail logs. For the CLI reference for this API, see elastic-beanstalk-retrieve-environment-info (p. 509). For the API reference, go to RetrieveEnvironmentInfo in the *AWS Elastic Beanstalk API Reference*.

Downloading Bundle Logs from the Elastic Beanstalk Console

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the name of the environment for which you want to view logs.


All Applications

Filter by Application Name:

My First Elastic Beanstalk Application

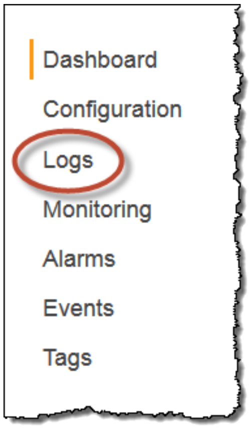
Default-Environment

Environment tier: Web Server
Running versions: Sample Application
Last modified: 2015-03-18 15:00:18 UTC-...
URL: Default-Environment- .el...

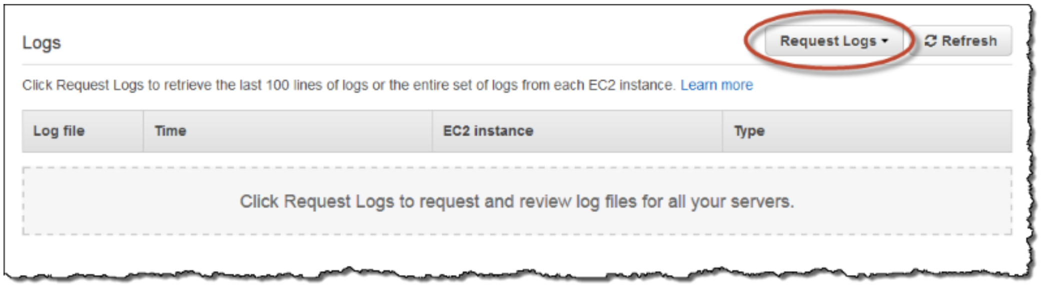
my2ndElasticBeans-env

Environment tier: Worker
Running versions: Sample Application
Last modified: 2015-04-01 12:27:22 UTC-...

3. In the navigation menu, click **Logs**. The **Logs** page lists logs you recently requested for your environment.



4. To get all logs from each of the Amazon EC2 instances for your Elastic Beanstalk application, click **Request Logs**, and then click **Full Logs**.



Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. Learn more

Log file	Time	EC2 instance	Type

Click Request Logs to request and review log files for all your servers.

Note

It takes several seconds to retrieve the log files. You might need to click the **Refresh** button.

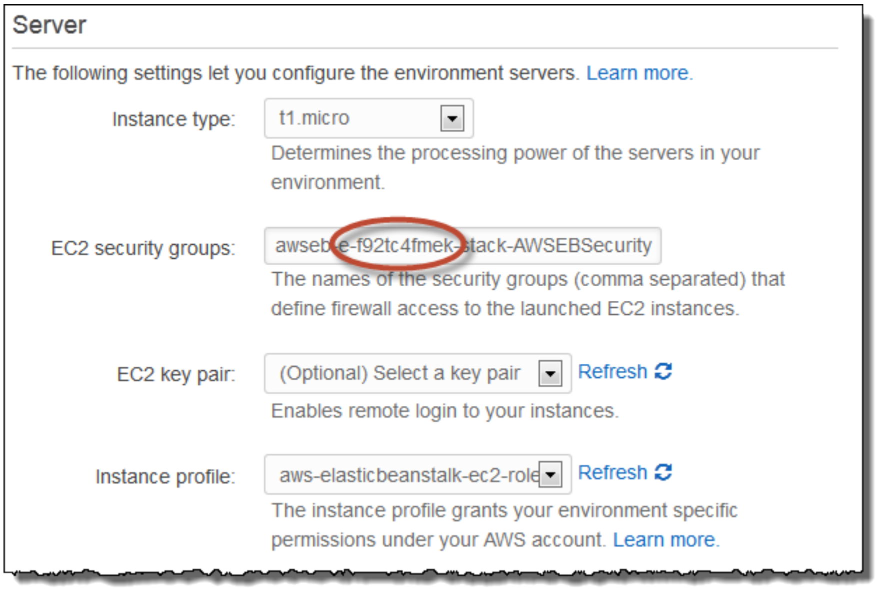
5. To view the contents of the logs you requested, in the **Log file** column, click **Download**. The bundle logs are compressed into a `.zip` file. When you are prompted, you can either open the file or save the file to view later.

A copy of the logs is placed in the Amazon S3 bucket associated with your application for 15 minutes and then they are deleted. You can request new logs later if necessary. Depending on whether you are using a legacy or non-legacy container, you can access these logs in one of the following locations. If you are not sure if you are using a legacy or non-legacy container, see To check if you are using a legacy container type (p. 92).

- **non-legacy** — elasticbeanstalk-*region*-*account id*/resources/environments/logs/tail/*environment ID*/*instance ID*/

  You can find your instance ID on the **Logs** page in the **EC2 instance** column.

  You can find your environment ID in the **Server** section of the **Instances** configuration page.



Server

The following settings let you configure the environment servers. Learn more.

Instance type: t1.micro
Determines the processing power of the servers in your environment.

EC2 security groups: awseb-e-f92tc4fmek-stack-AWSEBSecurity
The names of the security groups (comma separated) that define firewall access to the launched EC2 instances.

EC2 key pair: (Optional) Select a key pair
Enables remote login to your instances.

Instance profile: aws-elasticbeanstalk-ec2-role
The instance profile grants your environment specific permissions under your AWS account. Learn more.

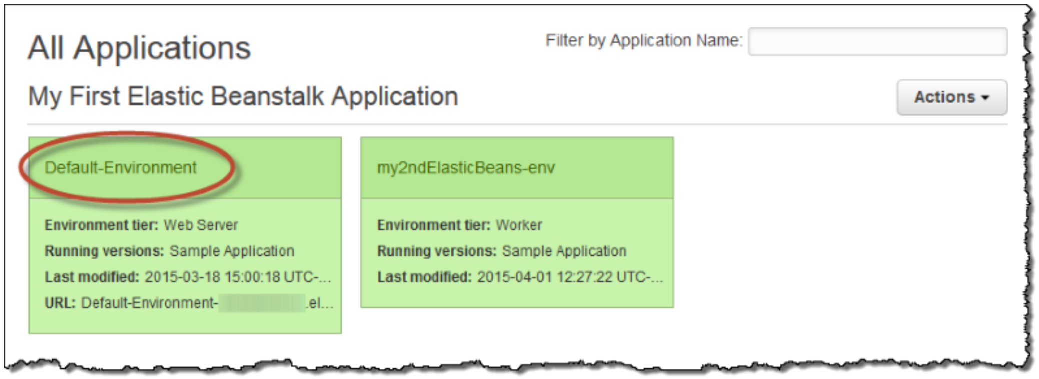
- **legacy** — Use the `RetrieveEnvironmentInfo` API to retrieve the location for the tail logs. For the CLI reference for this API, see elastic-beanstalk-retrieve-environment-info (p. 509). For the API reference, go to RetrieveEnvironmentInfo in the *AWS Elastic Beanstalk API Reference*.

Configuring Your Environment to Publish Logs to Amazon S3

You can configure your environment so that the logs from the Amazon EC2 instances running your applications are copied by Elastic Beanstalk to the Amazon S3 bucket associated with your application.

To configure your environment to publish logs to Amazon S3

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



All Applications

Filter by Application Name:

My First Elastic Beanstalk Application

Default-Environment
Environment tier: Web Server
Running versions: Sample Application
Last modified: 2015-03-18 15:00:18 UTC-...
URL: Default-Environment-[redacted].el...

my2ndElasticBeans-env
Environment tier: Worker
Running versions: Sample Application
Last modified: 2015-04-01 12:27:22 UTC-...

3. In the navigation pane, click **Configuration**.
4. On the **Configuration** page, next to **Software Configuration**, click the gear icon (⚙) to edit the container settings.

5. Under **Log Options**, select the **Enable log file rotation to Amazon S3** check box.

To access your logs

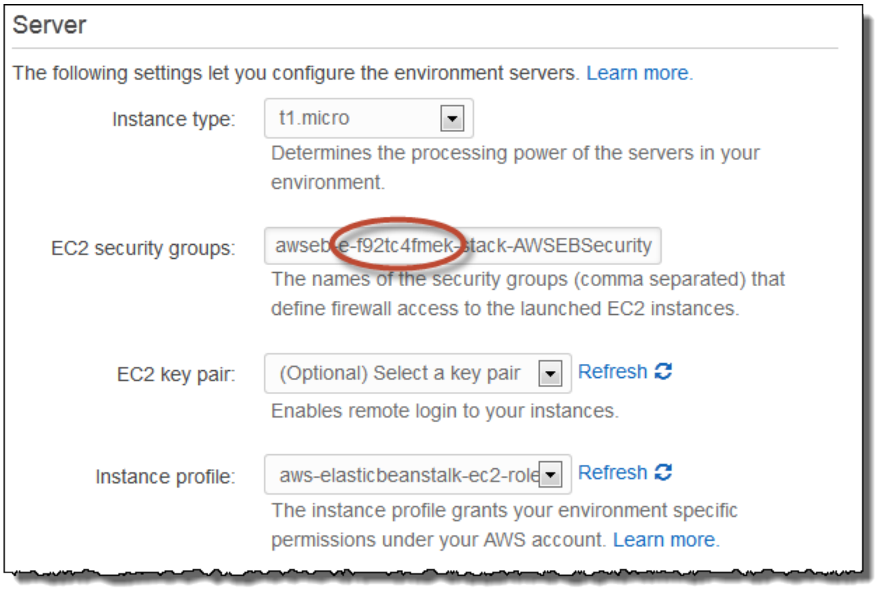
1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Depending on whether you are using a legacy or non-legacy container, navigate to one of the Amazon S3 locations. If you are not sure if you are using a legacy or non-legacy container, see To check if you are using a legacy container type (p. 92).

   - **legacy** — elasticbeanstalk-*region*-*account id*/*environment name*/logs/*instance ID*/

     You can find your instance ID on the **Logs** page in the **EC2 instance** column.
   - **non-legacy** — elasticbeanstalk-*region*-*account id*/resources/environments/logs/publish/*environment ID*/*instance ID*/

     You can find your instance ID on the **Logs** page in the **EC2 instance** column.

     You can find your environment ID in the **Server** section of the **Instances** configuration page.



Server

The following settings let you configure the environment servers. Learn more.

Instance type: t1.micro

Determines the processing power of the servers in your environment.

EC2 security groups: awseb-e-f92tc4fmek-stack-AWSEBSecurity

The names of the security groups (comma separated) that define firewall access to the launched EC2 instances.

EC2 key pair: (Optional) Select a key pair

Enables remote login to your instances.

Instance profile: aws-elasticbeanstalk-ec2-role

The instance profile grants your environment specific permissions under your AWS account. Learn more.

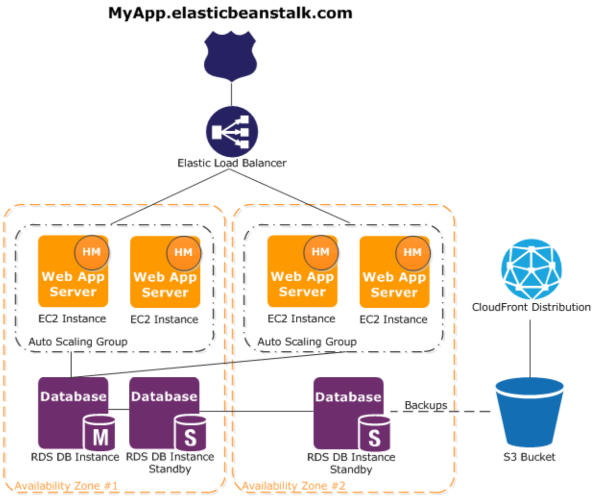
Using Elastic Beanstalk with Other AWS Services

Topics

This topic discusses the integration of Elastic Beanstalk with other AWS services.

Architectural Overview

The following diagram illustrates an example architecture of Elastic Beanstalk across multiple Availability Zones working with other AWS products such as Amazon CloudFront, Amazon Simple Storage Service (Amazon S3), and Amazon Relational Database Service (Amazon RDS). For a more detailed discussion about Amazon Route 53, Elastic Load Balancing, Amazon Elastic Compute Cloud (Amazon EC2) and host manager (HM), see Architectural Overview (p. 16).



To plan for fault-tolerance, it is advisable to have N+1 Amazon EC2 instances and spread your instances across multiple Availability Zones. In the unlikely case that one Availability Zone goes down, you will still have your other Amazon EC2 instances running in another Availability Zone. You can adjust Auto Scaling to allow for a minimum number of instances as well as multiple Availability Zones. For instructions on how to do this, see Launch Configuration (p. 178). For more information about building fault-tolerant applications, go to Building Fault-Tolerant Applications on AWS.

The following sections discuss in more detail integration with Amazon CloudFront, Amazon CloudWatch, Amazon DynamoDB Amazon ElastiCache, Amazon RDS, Amazon Route 53, Amazon Simple Storage Service, Amazon VPC , and IAM.

Using Elastic Beanstalk with Amazon CloudFront

Amazon CloudFront distributes your web content (such as images, video, and so on) using a network of edge locations around the world. End users are routed to the nearest edge location, so content is delivered with the best possible performance. CloudFront works seamlessly with Amazon S3. After you create and deploy your Elastic Beanstalk you can sign up for Amazon CloudFront and start using Amazon CloudFront to distribute your content. Create your distribution from a custom origin, and use an Elastic Beanstalk domain name. To get started using Amazon CloudFront, go to the Amazon CloudFront Developer Guide.

Using Elastic Beanstalk with AWS CloudTrail

AWS CloudTrail is an AWS service that logs the history of API calls from other AWS services. CloudTrail can identify which users and accounts called which APIs, each call's source IP address, and when the calls occurred. You turn on CloudTrail after you deploy your application to Elastic Beanstalk.

CloudTrail delivers log files to a new or existing Amazon S3 bucket. You can then use Amazon S3 to view the encrypted log files. You can store log files in your bucket indefinitely for as long as you want or define Amazon S3 lifecycle rules to archive or delete log files automatically.

Other AWS services can enhance your use of CloudTrail. For example, you can configure Amazon SNS to notify you when CloudTrail delivers new log files to your Amazon S3 bucket. Or use IAM to explicitly specify who can perform various CloudTrail tasks. These include creating, configuring, or deleting CloudTrail trails, starting and stopping logging, and accessing buckets with log files.

To get started using CloudTrail, go to the CloudTrail User Guide.

Using Elastic Beanstalk with Amazon CloudWatch

Amazon CloudWatch enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. Amazon CloudWatch monitoring enables you to collect, analyze, and view system and application metrics so that you can make operational and business decisions more quickly and with greater confidence. You can use Amazon CloudWatch to collect metrics about your Amazon Web Services (AWS) resources—such as the performance of your Amazon EC2 instances. You can also publish your own metrics directly to Amazon CloudWatch. Amazon CloudWatch alarms help you implement decisions more easily by enabling you to send notifications or automatically make changes to the resources you are monitoring, based on rules that you define. For example, you can create alarms that initiate Auto Scaling and Amazon Simple Notification Service (Amazon SNS) actions on your behalf. Elastic Beanstalk automatically uses Amazon CloudWatch to help you monitor your application and environment status. You can navigate to the Amazon CloudWatch console to see your dashboard and get an overview of all of your resources as well as your alarms. You can also choose to view more metrics or add custom metrics. For more information about Amazon CloudWatch, go to the Amazon CloudWatch Developer Guide. For an example of how to use Amazon CloudWatch with Elastic Beanstalk, see Example: Using Custom Amazon CloudWatch Metrics (p. 157).

For an example walkthrough using custom Amazon CloudWatch metrics, see Example: Using Custom Amazon CloudWatch Metrics (p. 157).

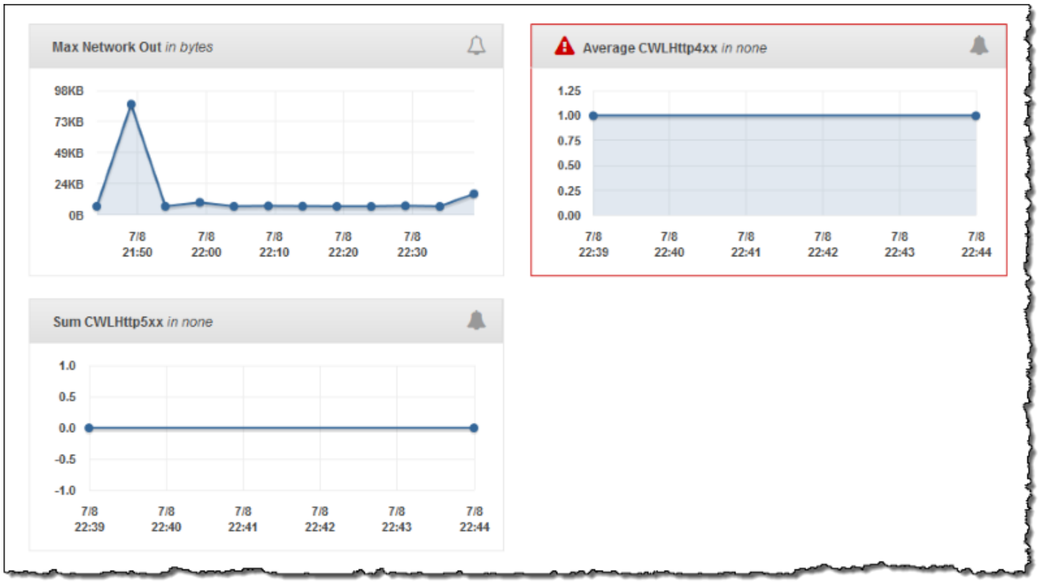
Using Elastic Beanstalk with Amazon CloudWatch Logs

You can integrate Elastic Beanstalk with Amazon CloudWatch Logs. CloudWatch Logs is available to environments in the following regions:

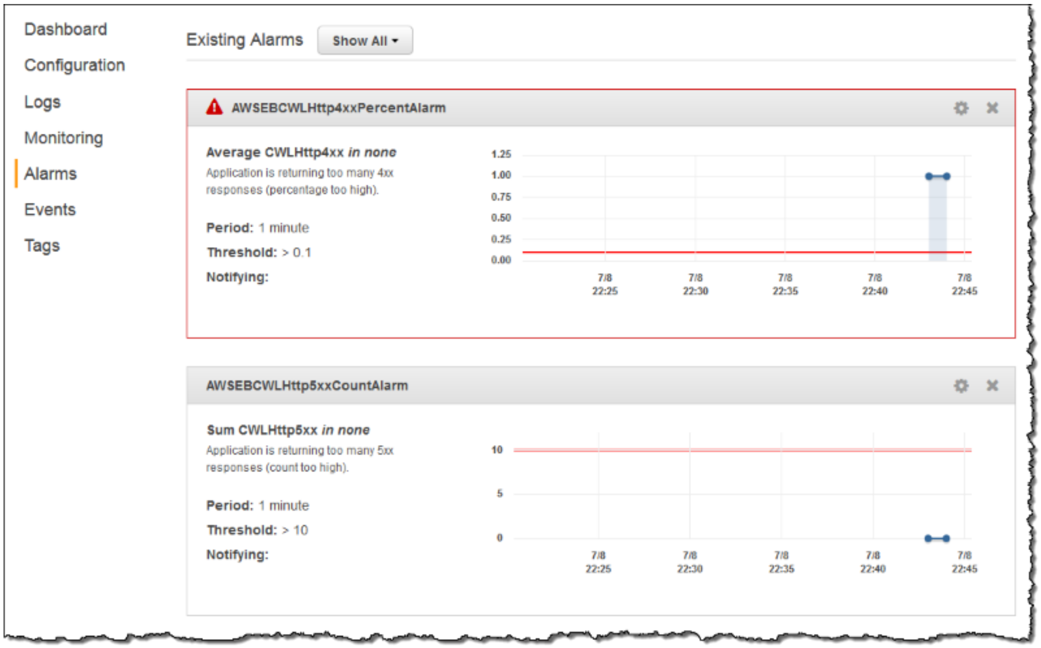
- US East (N. Virginia)
- US West (Oregon)
- EU (Ireland)
- EU (Frankfurt)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)

With CloudWatch Logs, you can monitor and archive your Elastic Beanstalk application, system, and custom log files. Furthermore, you can configure alarms that make it easier for you to take actions in response to specific log stream events that your metric filters extract. The CloudWatch Logs agent installed on each Amazon EC2 in your environment publishes metric data points to the CloudWatch service for each log group you configure. Each log group applies its own filter patterns to determine what log stream events to send to CloudWatch as data points. Log streams that belong to the same log group share the same retention, monitoring, and access control settings. For more information about CloudWatch Logs, including terminology and concepts, go to Monitoring System, Application, and Custom Log Files.

The following figure displays graphs on the **Monitoring** page for an environment that is configured with CloudWatch Logs integration. The example metrics in this environment are named **CWLHttp4xx** and **CWLHttp5xx**. In the image, the **CWLHttp4xx** metric has triggered an alarm according to conditions specified in the configuration files.



The following figure displays graphs on the **Alarms** page for the example alarms named **AWSEBCWLHttp4xxPercentAlarm** and **AWSEBCWLHttp5xxCountAlarm** that correspond to the **CWLHttp4xx** and **CWLHttp5xx** metrics, respectively.



AWSEBCWLHttp4xxPercentAlarm

Average CWLHttp4xx *in none*
Application is returning too many 4xx responses (percentage too high).

Period: 1 minute
Threshold: > 0.1
Notifying:

AWSEBCWLHttp5xxCountAlarm

Sum CWLHttp5xx *in none*
Application is returning too many 5xx responses (count too high).

Period: 1 minute
Threshold: > 10
Notifying:

Granting IAM Permissions for the CloudWatch Logs Agent

Before you can configure integration with CloudWatch Logs, you must set up IAM permissions to use with the CloudWatch Logs agent. You can attach the following custom policy to the instance profile (p. 20) that you assign to your environment:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:GetLogEvents",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:*:*"
      ]
    }
  ]
}
```

Setting Up CloudWatch Logs Integration with Configuration Files

When you create or update an environment, you can use the sample configuration files in the following list to set up and configure integration with CloudWatch Logs. You can include the `.zip` file that contains following configuration files or the extracted configuration files in the `.ebextensions` directory at the top level of your application source bundle. Use the appropriate files for the web server for your container type. For more information about the web server used by each container type, see Supported Platforms (p. 24).

> **Note**
> You cannot configure CloudWatch Logs integration with Elastic Beanstalk applications created in .NET containers.

You can download the configuration files at the following locations:

- Tomcat (Java) configuration files
- Apache (PHP and Python) configuration files
- Nginx (Ruby, Node.js, and Docker) configuration files

Each `.zip` file contains the following configuration files:

- `cwl-setup.config` – This file installs the CloudWatch Logs agent on each Amazon EC2 instance in your environment and then configures the agent. This file also creates the `general.conf` file when Elastic Beanstalk launches the instance. You can use the `cwl-setup.config` file without any modifications.

  If you prefer, you can manually set up the CloudWatch Logs agent on a new instance as explained in either Quick Start: Install and Configure the CloudWatch Logs Agent on a New EC2 Instance (for new instances) or Quick Start: Install and Configure the CloudWatch Logs Agent on an Existing EC2 Instance (for existing instances) in the *Amazon CloudWatch Developer Guide*.
- `cwl-webrequest-metrics.config` – This file specifies which logs the CloudWatch Logs agent monitors. The file also specifies the metric filters the CloudWatch Logs agent applies to each log that it monitors. Metric filters include filter patterns that map to the space-delimited entries in your log files.

(If you have custom logs, update or replace the example filter patterns in this example configuration file as needed.)

Metric filters also include metric transformations that specify what metric name and value to use when the CloudWatch Logs agent sends metric data points to the CloudWatch service. The CloudWatch Logs agent sends metric data points based on whether any entries in the web server access log file match the filter patterns.

Finally, the configuration file also includes an alarm action to send a message to an Amazon Simple Notification Service topic, if you created one for your environment, when the alarm conditions specified in the `cwl-setup.config` file are met. For more information about filter patterns, see Filter and Pattern Syntax in the *Amazon CloudWatch Developer Guide*. For more information about Amazon SNS, go to the Amazon Simple Notification Service Developer Guide. For more information about managing alarms from the Elastic Beanstalk management console, see Manage Alarms (p. 276).

Note
CloudWatch costs are applied to your AWS account for any alarms that you use.

- `eb-logs.config` – This file sets up the CloudWatch Logs log files for the CloudWatch Logs agent. This configuration file also ensures that log files are copied to Amazon S3 as part of log rotation. You can use this file without any modifications.

Troubleshooting CloudWatch Logs Integration

If Elastic Beanstalk cannot launch your environment when try to integrate Elastic Beanstalk with CloudWatch Logs, you can investigate the following common issues:

- Your IAM role lacks the required IAM permissions.
- You attempted to launch an environment in a region where CloudWatch Logs is not supported.
- Access logs do not exist at the path specified in the `cwl-webrequest-metrics.config` file (/var/log/httpd/elasticbeanstalk-access_log).

Using Elastic Beanstalk with DynamoDB

DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. If you are a developer, you can use DynamoDB to create a database table that can store and retrieve any amount of data, and serve any level of request traffic. DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance. All data items are stored on Solid State Disks (SSDs) and are automatically replicated across multiple Availability Zones in a Region to provide built-in high availability and data durability.

If you are a database administrator, you can launch a new DynamoDB database table, scale up or down your request capacity for the table without downtime or performance degradation, and gain visibility into resource utilization and performance metrics, all through the AWS Management Console. With DynamoDB, you can offload the administrative burdens of operating and scaling distributed databases to AWS, so you don’t have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

For more information about DynamoDB, go to the DynamoDB Developer Guide. For an example walkthrough using DynamoDB with Elastic Beanstalk, see Example: DynamoDB, CloudWatch, and SNS (p. 138). For information about a project that builds on the AWS SDK for Java to use DynamoDB as a session-state provider for Apache Tomcat applications, see Manage Tomcat Session State with DynamoDB in the AWS SDK for Java documentation.

Using Elastic Beanstalk with Amazon ElastiCache

Amazon ElastiCache is a web service that makes it easy to set up, manage, and scale distributed in-memory cache environments in the cloud. It provides a high performance, resizable, and cost-effective in-memory cache, while removing the complexity associated with deploying and managing a distributed cache environment. Amazon ElastiCache is protocol-compliant with Memcached, so the code, applications, and most popular tools that you use today with your existing Memcached environments will work seamlessly with the service. For more information about Amazon ElastiCache, go to the Amazon ElastiCache product page.

To use Elastic Beanstalk with Amazon ElastiCache

1. Create an ElastiCache cluster. For instructions on how to create an ElastiCache cluster, go to Create a Cache Cluster in the *Amazon ElastiCache Getting Started Guide*.
2. Configure your Amazon ElastiCache Security Group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to Authorize Access in the *Amazon ElastiCache Getting Started Guide*.

If you are using a non-legacy container, you can also use configuration files to customize your Elastic Beanstalk environment to use Amazon ElastiCache. For information on supported container types and customizing your environment, see Elastic Beanstalk Environment Configuration (p. 99). For an example snippet using Amazon ElastiCache with Elastic Beanstalk, see Example Snippets: ElastiCache (p. 130).

Using Elastic Beanstalk with Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you to focus on your applications and business.

If you plan to use Amazon RDS, it is advisable to configure Amazon RDS across multiple Availability Zones. This enables a synchronous standby replica of your database to be provisioned in a different Availability Zone. To keep both databases in sync, updates to your database instance are synchronously replicated across Availability Zones. In case of a failover scenario, the standby is promoted to be the primary and will handle the database operations. Running your database instance in multiple Availability Zones safeguards your data in the unlikely event of a database instance component failure or service health disruption in one Availability Zone.

The instructions for configuring your Elastic Beanstalk application with Amazon RDS depend on the programming language you use.

- Java — Using Amazon RDS and MySQL Connector/J (p. 592)
- .NET (SQL Server) — Tutorial: How to Deploy a .NET Sample Application Using AWS Elastic Beanstalk (p. 613)
- .NET (MySQL Server) — Using Amazon RDS (p. 646)
- Node.js — Using Amazon RDS with Node.js (p. 702)
- PHP — Using Amazon RDS with PHP (p. 729)
- Python — Using Amazon RDS with Python (p. 762)
- Ruby — Using Amazon RDS with Ruby (p. 789)

Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer

Due to the way the DNS protocol works, there is no way to refer your Elastic Load Balancer from the root (also known as the apex) of the domain. For instance, you can create a DNS entry that maps http://www.example.com to an Elastic Load Balancer, but you cannot do the same for http://example.com. Amazon Route 53 enables you to map the apex of a hosted zone to your Elastic Load Balancer using an Alias record. When Amazon Route 53 encounters an Alias record, it looks up the A records associated with the target DNS name in the Alias, and returns the IP addresses from that name. The following procedure steps you through how to use Amazon Route 53 to map your root domain to your Elastic Load Balancer.

To map your root domain and subdomains to your Elastic Load Balancing load balancer

1. Follow the Amazon Route 53 Getting Started Guide instructions to sign up for Route 53, create a hosted zone, and then update your name server records with your registrar.
2. Get the value of the hosted zone ID for your load balancer.

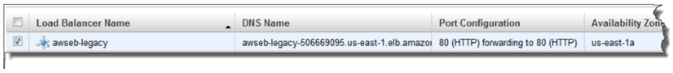
   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   b. From the region list, select a region.

   c. In the **Navigation** pane, click **Load Balancers** .

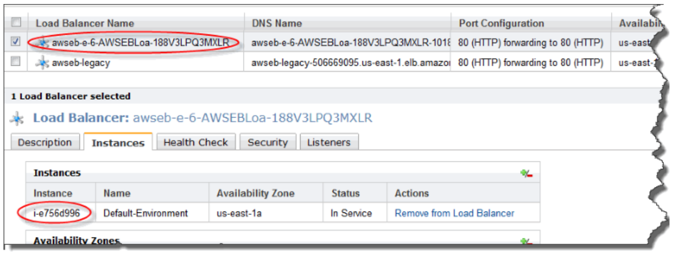
   d. Select the load balancer associated with your Elastic Beanstalk application. Depending on your container type, the load balancer will appear in one of the following formats:

   - Legacy container — **awseb-<*`your app environment name`*>**

□	Load Balancer Name	DNS Name	Port Configuration	Availability Zon
☑	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-1a

   - Nonlegacy container — contains **awseb** in the load balancer name. To verify you have the correct load balancer for your environment, check the instance name in the **Instances** tab. The instance name should be your environment name.

□	Load Balancer Name	DNS Name	Port Configuration	Availabil
☑	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR-101	80 (HTTP) forwarding to 80 (HTTP)	us-east
□	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-

   1 Load Balancer selected

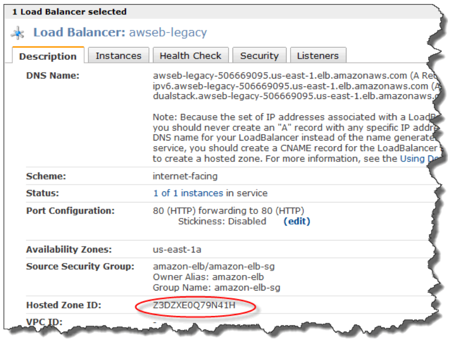
   Load Balancer: awseb-e-6-AWSEBLoa-188V3LPQ3MXLR

   Instances

Instance	Name	Availability Zone	Status	Actions
i-e756d996	Default-Environment	us-east-1a	In Service	Remove from Load Balancer

   Availability Zones

   Your hosted ID will appear in the **Load Balancer** details pane on the **Description** tab. Make a note of your hosted ID.



1 Load Balancer selected

Load Balancer: awseb-legacy

DNS Name:	awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A Re ipv6.awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A dualstack.awseb-legacy-506669095.us-east-1.elb.amazonaws. Note: Because the set of IP addresses associated with a LoadB you should never create an "A" record with any specific IP addre DNS name for your LoadBalancer instead of the name generate service, you should create a CNAME record for the LoadBalancer to create a hosted zone. For more information, see the Using D
Scheme:	internet-facing
Status:	1 of 1 instances in service
Port Configuration:	80 (HTTP) forwarding to 80 (HTTP) Stickiness: Disabled (edit)
Availability Zones:	us-east-1a
Source Security Group:	amazon-elb/amazon-elb-sg Owner Alias: amazon-elb Group Name: amazon-elb-sg
Hosted Zone ID:	Z3DZXE0Q79N41H
VPC ID:	-

3. Create alias resource record sets in your hosted zone for your root domain and subdomain. For instructions, go to How to Create an Alias Resource Record Set in the *Amazon Route 53 Developer Guide*.
4. Your root domain and subdomain are now mapped to your Elastic Beanstalk elastic load balancer. Type your domain name in your web browser to verify it worked.

If you rebuild your environment, launch a new environment, or swap your environment URL, you will need to map your root domain to the load balancer in your new environment.

To map your root domain to your new Elastic Load Balancer

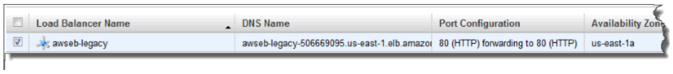
1. Get the value of the hosted zone ID for your old Elastic Load Balancer.

   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   b. In the navigation pane, click **Load Balancers**.

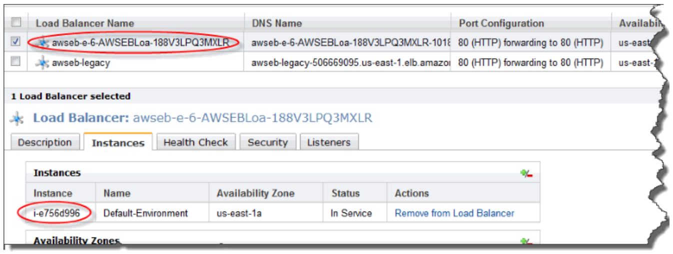
   c. Select the load balancer associated with your Elastic Beanstalk application. Depending on your container type, the load balancer will appear in one of the following formats:

   - Legacy container — **awseb-<*`your app environment name`*>**

□	Load Balancer Name	DNS Name	Port Configuration	Availability Zon
☑	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-1a

   - Non-legacy container — contains **awseb** in the load balancer name. To verify you have the correct load balancer for your environment, check the instance name in the **Instances** tab. The instance name should be your environment name.

□	Load Balancer Name	DNS Name	Port Configuration	Availabil
☑	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR	awseb-e-6-AWSEBLoa-188V3LPQ3MXLR-101	80 (HTTP) forwarding to 80 (HTTP)	us-east
□	awseb-legacy	awseb-legacy-506669095.us-east-1.elb.amazo	80 (HTTP) forwarding to 80 (HTTP)	us-east-

     1 Load Balancer selected

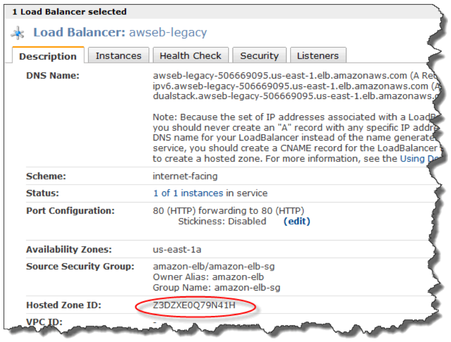
     Load Balancer: awseb-e-6-AWSEBLoa-188V3LPQ3MXLR

     Instances

Instance	Name	Availability Zone	Status	Actions
i-e756d996	Default-Environment	us-east-1a	In Service	Remove from Load Balancer

     Availability Zones

   Your hosted ID will appear in the **Load Balancer** details pane on the **Description** tab. Make a note of your hosted ID.



1 Load Balancer selected

Load Balancer: awseb-legacy

DNS Name:	awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A Re ipv6.awseb-legacy-506669095.us-east-1.elb.amazonaws.com (A dualstack.awseb-legacy-506669095.us-east-1.elb.amazonaws. Note: Because the set of IP addresses associated with a LoadB you should never create an "A" record with any specific IP addre DNS name for your LoadBalancer instead of the name generate service, you should create a CNAME record for the LoadBalancer to create a hosted zone. For more information, see the Using D
Scheme:	internet-facing
Status:	1 of 1 instances in service
Port Configuration:	80 (HTTP) forwarding to 80 (HTTP) Stickiness: Disabled (edit)
Availability Zones:	us-east-1a
Source Security Group:	amazon-elb/amazon-elb-sg Owner Alias: amazon-elb Group Name: amazon-elb-sg
Hosted Zone ID:	Z3DZXE0Q79N41H
VPC ID:	-

2. Change an alias resource record set in your hosted zone. For instructions, go to Creating, Changing, and Deleting Resource Record Sets Using the Route 53 Console in the *Amazon Route 53 Developer Guide*.
3. Your root domain and subdomain are now mapped to your Elastic Beanstalk elastic load balancer. Type your domain name in your web browser to verify it worked.
4. After you have confirmed that the changes completed successfully, you can terminate your old environment. For instructions on how to terminate an environment, see Terminate an Environment (p. 95).

Using Elastic Beanstalk with Amazon S3

Amazon S3 is a simple web service that provides highly durable, fault-tolerant data storage. Behind the scenes, Amazon S3 stores objects redundantly on multiple devices across multiple facilities in an Amazon S3 Region. In the unlikely event of a failure in an Amazon Web Service data center, you will still have access to your data. Elastic Beanstalk automatically signs you up for Amazon S3 when you sign up for Elastic Beanstalk. When you create your application and deploy it to Elastic Beanstalk your application will be automatically uploaded to an Amazon S3 bucket. To learn more about Amazon S3, go to the Amazon Simple Storage Service (Amazon S3) product page. For code samples that demonstrate how to retrieve objects directly from S3, see Getting Objects.

Using Elastic Beanstalk with Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) enables you to define a virtual network in your own isolated section within the Amazon Web Services (AWS) cloud, known as a *virtual private cloud (VPC)*. Using VPC, you can deploy a new class of web applications on Elastic Beanstalk, including internal web applications (such as your recruiting application), web applications that connect to an on-premise database (using a VPN connection), as well as private web service back-ends. Elastic Beanstalk launches your AWS resources, such as instances, into your VPC. Your VPC closely resembles a traditional network, with the benefits of using AWS's scalable infrastructure. You have complete control over your VPC; you can select the IP address range, create subnets, and configure routes and network gateways. To protect the resources in each subnet, you can use multiple layers of security, including security groups and network access control lists. For more information about Amazon VPC, go to the Amazon VPC User Guide.

What VPC Configurations Do I Need?

When you use Amazon VPC with Elastic Beanstalk, you can launch Elastic Beanstalk resources, such as Amazon EC2 instances, in a public or private subnet. The subnets that you require depend on your Elastic Beanstalk application environment type and whether the resources you launch are public or private. The following scenarios discuss sample VPC configurations that you might use for a particular environment.

This feature is not supported by **legacy** containers. See To check if you are using a legacy container type (p. 92) if you are unsure if you are on a legacy configuration or not.

Single-instance environments

For single-instance environments, Elastic Beanstalk assigns an Elastic IP address (a static, public IP address) to the instance so that it can communicate directly with the Internet. No additional network interface, such as a network address translator (NAT), is required for a single-instance environment.

If you have a single-instance environment without any associated private resources, such as a back-end Amazon RDS DB instance, create a VPC with one public subnet and include the instance in that subnet. For more information, see Example: Launching a Single-Instance Environment without Any Associated Private Resources in a VPC (p. 299).

If you have resources that you don't want public, create a VPC with one public subnet and one private subnet. Add all your public resources like the single Amazon EC2 instance in the public subnet, and add private resources like a back-end Amazon RDS DB instance in the private subnet. If you do launch an Amazon RDS DB instance in a VPC, you must create at least two different private subnets that are in different Availability Zones (an Amazon RDS requirement).

Load-balancing, autoscaling environments

For load-balancing, autoscaling environments, you can either create a public and private subnet for your VPC or use a single public subnet. In the case of a load-balancing, autoscaling environment with both a public and private subnet, Amazon EC2 instances in the private subnet require Internet connectivity. Consider the following scenarios.

If you want your Amazon EC2 instances to have a private IP address, create a public and private subnet for your VPC in each Availability Zone (an Elastic Beanstalk requirement). Then add your public resources, like the load balancer and NAT, to the public subnet. That way, Elastic Beanstalk assigns them unique Elastic IP addresses (a static, public IP address). Launch your Amazon EC2 instances in the private subnet so that Elastic Beanstalk assigns them nonrouteable private IP addresses.

Without a public IP address, an Amazon EC2 instance can't directly communicate with the Internet. Although Amazon EC2 instances in a private subnet can't send outbound traffic by default, neither can they receive unsolicited inbound connections from the Internet.

To enable communication between the private subnet and the public subnet and the Internet beyond the public subnet, create routing rules that do the following:

- Route all inbound traffic to an Amazon EC2 instance through a load balancer.
- Route all outbound traffic from an Amazon EC2 instance through a NAT.

If you have associated resources that are private, such as a back-end Amazon RDS DB instance, launch the DB instance in the private subnet. If you do launch an Amazon RDS DB instance in a VPC, you must create at least two different private subnets that are in different Availability Zones (an Amazon RDS requirement). For more information, see Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS (p. 315).

If you don't have any private resources associated with your Amazon EC2 instances, you can create a single public subnet for your VPC. If you want to use a single public subnet, you must choose the **Associate Public IP Address** option to add the load balancer and your Amazon EC2 instances to the public subnet. Elastic Beanstalk assigns an public IP address to each Amazon EC2 instance and eliminates the need for a NAT for the instances to communicate with the Internet.

For more information, see Example: Launching a Load-Balancing, Autoscaling Environment with Public and Private Resources in a VPC (p. 302).

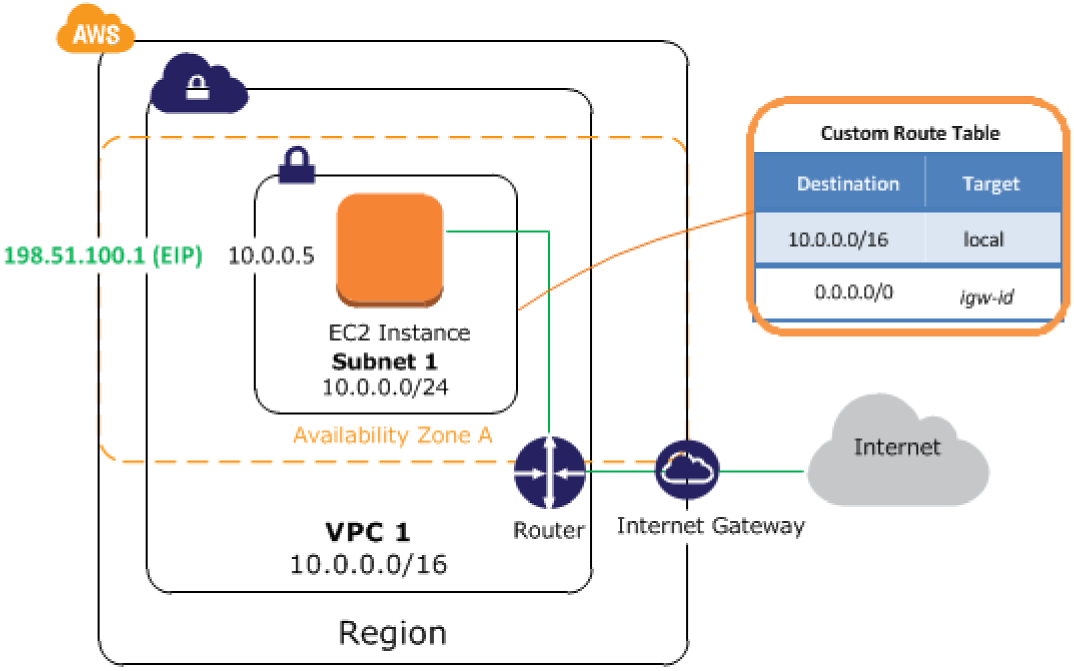
If you require direct access to your Amazon EC2 instances in a private subnet (for example, if you want to use SSH to sign in to an instance), create a bastion host in the public subnet that proxies requests from the Internet. From the Internet, you can connect to your instances by using the bastion host. For more information, see Example: Launching an Elastic Beanstalk Application in a VPC with Bastion Hosts (p. 308).

Extend your own network into AWS

If you want to extend your own network into the cloud and also directly access the Internet from your VPC, create a VPN gateway. For instructions on creating a VPN Gateway, see Scenario 3: VPC with Public and Private Subnets and Hardware VPN Access.

Example: Launching a Single-Instance Environment without Any Associated Private Resources in a VPC

You can deploy an Elastic Beanstalk application in a public subnet. Use this configuration if you just have a single instance without any private resources that are associated with the instance, such as an Amazon RDS DB instance that you don't want publicly accessible. Elastic Beanstalk assigns an Elastic IP address to the instance so that it can access the Internet through the VPC Internet gateway.



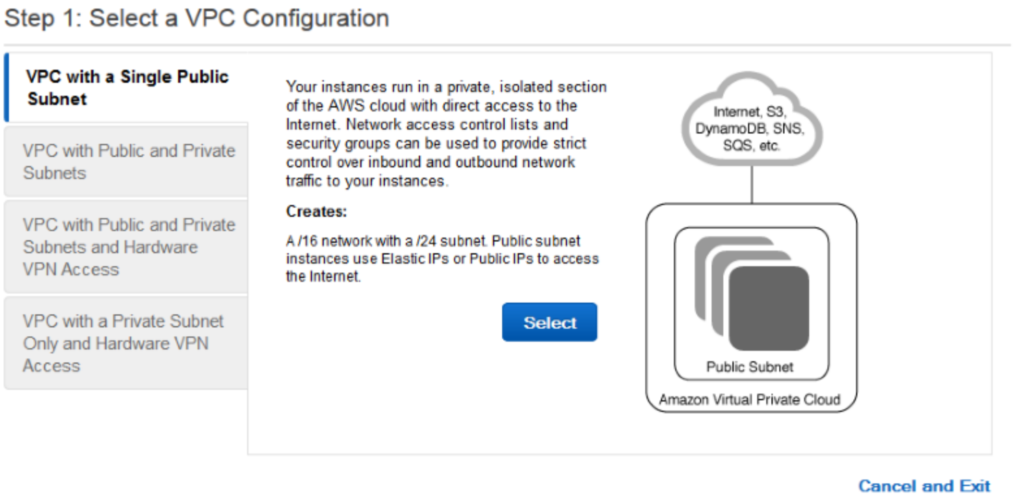
Custom Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Step 1: Create a VPC with a Public Subnet

To create a VPC

1. Sign in to the Amazon VPC console.
2. In the navigation pane, click **VPC Dashboard**, locate the **Resources** area, and then click **Start VPC Wizard**.
3. Select **VPC with a Single Public Subnet** and then click **Select**.



Step 1: Select a VPC Configuration

VPC with a Single Public Subnet

VPC with Public and Private Subnets

VPC with Public and Private Subnets and Hardware VPN Access

VPC with a Private Subnet Only and Hardware VPN Access

Your instances run in a private, isolated section of the AWS cloud with direct access to the Internet. Network access control lists and security groups can be used to provide strict control over inbound and outbound network traffic to your instances.

Creates:

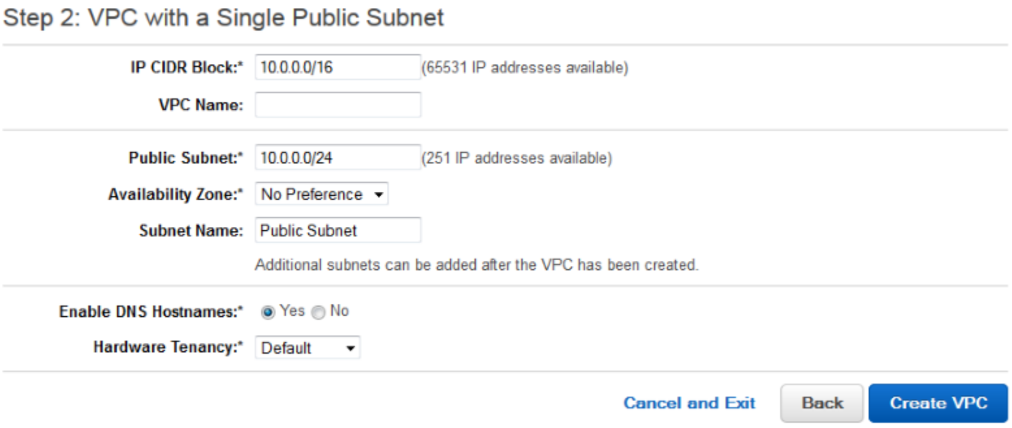
A /16 network with a /24 subnet. Public subnet instances use Elastic IPs or Public IPs to access the Internet.

Select
Internet, S3, DynamoDB, SNS, SQS, etc.
Public Subnet
Amazon Virtual Private Cloud
Cancel and Exit

The next page shows the CIDR block used for the VPC and subnet, the subnet name, and the Availability Zone associated with the subnet. There is no default name for a VPC, but you can specify one. On this page, you can also enable DNS hostnames and choose the instance tenancy attribute for instances that are launched in the VPC

> **Note**
> If you choose **Default** instance tenancy, instances run on shared hardware. If you choose **Dedicated** instance tenancy, instances run on single-tenant hardware. You can't change the instance tenancy of a VPC after you create it.

.



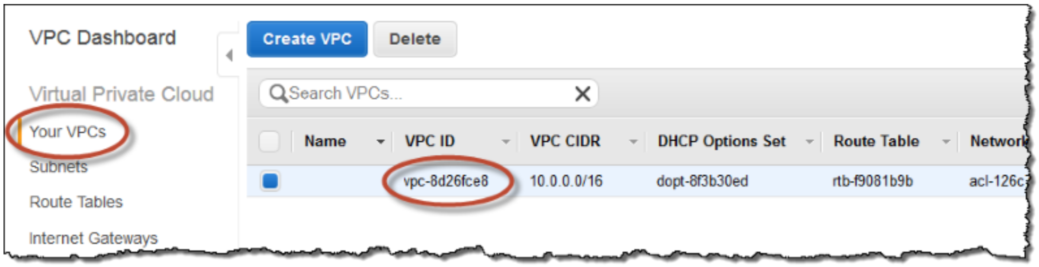
Step 2: VPC with a Single Public Subnet

IP CIDR Block:*	10.0.0.0/16 (65531 IP addresses available)
VPC Name:	
Public Subnet:*	10.0.0.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Subnet Name:	Public Subnet
	Additional subnets can be added after the VPC has been created.
Enable DNS Hostnames:*	● Yes ○ No
Hardware Tenancy:*	Default

4. After you review settings and make changes as needed, click **Create VPC**.

   AWS creates your VPC, subnet, Internet gateway, and route table. Click **Close** to end the wizard.

   The VPC is assigned a VPC ID after AWS successfully creates it. You need the VPC ID for the next step. To view your VPC ID, click **Your VPCs** in the navigation pane of the Amazon VPC console.



Name	VPC ID	VPC CIDR	DHCP Options Set	Route Table	Network
	vpc-8d26fce8	10.0.0.0/16	dopt-8f3b30ed	rtb-f9081b9b	acl-126c

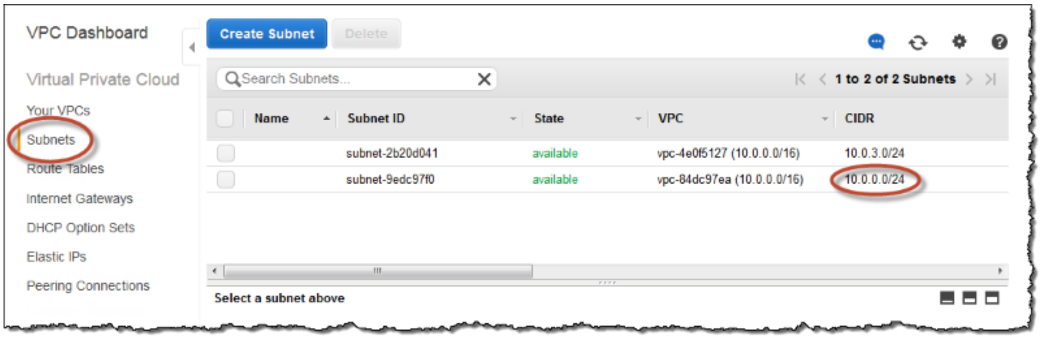
Step 3: Deploy to Elastic Beanstalk

After you set up your VPC, you can deploy your application to Elastic Beanstalk and create your environment inside your VPC. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, eb, API-based command line interface (CLI), or API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS Toolkit, eb, CLI, or API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk Console

When you create an Elastic Beanstalk application or launch an environment, the Elastic Beanstalk console walks you through creating your environment inside a VPC. For more information, see Create an Application (p. 34).

You'll need to select the VPC ID and subnet ID for your instance. By default, VPC creates a public subnet using 10.0.0.0/24. You can view your subnet IDs by clicking **Subnets** in the Amazon VPC console.



Name	Subnet ID	State	VPC	CIDR
	subnet-2b20d041	available	vpc-4e0f5127 (10.0.0.0/16)	10.0.3.0/24
	subnet-9edc97f0	available	vpc-84dc97ea (10.0.0.0/16)	10.0.0.0/24

Deploying with the AWS Toolkits, Eb, CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, eb, CLI, or API, you need to specify your VPC option settings in a file and deploy it with your source bundle. To deploy your application using a toolkit or eb, create a configuration file with a `.config` extension (e.g., `myconfig.config`) and place it inside a directory named `.ebextensions` in the top-level directory of your source bundle. If you use the CLI or API, you can specify these settings in a file name and pass the file name in as a parameter. Deploy your application to Elastic Beanstalk using one of the following methods:

- Java — Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576)
- The EB CLI — Tools (p. 384)

- CLI or API — Create an Application (p. 34) (see the CLI or API section)
- CLI or API — Launching a New AWS Elastic Beanstalk Environment (p. 55) (see the CLI or API section)

When you create your configuration file, you will need to specify at least the following:

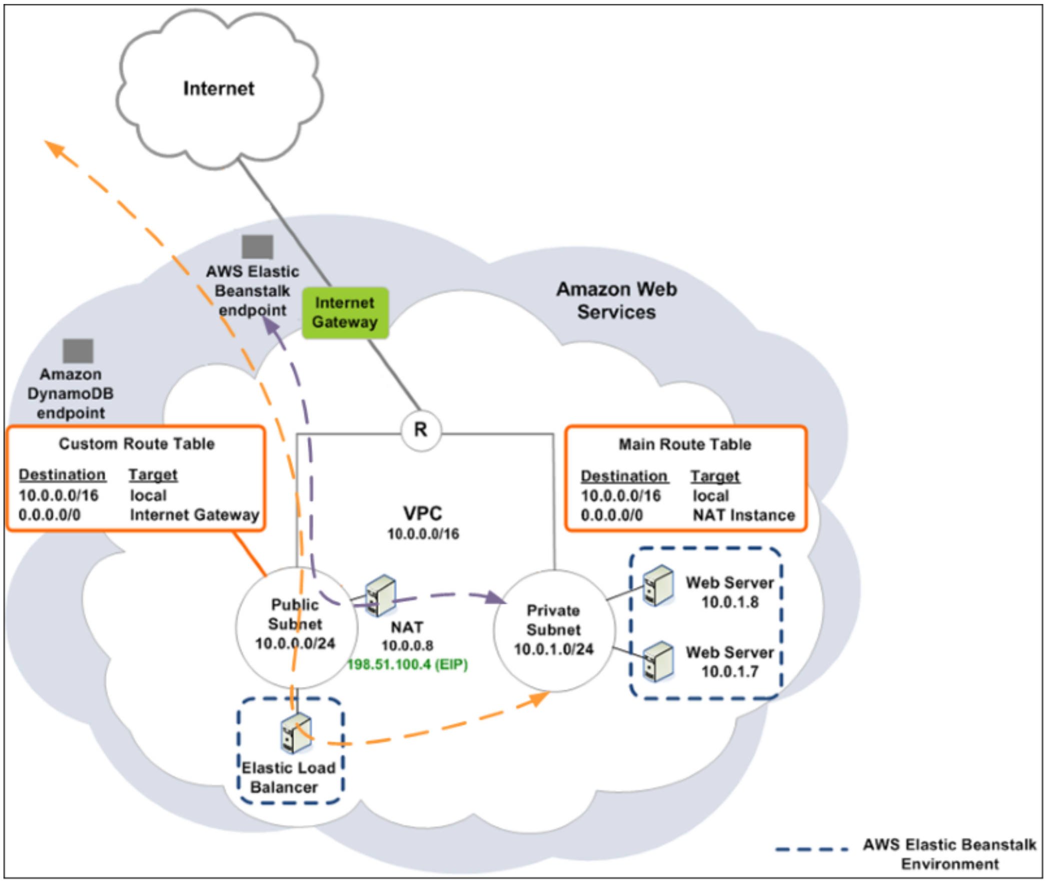
- **VPCId** — Contains the ID of the VPC.
- **Subnets** — Contains the ID of the public subnet that contains the instance.

The following is an example of the settings you could set when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration Options (p. 103).

```
option_settings:
  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-170647c

  - namespace: aws:ec2:vpc
    option_name: Subnets
    value: subnet-4f195024
```

Example: Launching a Load-Balancing, Autoscaling Environment with Public and Private Resources in a VPC

You can deploy an Elastic Beanstalk application in a load balancing, autoscaling environment in a VPC that has both a public and private subnet. Use this configuration if you want Elastic Beanstalk to assign private IP addresses to your Amazon EC2 instances. In this configuration, the Amazon EC2 instances in the private subnet require a load balancer and a network address translation (NAT) instance in the public subnet. The load balancer routes inbound traffic from the Internet to the Amazon EC2 instances. You need to launch a NAT instance to route outbound traffic from the Amazon EC2 instances to the Internet. You also need to configure the default VPC security group to allow traffic from the Amazon EC2 instances to the NAT instance. Your infrastructure will look similar to the following diagram.



To deploy an Elastic Beanstalk application inside a VPC using a NAT instance, you need to do the following:

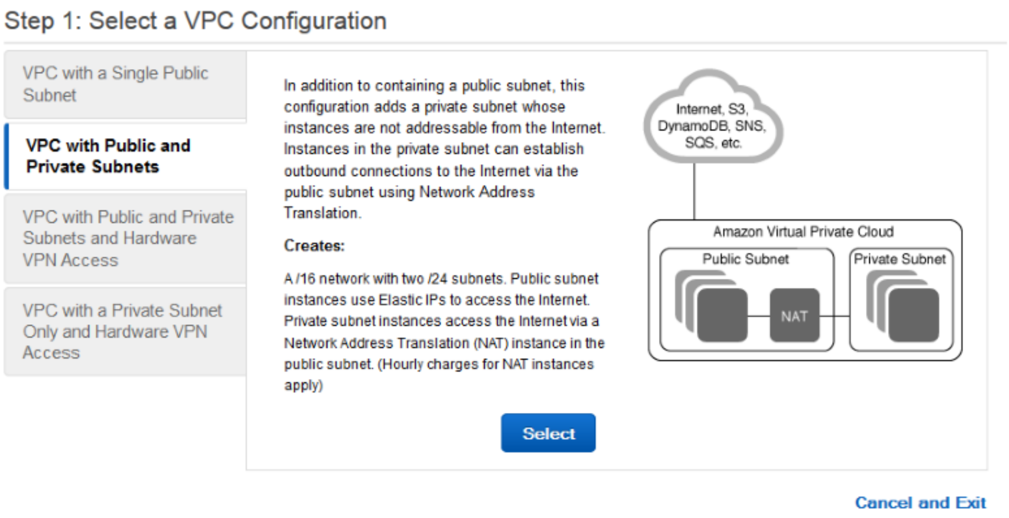
- Step 1: Create a VPC with a Public and Private Subnet (p. 303)
- Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 305)
- Step 3: Deploy to Elastic Beanstalk (p. 306)

Step 1: Create a VPC with a Public and Private Subnet

You can use the Amazon VPC console to create a VPC.

To create a VPC

1. Sign in to the Amazon VPC console.
2. In the navigation pane, click **VPC Dashboard**. Then click **Start VPC Wizard**.
3. Click **VPC with Public and Private Subnets** and then click **Select**.



Step 1: Select a VPC Configuration

- VPC with a Single Public Subnet
- **VPC with Public and Private Subnets**
- VPC with Public and Private Subnets and Hardware VPN Access
- VPC with a Private Subnet Only and Hardware VPN Access

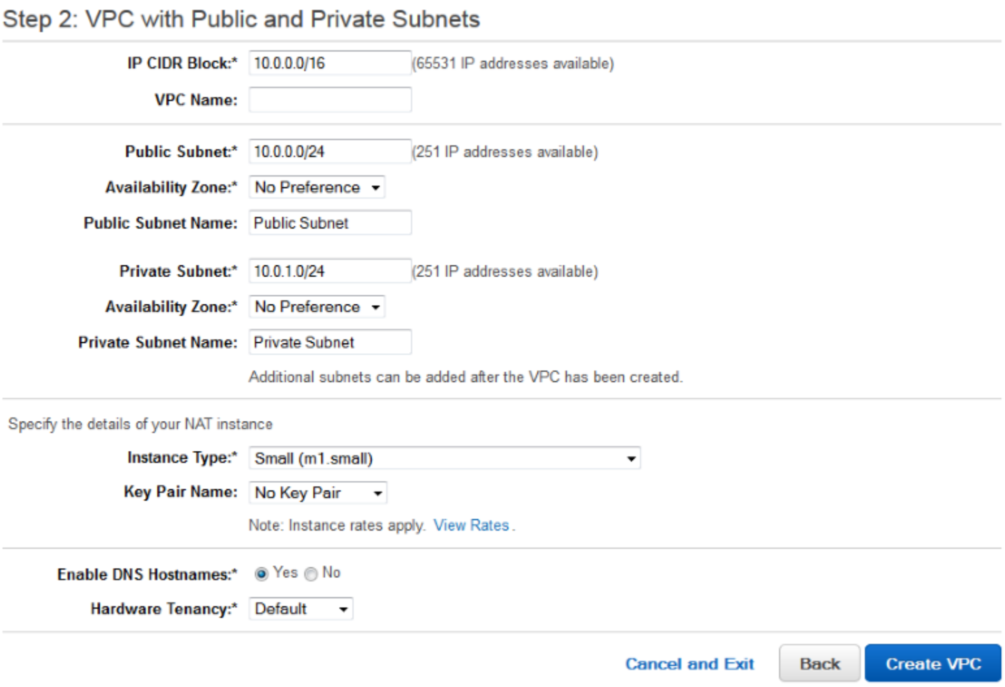
In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation.

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via a Network Address Translation (NAT) instance in the public subnet. (Hourly charges for NAT instances apply)

Internet, S3, DynamoDB, SNS, SQS, etc.
Amazon Virtual Private Cloud
Public Subnet
Private Subnet
NAT
Select
Cancel and Exit

4. Your Elastic Load Balancing load balancer and your Amazon EC2 instances must be in the same Availability Zone so they can communicate with each other. Choose the same Availability Zone from each **Availability Zone** list.



Step 2: VPC with Public and Private Subnets

IP CIDR Block:*	10.0.0.0/16 (65531 IP addresses available)
VPC Name:	
Public Subnet:*	10.0.0.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Public Subnet Name:	Public Subnet
Private Subnet:*	10.0.1.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Private Subnet Name:	Private Subnet
	Additional subnets can be added after the VPC has been created.

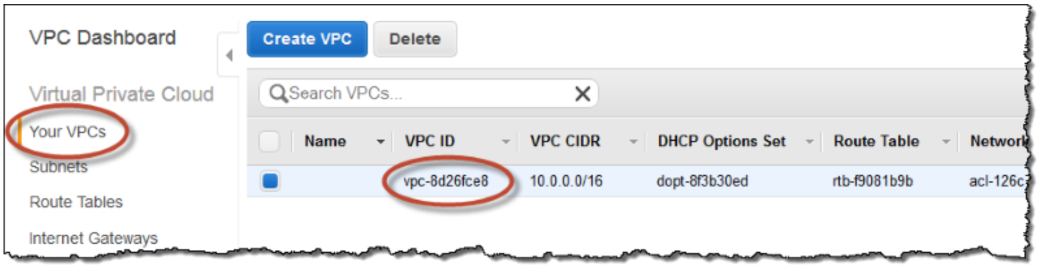
Specify the details of your NAT instance

Instance Type:*	Small (m1.small)
Key Pair Name:	No Key Pair
	Note: Instance rates apply. View Rates.
Enable DNS Hostnames:*	● Yes ○ No
Hardware Tenancy:*	Default

5. Click **Create VPC**.

   The wizard begins to create your VPC, subnets, and Internet gateway. It also updates the main route table and creates a custom route table. Finally, the wizard launches a NAT instance in the public subnet and prepares it for use. This preparation includes disabling the source/destination check on the instance and assigning the instance an Elastic IP address.

After the VPC is successfully created, you will get a VPC ID. You will need this for this for the next step. To view your VPC ID, click **Your VPCs** in the left pane of the Amazon VPC console.



	Name	VPC ID	VPC CIDR	DHCP Options Set	Route Table	Network
■		vpc-8d26fce8	10.0.0.0/16	dopt-8f3b30ed	rtb-f9081b9b	acl-126c

Step 2: Configure the Default VPC Security Group for the NAT Instance

You will need to update the default VPC security group to allow traffic from the EC2 instances to the NAT instance. To do this, you will first create a new security group, and then update your default security group to allow traffic from this new security group.

To create a security group

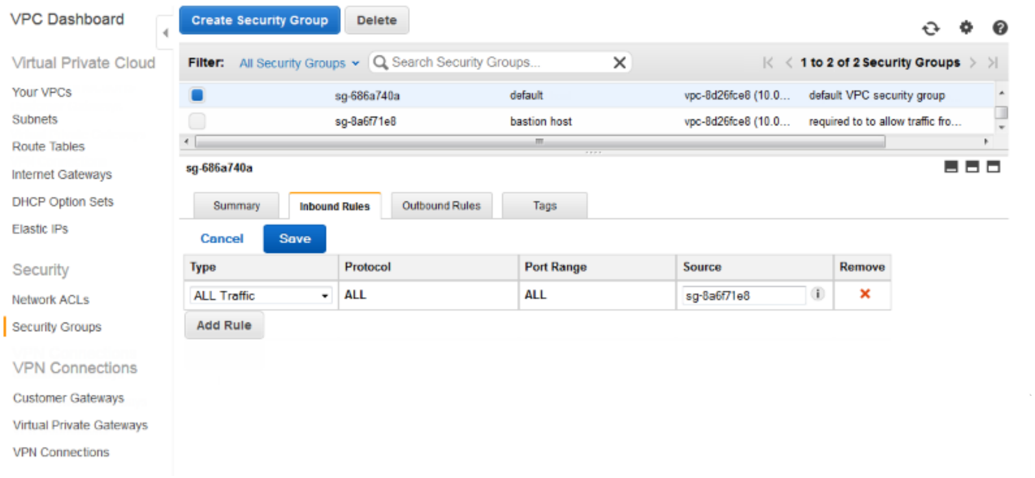
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, click **Security Groups**.
3. Click **Create Security Group**.
4. Enter the security group name, a description of the group, and select the ID for your VPC. You may also enter a tag for the group (optional). Then click **Yes, Create**.

   The security group is created and appears on the **Security Groups** page. Notice that it has an ID (e.g., sg-xxxxxxxx). You might have to turn on the **Group ID** column by clicking **Show/Hide** in the top right corner of the page.

To update the default VPC security group for the NAT instance

1. In the list of security groups, select the check box for the default VPC security group for the NAT instance.
2. Add a rule to allow all traffic access to the group from the security group you just created.

   a. On the **Inbound Rules** tab, click **Edit**.
   b. Under **Type**, select **All Traffic**.
   c. Under **Source**, type the ID (e.g., sg-xxxxxxxx) of the security group, and then click **Save**.



☑	sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group
□	sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to to allow traffic fro...

sg-686a740a

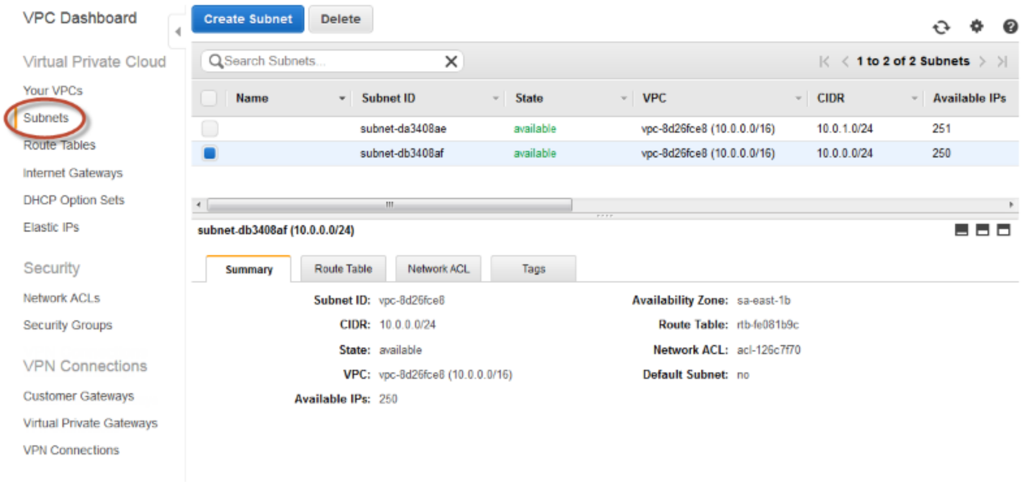
Type	Protocol	Port Range	Source	Remove
ALL Traffic	ALL	ALL	sg-8a6f71e8	✕

Step 3: Deploy to Elastic Beanstalk

After you set up your VPC, you can deploy your application to Elastic Beanstalk and create your environment inside your VPC. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, eb, API-based command line interface (CLI), or API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS Toolkit, eb, CLI, or API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk Console

The Elastic Beanstalk console walks you through creating your new environment inside your VPC. You need to provide a `.war` file (for Java applications) or a `.zip` file for all other applications. The Elastic Beanstalk console asks you for the IDs for your VPC and VPC security group you created in the previous steps as well as the subnet IDs for your load balancer and EC2 instances. If you created a private subnet for your EC2 instances and a public subnet for your load balancer, make sure you select the public subnet ID for the load balancer, and the private subnet ID for your EC2 instances. By default, VPC creates a public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your subnet IDs by clicking **Subnets** in the Amazon VPC console.



Name	Subnet ID	State	VPC	CIDR	Available IPs
	subnet-da3408ae	available	vpc-8d26fce8 (10.0.0.0/16)	10.0.1.0/24	251
	subnet-db3408af	available	vpc-8d26fce8 (10.0.0.0/16)	10.0.0.0/24	250

subnet-db3408af (10.0.0.0/24)

Subnet ID: vpc-8d26fce8
CIDR: 10.0.0.0/24
State: available
VPC: vpc-8d26fce8 (10.0.0.0/16)
Available IPs: 250

Availability Zone: sa-east-1b
Route Table: rtb-fe081b9c
Network ACL: acl-126c7f70
Default Subnet: no

Deploying with the AWS Toolkits, Eb, CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, eb, CLI, or API, you need to specify your VPC option settings in a file and deploy it with your source bundle. To deploy your application using a toolkit or eb, create a configuration file with a `.config` extension (e.g., `myconfig.config`) and place it inside a directory named `.ebextensions` in the top-level directory of your source bundle. If you use the CLI or API, you can specify these settings in a file name and pass the file name in as a parameter. Deploy your application to Elastic Beanstalk using one of the following methods:

- Java — Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576)
- The EB CLI — Tools (p. 384)
- CLI or API — Create an Application (p. 34) (see the CLI or API section)
- CLI or API — Launching a New AWS Elastic Beanstalk Environment (p. 55) (see the CLI or API section)

When you create your configuration file with your option settings, you will need to specify at least the following:

- **VPCId** — Contains the ID of the VPC.
- **Subnets** — Contains the ID of the Auto Scaling group subnet. In this example, this is the ID of the private subnet.
- **ELBSubnets** — Contains the ID of the subnet for the elastic load balancer. In this example, this is the ID of the public subnet.
- **SecurityGroups** — Contains the ID of the security groups. In this example, you'll use the ID of the new security group you created in Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 305).

Optionally, you can also specify the following information:

- **ELBScheme** — Specify `internal` if you want to create an internal load balancer inside your VPC so that your Elastic Beanstalk application cannot be accessed from outside your VPC.
- **DBSubnets** — Contains the ID of the DB subnets. This is only used if you want to add an Amazon RDS DB Instance as part of your application. For an example, see Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS (p. 315).

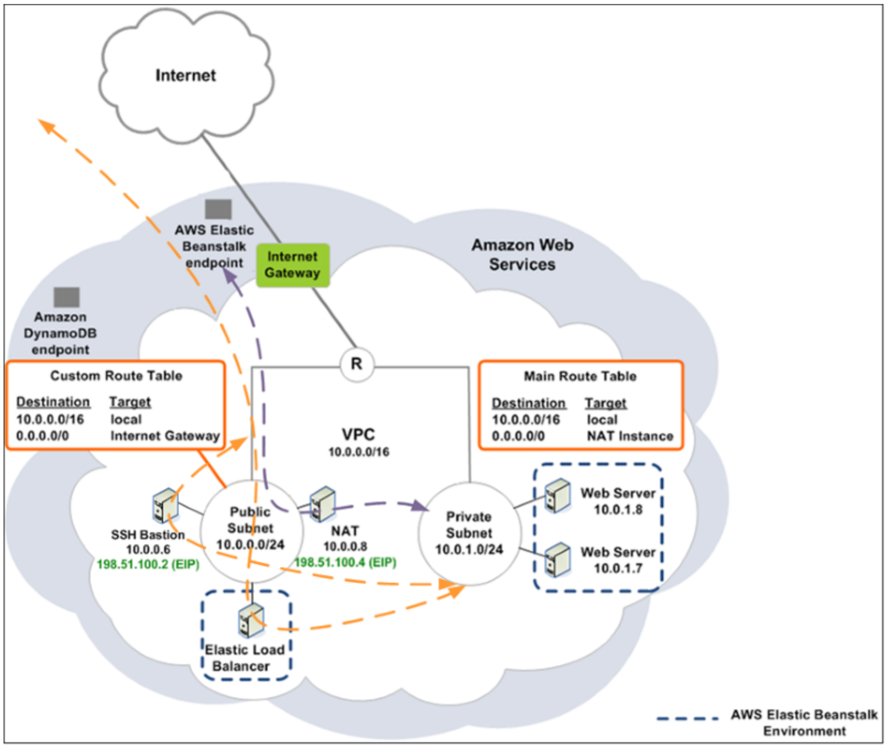
Note
When using DBSubnets, you need to create additional subnets in your VPC to cover all the Availability Zones in the region.

The following is an example of the option settings you could set when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration Options (p. 103).

```
option_settings:
  - namespace: aws:autoscaling:launchconfiguration
    option_name: EC2KeyName
    value: ec2keypair

  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-170647c

  - namespace: aws:ec2:vpc
    option_name: Subnets
    value: subnet-4f195024

  - namespace: aws:ec2:vpc
    option_name: ELBSubnets
    value: subnet-fe064f95

  - namespace: aws:autoscaling:launchconfiguration
    option_name: InstanceType
    value: m1.small

  - namespace: aws:autoscaling:launchconfiguration
    option_name: SecurityGroups
    value: sg-7f1ef110
```

Example: Launching an Elastic Beanstalk Application in a VPC with Bastion Hosts

If your Amazon EC2 instances are located inside the private subnet, you will not be able to connect to them directly. To connect to your instances, you need to create and connect to a bastion host in your public subnet. This section provides an example of how to create a VPC with a private and public subnet. The instances are located inside the private subnet, and the bastion host, NAT instance, and Elastic Load Balancing load balancer are located inside the public subnet. Your infrastructure will look similar to the following diagram:



In this walkthrough, you'll add a bastion host to your VPC.

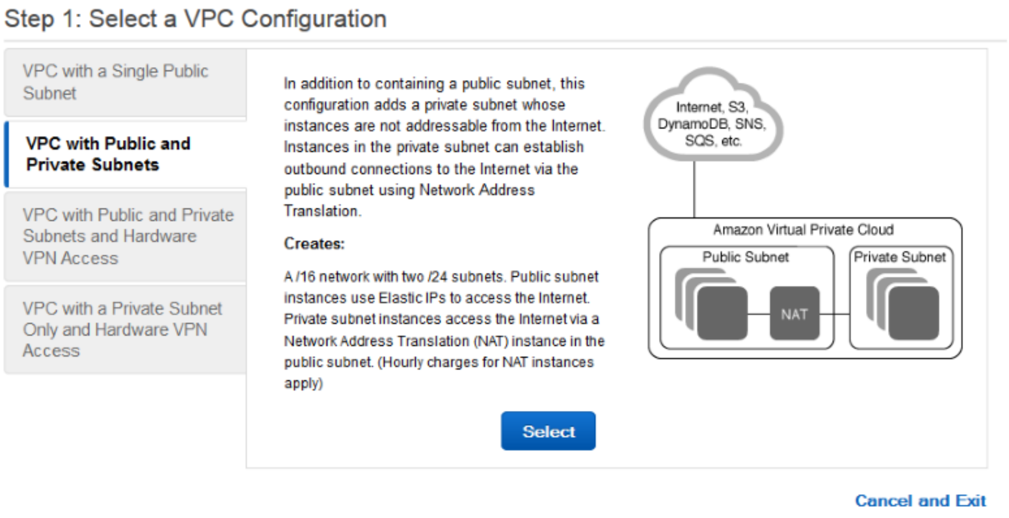
1. Step 1: Create a VPC with a Public and Private Subnet (p. 309)
2. Step 2: Configure the Security Groups (p. 311)
3. Step 3: Create a Bastion Host (p. 313)
4. Step 4: Define the Option Settings (p. 314)
5. Step 5: Deploy to Elastic Beanstalk (p. 315)

Step 1: Create a VPC with a Public and Private Subnet

You can use the Amazon VPC console to create a VPC.

To create a VPC

1. Sign in to the Amazon VPC console.
2. In the navigation pane, click **VPC Dashboard**. Then click **Start VPC Wizard**.
3. Click **VPC with Public and Private Subnets** and then click **Select**.



Step 1: Select a VPC Configuration

- VPC with a Single Public Subnet
- **VPC with Public and Private Subnets**
- VPC with Public and Private Subnets and Hardware VPN Access
- VPC with a Private Subnet Only and Hardware VPN Access

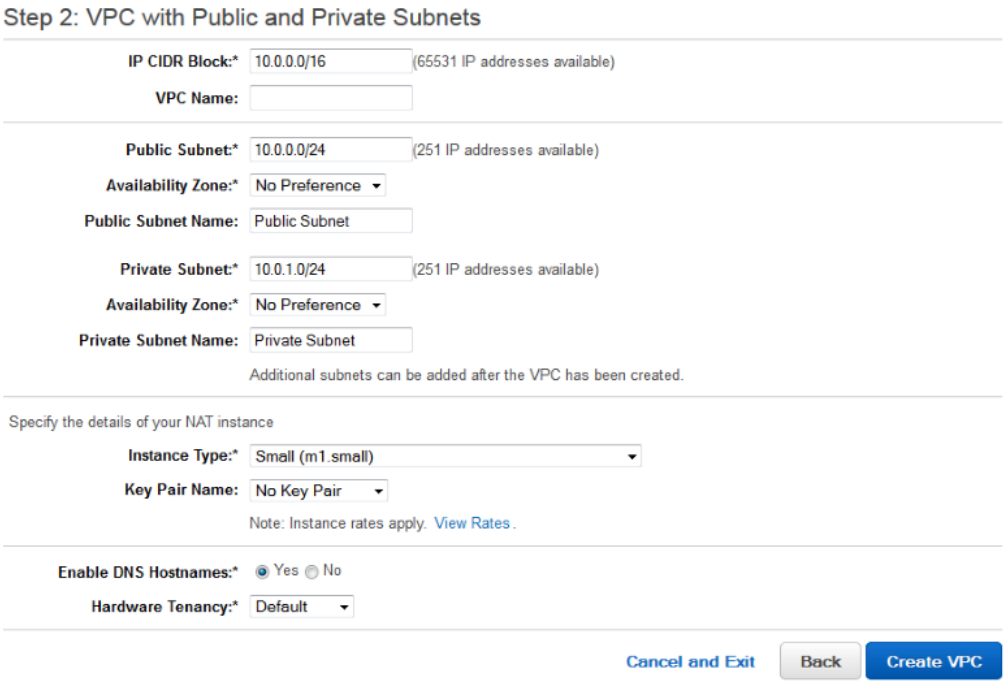
In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation.

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via a Network Address Translation (NAT) instance in the public subnet. (Hourly charges for NAT instances apply)

Internet, S3, DynamoDB, SNS, SQS, etc.
Amazon Virtual Private Cloud
Public Subnet
Private Subnet
NAT
Select
Cancel and Exit

4. Your Elastic Load Balancing load balancer and your Amazon EC2 instances must be in the same Availability Zone so they can communicate with each other. Choose the same Availability Zone from each **Availability Zone** list.



Step 2: VPC with Public and Private Subnets

IP CIDR Block:*	10.0.0.0/16 (65531 IP addresses available)
VPC Name:	
Public Subnet:*	10.0.0.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Public Subnet Name:	Public Subnet
Private Subnet:*	10.0.1.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Private Subnet Name:	Private Subnet

Additional subnets can be added after the VPC has been created.

Specify the details of your NAT instance

Instance Type:*	Small (m1.small)
Key Pair Name:	No Key Pair

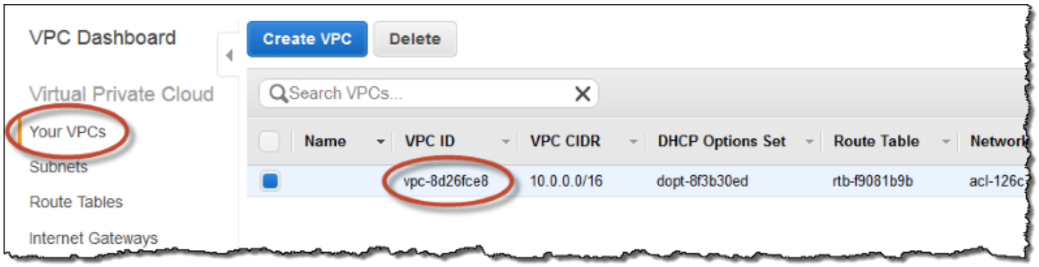
Note: Instance rates apply. View Rates.

Enable DNS Hostnames:*	● Yes ○ No
Hardware Tenancy:*	Default

5. Click **Create VPC**.

   The wizard begins to create your VPC, subnets, and Internet gateway. It also updates the main route table and creates a custom route table. Finally, the wizard launches a NAT instance in the public subnet and prepares it for use. This preparation includes disabling the source/destination check on the instance and assigning the instance an Elastic IP address.

After the VPC is successfully created, you will get a VPC ID. You will need this for this for the next step. To view your VPC ID, click **Your VPCs** in the left pane of the Amazon VPC console.



Name	VPC ID	VPC CIDR	DHCP Options Set	Route Table	Networ
	vpc-8d26fce8	10.0.0.0/16	dopt-8f3b30ed	rtb-f9081b9b	acl-126c

Step 2: Configure the Security Groups

You'll need to create two new security groups: one for your bastion host so you can connect to the Amazon EC2 instances, and the other so that your instances can connect to the NAT instance. Then you'll need to update your security group for the NAT instance so that your Amazon EC2 instances can connect to the NAT instance.

First, create two new security groups.

To create a security group

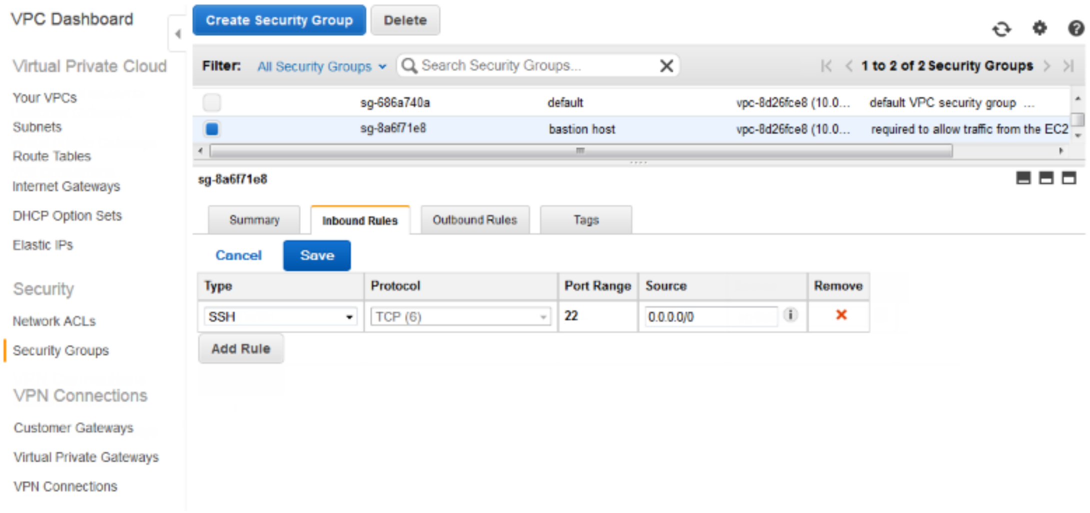
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, click **Security Groups**.
3. Click **Create Security Group**.
4. Enter the security group name, a description of the group, and select the ID for your VPC. You may also enter a tag for the group (optional). Then click **Yes, Create**.

   The security group is created and appears on the **Security Groups** page. Notice that it has an ID (e.g., sg-xxxxxxxx). You might have to turn on the **Group ID** column by clicking **Show/Hide** in the top right corner of the page.

Next, update your security group for the bastion host to enable SSH access.

To update the security group for the bastion host

1. In the list of security groups, select the check box for the security group you just created for your bastion host.
2. On the **Inbound Rules** tab, click **Edit**.
3. Under **Type**, select **SSH**.
4. Click **Source**. For this exercise, type `0.0.0.0/0`, and then click **Save**. This allows access to the host from everyone.

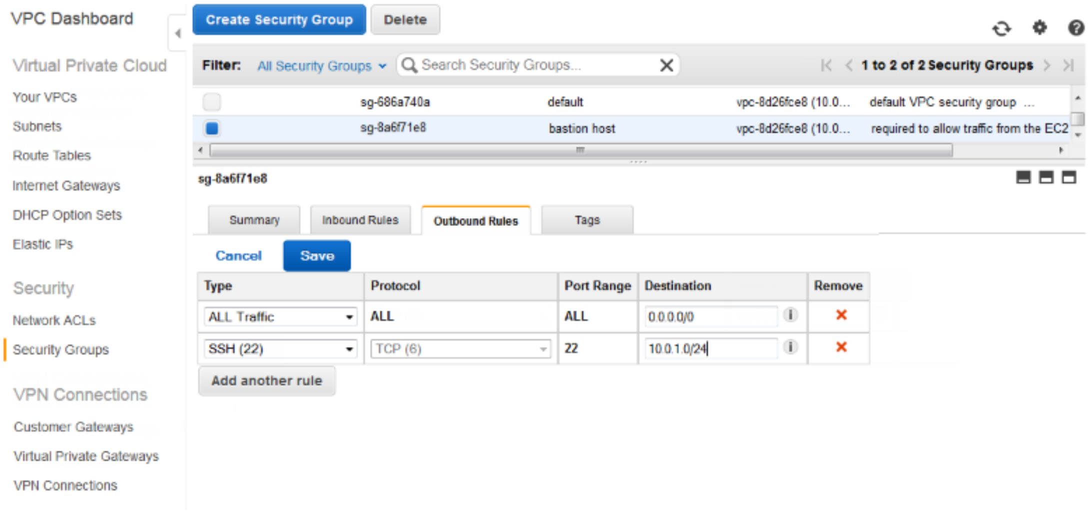


□	sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group ...
■	sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to allow traffic from the EC2

sg-8a6f71e8

Type	Protocol	Port Range	Source	Remove
SSH	TCP (6)	22	0.0.0.0/0	✕

5. On the **Outbound Rules** tab, click **Edit**.
6. In the row below the existing rule for **ALL Traffic**, which comes with your VPC by default, for **Type**, select **SSH**,
7. Click **Destination**, type `10.0.1.0/24` (which is your private subnet), and then click **Save**. This allows access to the Amazon EC2 instances from your bastion host.



□	sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group ...
■	sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to allow traffic from the EC2

sg-8a6f71e8

Type	Protocol	Port Range	Destination	Remove
ALL Traffic	ALL	ALL	0.0.0.0/0	✕
SSH (22)	TCP (6)	22	10.0.1.0/24	✕

Next, update your security group for your NAT instance.

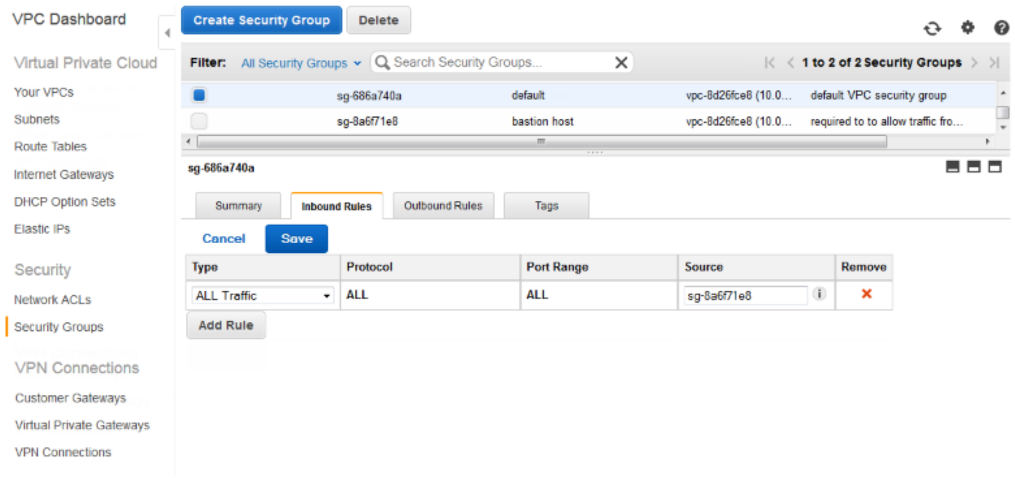
To update the default VPC security group for the NAT instance

1. In the list of security groups, select the check box for the default VPC security group for the NAT instance.
2. Add a rule to allow all traffic access to the group from the security group you just created.

    a. On the **Inbound Rules** tab, click **Edit**.

    b. Under **Type**, select **All Traffic**.

    c. Under **Source**, type the ID (e.g., sg-xxxxxxxx) of the security group, and then click **Save**.



☑	sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group
□	sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to to allow traffic fro...

sg-686a740a

Type	Protocol	Port Range	Source	Remove
ALL Traffic	ALL	ALL	sg-8a6f71e8	✕

Step 3: Create a Bastion Host

Next, you'll need to launch an EC2 instance in your public subnet, and assign an Elastic IP address so that you can connect to it. We'll use the Amazon EC2 console for this step.

Launch an EC2 instance inside a VPC

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the Amazon EC2 console dashboard, click **Launch Instance**.
3. On the **Choose an Amazon Machine Image (AMI)** page, the **Quick Start** tab displays a list of basic configurations called Amazon Machine Images (AMI). Choose the AMI that you want to use and click its **Select** button. In this example, we'll use an Amazon Linux AMI.
4. For this exercise, on the **Choose an Instance Type** page, accept the default value, **t1.micro**, to launch a single micro instance into your subnet. Click **Next: Configure Instance Details**.
5. On the **Configure Instance Details** page, next to **Network**, select your VPC ID.
6. Confirm that **Subnet** displays your public subnet ID, and then click **Next: Add Storage**.
7. On the **Add Storage** page, click **Next: Tag Instance** to accept the defaults for the purposes of this exercise.
8. On the **Tag Instance** page, click **Next: Configure Security Group**. (You do not need tags for the purposes of this exercise.)
9. On the **Configure Security Group** page, next to **Assign a security group**, click **Select an existing security group**.
10. Select the security group you created for your bastion host, and then click **Review and Launch**.
11. On the **Review Instance Launch** page, review your settings. When you're satisfied with your selections, click **Launch**.
12. In the **Select an existing key pair or create a new key pair** dialog box, you can select an existing key pair or create a new one. For this exercise, we'll create a key pair.

    a. Click **Create a new key pair**.

    b. Enter a name for your key pair (for example, **`VPC_Keypair`**), and then click **Download Key Pair**. You need the contents of the private key to connect to your instance after the instance launches. Amazon Web Services doesn't keep the private portion of key pairs.

> **Note**
> EC2 uses this name to also name the private key file (with a `.pem` extension) associated with the pair.

When prompted, save the private key in a safe place on your system.

Next, create an Elastic IP address for your EC2 instance. When you assign an Elastic IP address to your running instance, it provides your instance, which is private by default, with a public IP address so that it can be reached from the Internet.

To assign a VPC Elastic IP address to an instance

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, click **Elastic IPs**.
3. Click **Allocate New Address**.
4. In the **Allocate New Address** dialog box, for **EIP used in:**, select **VPC**, and then click **Yes, Allocate**.
5. Select the new IP address from the list and click **Associate Address**.
6. In the **Associate Address** dialog box, select the instance to associate the address with, and then click **Yes, Associate**.

Your instance now has an Elastic IP address associated with it that makes it accessible from the Internet. You can also access it using SSH from your home network, specifying the Elastic IP address of the instance as the address to connect to. Follow the next steps to define your option settings and deploy to Elastic Beanstalk. After you deploy to Elastic Beanstalk you can connect to your EC2 instances from your bastion host.

Step 4: Define the Option Settings

When you update the option settings, you will specify the following:

- **VPCId**–Contains the ID of the VPC.
- **Subnets**–Contains the ID of the Auto Scaling group subnet. In this example, this is the ID of the private subnet.
- **ELBSubnets**–Contains the ID of the subnet for the Elastic Load Balancing load balancer. In this example, this is the ID of the public subnet.
- **SecurityGroups**–Contains the ID of the security groups. In this example, you'll use the IDs of the new security groups you created in Step 2: Configure the Security Groups (p. 311).

SSHSourceRestriction is an optional setting if you want to lock down SSH access to an environment. In this example, we use it to lockdown SSH access to the EC2 instances so that only the bastion host can access the instances in the private subnet. For more information, see General Options for all Environments (p. 103).

The following is an example of the option settings you could use when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration Options (p. 103).

```
option_settings:
  - namespace: aws:autoscaling:launchconfiguration
  option_name: EC2KeyName
  value: bastionhostkeypair

- namespace: aws:ec2:vpc
  option_name: VPCId
  value: vpc-170647c

- namespace: aws:ec2:vpc
  option_name: Subnets
  value: subnet-4f195024

- namespace: aws:ec2:vpc
  option_name: ELBSubnets
  value: subnet-fe064f95

- namespace: aws:autoscaling:launchconfiguration
  option_name: InstanceType
  value: m1.small

- namespace: aws:autoscaling:launchconfiguration
  option_name: SecurityGroups
  value: sg-7f1ef110

- namespace: aws:autoscaling:launchconfiguration
  option_name: SecurityGroups
  value: sg-9h0lc223
```

Step 5: Deploy to Elastic Beanstalk

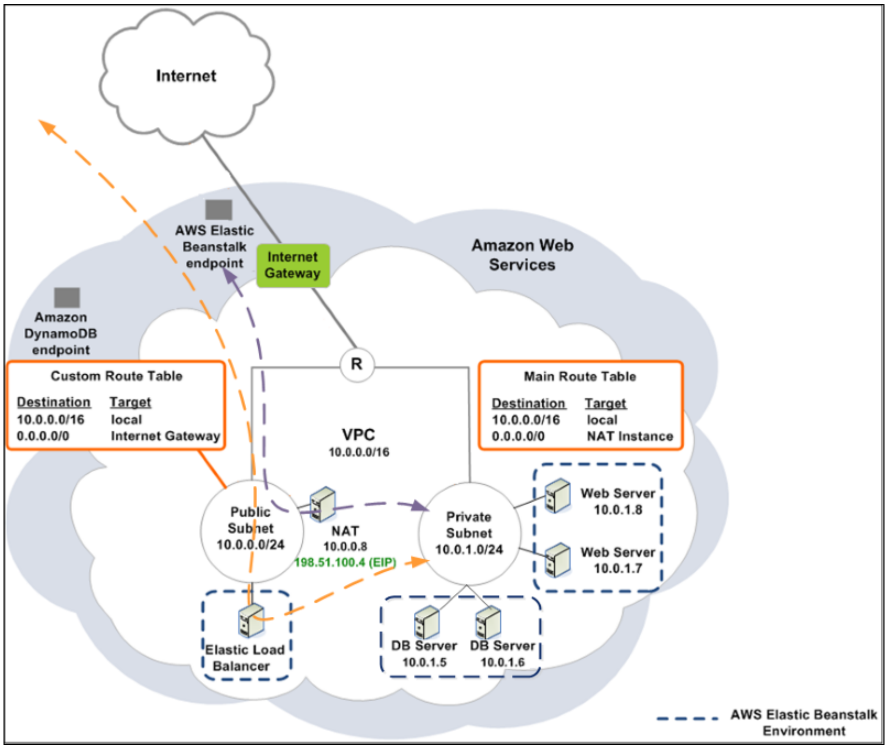
Next, you will deploy your Elastic Beanstalk using the option settings you created in the previous step. To deploy your Elastic Beanstalk application with your options settings, you need to create an `.ebextensions` directory at the top-level directory of your source bundle, place your configuration file (e.g., `myapp.config`) inside the `.ebextensions` directory, and deploy your source bundle to Elastic Beanstalk using the Elastic Beanstalk console. Alternatively, you can also use eb, the CLI, or the API.

To learn how to deploy your source bundle to Elastic Beanstalk using the Elastic Beanstalk console, see Create an Application (p. 34).

Now that you have deployed to Elastic Beanstalk you can connect to your EC2 instances from your bastion host.

Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS

This topic walks you through deploying an Elastic Beanstalk application with Amazon RDS in a VPC using a NAT instance. Your infrastructure will look similar to the following diagram:



To deploy an Elastic Beanstalk application with Amazon RDS inside a VPC using a NAT instance, you need to do the following:

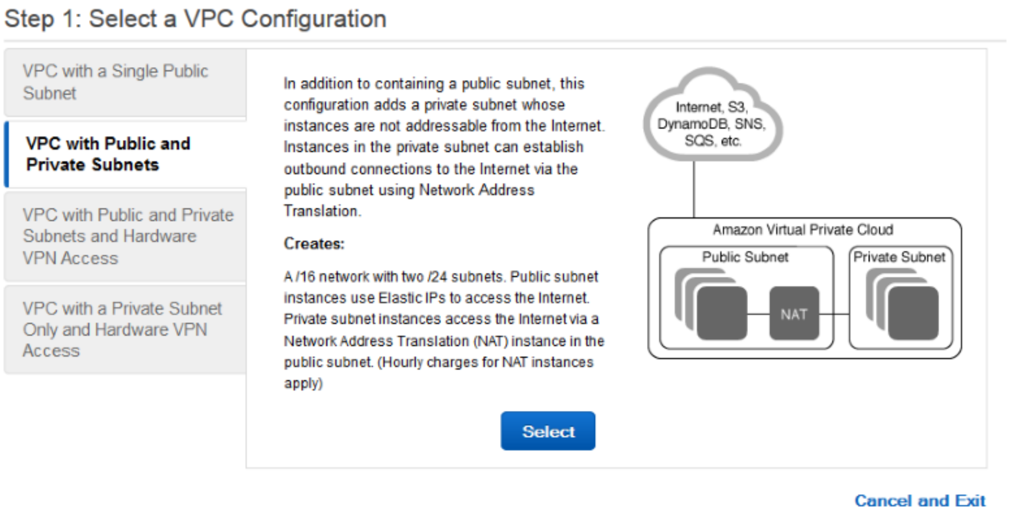
1. Step 1: Create a VPC with a Public and Private Subnet (p. 316)
2. Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 318)
3. Step 3: Create a DB Subnet Group (p. 319)
4. Step 4: Deploy to Elastic Beanstalk (p. 320)

Step 1: Create a VPC with a Public and Private Subnet

You can use the Amazon VPC console to create a VPC.

To create a VPC

1. Sign in to the Amazon VPC console.
2. In the navigation pane, click **VPC Dashboard**. Then click **Start VPC Wizard**.
3. Click **VPC with Public and Private Subnets** and then click **Select**.



Step 1: Select a VPC Configuration

- VPC with a Single Public Subnet
- **VPC with Public and Private Subnets**
- VPC with Public and Private Subnets and Hardware VPN Access
- VPC with a Private Subnet Only and Hardware VPN Access

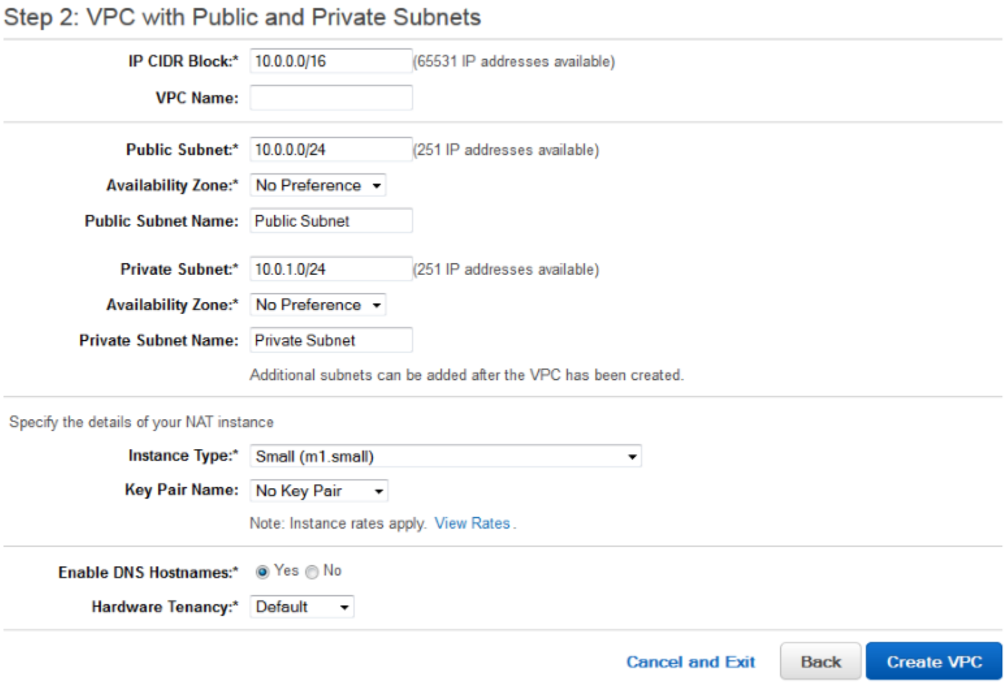
In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation.

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via a Network Address Translation (NAT) instance in the public subnet. (Hourly charges for NAT instances apply)

Internet, S3, DynamoDB, SNS, SQS, etc.
Amazon Virtual Private Cloud
Public Subnet
Private Subnet
NAT
Select
Cancel and Exit

4. Your Elastic Load Balancing load balancer and your Amazon EC2 instances must be in the same Availability Zone so they can communicate with each other. Choose the same Availability Zone from each **Availability Zone** list.



Step 2: VPC with Public and Private Subnets

IP CIDR Block:*	10.0.0.0/16 (65531 IP addresses available)
VPC Name:	
Public Subnet:*	10.0.0.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Public Subnet Name:	Public Subnet
Private Subnet:*	10.0.1.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Private Subnet Name:	Private Subnet

Additional subnets can be added after the VPC has been created.

Specify the details of your NAT instance

Instance Type:*	Small (m1.small)
Key Pair Name:	No Key Pair

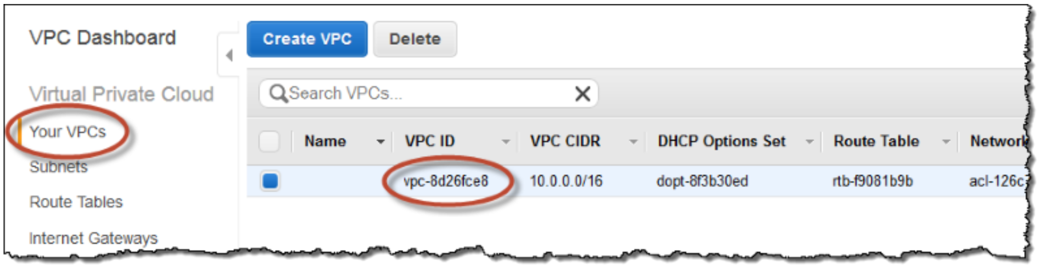
Note: Instance rates apply. View Rates.

Enable DNS Hostnames:*	● Yes ○ No
Hardware Tenancy:*	Default

5. Click **Create VPC**.

   The wizard begins to create your VPC, subnets, and Internet gateway. It also updates the main route table and creates a custom route table. Finally, the wizard launches a NAT instance in the public subnet and prepares it for use. This preparation includes disabling the source/destination check on the instance and assigning the instance an Elastic IP address.

After the VPC is successfully created, you will get a VPC ID. You will need this for this for the next step. To view your VPC ID, click **Your VPCs** in the left pane of the Amazon VPC console.



Name	VPC ID	VPC CIDR	DHCP Options Set	Route Table	Network
	vpc-8d26fce8	10.0.0.0/16	dopt-8f3b30ed	rtb-f9081b9b	acl-126c

Step 2: Configure the Default VPC Security Group for the NAT Instance

You will need to update the default VPC security group to allow traffic from the EC2 instances that Elastic Beanstalk creates to the NAT instance. To do this, you will first create a new security group, and then update your default security group to allow traffic from this new security group.

To create a security group

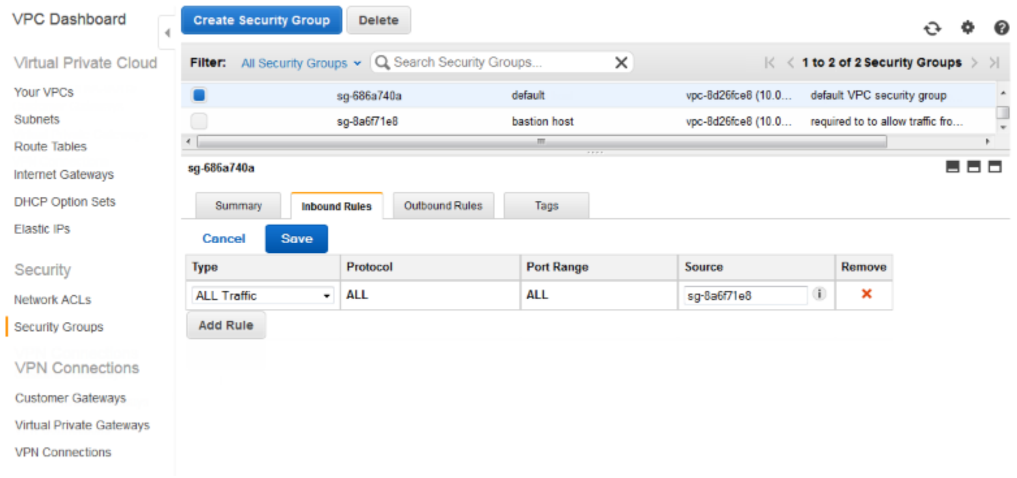
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, click **Security Groups**.
3. Click **Create Security Group**.
4. Enter the security group name, a description of the group, and select the ID for your VPC. You may also enter a tag for the group (optional). Then click **Yes, Create**.

   The security group is created and appears on the **Security Groups** page. Notice that it has an ID (e.g., sg-xxxxxxxx). You might have to turn on the **Group ID** column by clicking **Show/Hide** in the top right corner of the page.

To update the default VPC security group for the NAT instance

1. In the list of security groups, select the check box for the default VPC security group for the NAT instance.
2. Add a rule to allow all traffic access to the group from the security group you just created.

   a. On the **Inbound Rules** tab, click **Edit**.
   b. Under **Type**, select **All Traffic**.
   c. Under **Source**, type the ID (e.g., sg-xxxxxxxx) of the security group, and then click **Save**.



■	sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group
□	sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to to allow traffic fro...

sg-686a740a

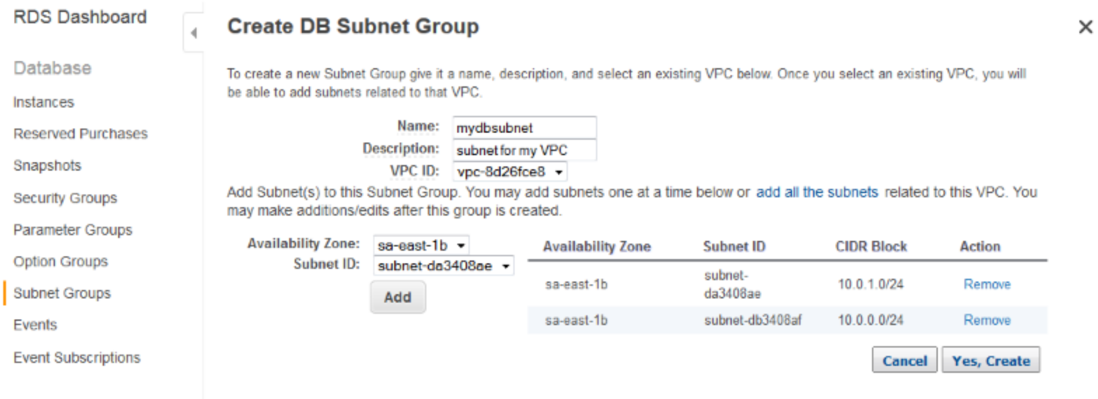
Type	Protocol	Port Range	Source	Remove
ALL Traffic	ALL	ALL	sg-8a6f71e8	✕

Step 3: Create a DB Subnet Group

A DB Subnet Group for a VPC is a collection of subnets (typically private) that you may want to designate for your back-end RDS DB Instances. Each DB Subnet Group should have at least one subnet for every Availability Zone in a given region.

Create a DB subnet group

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, click **Subnet Groups**.
3. Click **Create DB Subnet Group**.
4. Click **Name**, and then type the name of your DB Subnet Group.
5. Click **Description**, and then describe your DB Subnet Group.
6. Next to **VPC ID**, select the ID of the VPC that you created.
7. Click the **add all the subnets** link in the **Add Subnet(s) to this Subnet Group** section.



Create DB Subnet Group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Name: mydbsubnet
Description: subnet for my VPC
VPC ID: vpc-8d26fce8

Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created.

Availability Zone: sa-east-1b
Subnet ID: subnet-da3408ae

Availability Zone	Subnet ID	CIDR Block	Action
sa-east-1b	subnet-da3408ae	10.0.1.0/24	Remove
sa-east-1b	subnet-db3408af	10.0.0.0/24	Remove

8. When you are finished, click **Yes, Create**.
9. In the confirmation window, click **Close**.

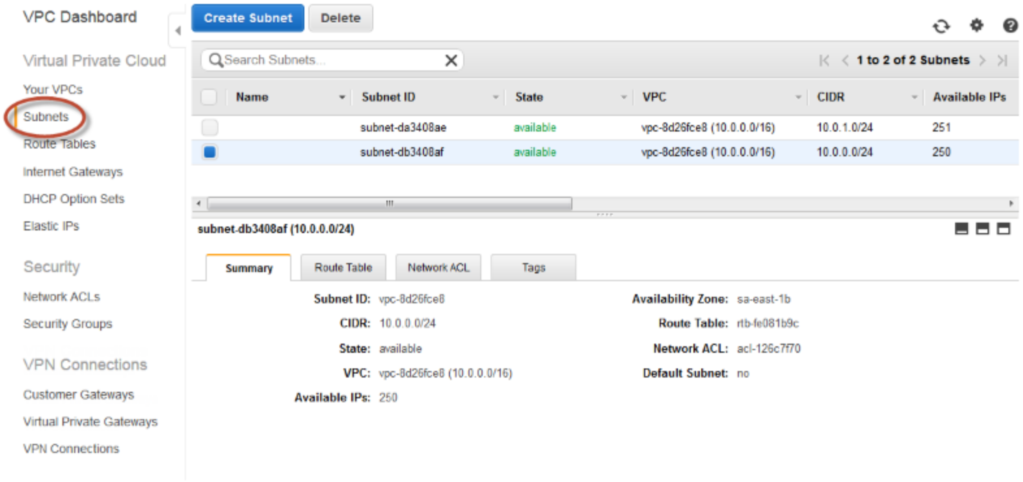
Your new DB Subnet Group appears in the DB Subnet Groups list of the RDS console. You can click it to see details, such as all of the subnets associated with this group, in the details pane at the bottom of the window.

Step 4: Deploy to Elastic Beanstalk

After you set up your VPC, you can deploy your application to Elastic Beanstalk and create your environment inside your VPC. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, eb, API-based command line interface (CLI), or API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS Toolkit, eb, CLI, or API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk Console

The Elastic Beanstalk console walks you through creating your new environment inside your VPC. You need to provide a `.war` file (for Java applications) or a `.zip` file for all other applications. The Elastic Beanstalk console asks you for the IDs for your VPC and VPC security group you created in the previous steps as well as the subnet IDs for your load balancer and EC2 instances. If you created a private subnet for your EC2 instances and a public subnet for your load balancer, make sure you select the public subnet ID for the load balancer, and the private subnet ID for your EC2 instances. By default, VPC creates a public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your subnet IDs by clicking **Subnets** in the Amazon VPC console.



	Name	Subnet ID	State	VPC	CIDR	Available IPs
□		subnet-da3408ae	available	vpc-8d26fce8 (10.0.0.0/16)	10.0.1.0/24	251
☑		subnet-db3408af	available	vpc-8d26fce8 (10.0.0.0/16)	10.0.0.0/24	250

subnet-db3408af (10.0.0.0/24)

Subnet ID: vpc-8d26fce8
CIDR: 10.0.0.0/24
State: available
VPC: vpc-8d26fce8 (10.0.0.0/16)
Available IPs: 250

Availability Zone: sa-east-1b
Route Table: rtb-fe081b9c
Network ACL: acl-126c7f70
Default Subnet: no

Deploying with the AWS Toolkits, Eb, CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, eb, CLI, or API, you need to specify your VPC option settings in a file and deploy it with your source bundle. To deploy your application using a toolkit or eb, create a configuration file with a `.config` extension (e.g., `myconfig.config`) and place it inside a directory named `.ebextensions` in the top-level directory of your source bundle. If you use the CLI or API, you can specify these settings in a file name and pass the file name in as a parameter. Deploy your application to Elastic Beanstalk using one of the following methods:

- Java — Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576)

- The EB CLI — Tools (p. 384)
- CLI or API — Create an Application (p. 34) (see the CLI or API section)
- CLI or API — Launching a New AWS Elastic Beanstalk Environment (p. 55) (see the CLI or API section)

When you update the option settings, you will need to specify at least the following:

- **VPCId**–Contains the ID of the VPC.
- **Subnets**–Contains the ID of the Auto Scaling group subnet. In this example, this is the ID of the private subnet.
- **ELBSubnets**–Contains the ID of the subnet for the elastic load balancer. In this example, this is the ID of the public subnet.
- **SecurityGroups**–Contains the ID of the security groups. In this example, you'll use the ID of the new security group you created in Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 318).
- **DBSubnets**–Contains the ID of the DB subnets.

  Note
  When using DBSubnets, you need to create additional subnets in your VPC to cover all the Availability Zones in the region.

Optionally, you can also specify the following information:

- **ELBScheme** — Specify `internal` if you want to create an internal load balancer inside your VPC so that your Elastic Beanstalk application cannot be accessed from outside your VPC.

The following is an example of the option settings you could use when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration Options (p. 103).

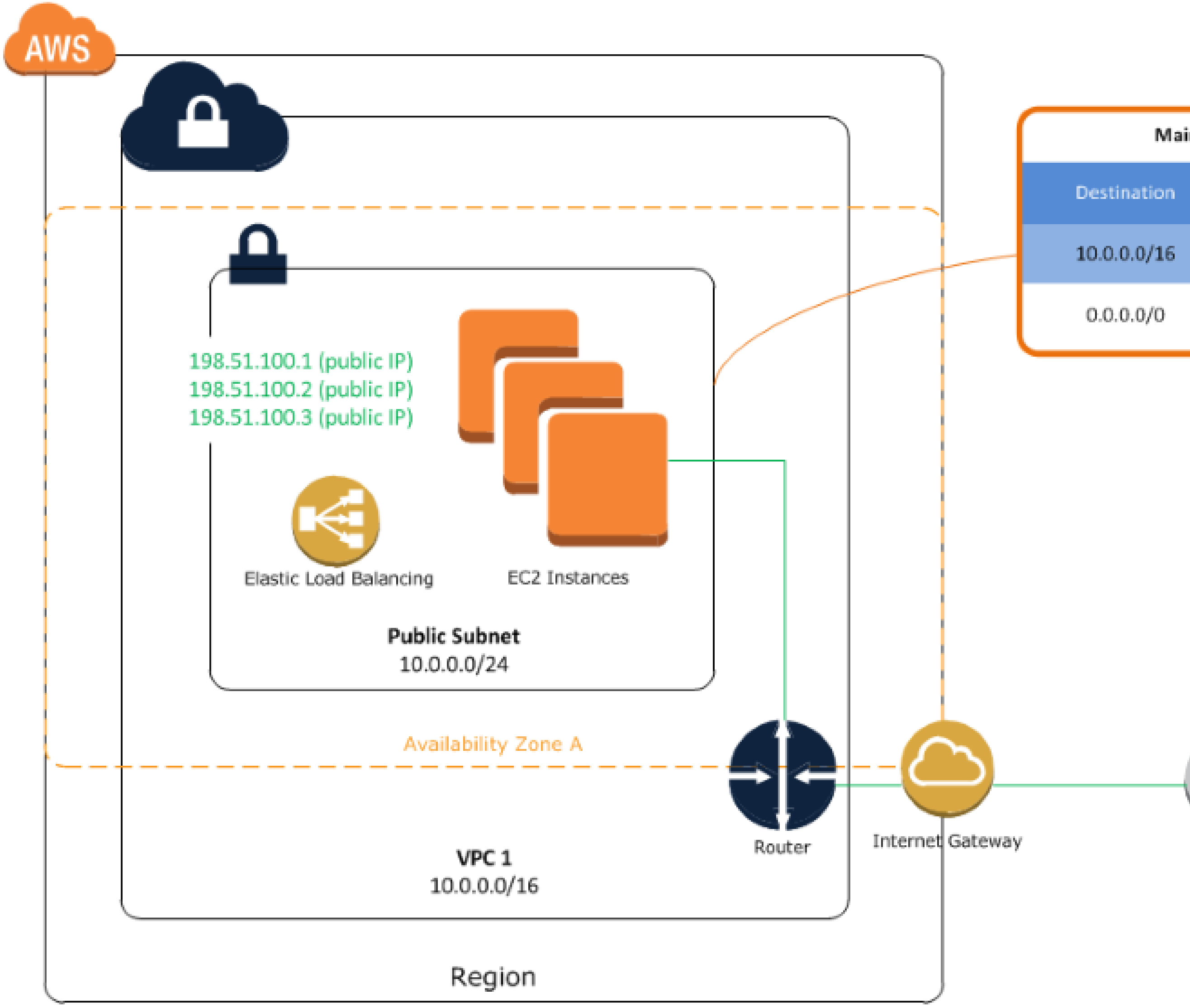
```
option_settings:
  - namespace: aws:autoscaling:launchconfiguration
    option_name: EC2KeyName
    value: ec2keypair

  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-170647c

  - namespace: aws:ec2:vpc
    option_name: Subnets
    value: subnet-4f195024

  - namespace: aws:ec2:vpc
    option_name: ELBSubnets
    value: subnet-fe064f95

  - namespace: aws:ec2:vpc
    option_name: DBSubnets
    value: subnet-fg148g78

  - namespace: aws:autoscaling:launchconfiguration
    option_name: InstanceType
    value: m1.small

  - namespace: aws:autoscaling:launchconfiguration
    option_name: SecurityGroups
    value: sg-7f1ef110
```

Note

When using DBSubnets, make sure you have subnets in your VPC to cover all the Availability Zones in the region.

Example: Launching a Load-Balancing, Autoscaling Environment with Public Instances in a VPC

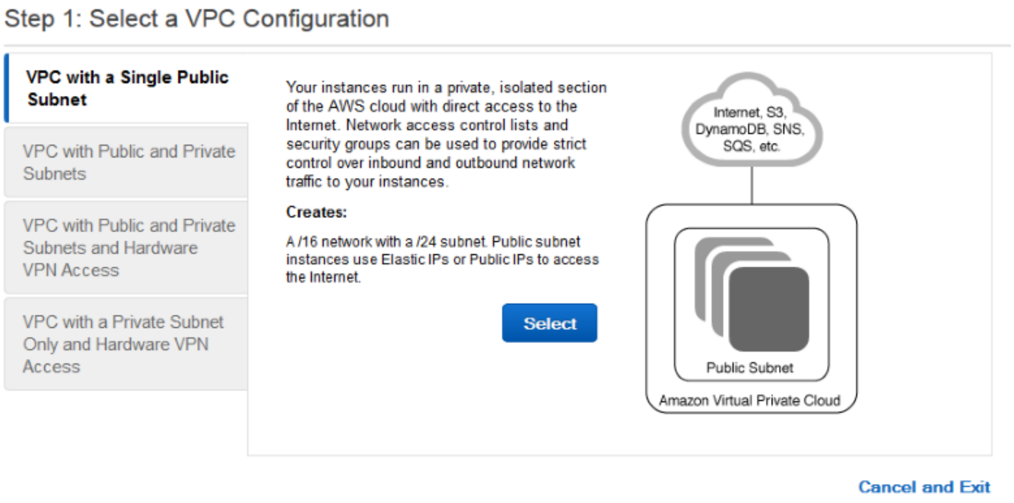
You can deploy an Elastic Beanstalk application in a load balancing, autoscaling environment in a single public subnet. Use this configuration if you have a single public subnet without any private resources associated with your Amazon EC2 instances. In this configuration, Elastic Beanstalk assigns public IP addresses to the Amazon EC2 instances so that each can directly access the Internet through the VPC Internet gateway. You do not need to launch a network address translation (NAT) instance or configure the default VPC security group to allow traffic from the Amazon EC2 instances to the NAT instance.



Step 1: Create a VPC with a Public Subnet

To create a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, click **VPC Dashboard**, locate the **Your Virtual Private Clouds** area, and then click **Start VPC Wizard**.
3. Select **VPC with a Single Public Subnet** and then click **Select**.



Step 1: Select a VPC Configuration

VPC with a Single Public Subnet
VPC with Public and Private Subnets
VPC with Public and Private Subnets and Hardware VPN Access
VPC with a Private Subnet Only and Hardware VPN Access

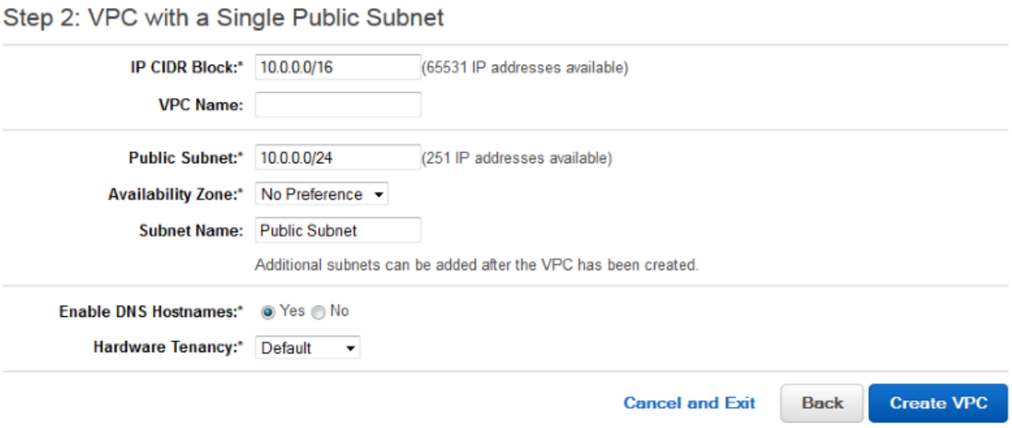
Your instances run in a private, isolated section of the AWS cloud with direct access to the Internet. Network access control lists and security groups can be used to provide strict control over inbound and outbound network traffic to your instances.

Creates:

A /16 network with a /24 subnet. Public subnet instances use Elastic IPs or Public IPs to access the Internet.

Select
Internet, S3, DynamoDB, SNS, SQS, etc.
Public Subnet
Amazon Virtual Private Cloud
Cancel and Exit

A confirmation page shows the CIDR blocks used for the VPC and subnet. The page also shows the subnet and the associated Availability Zone.



Step 2: VPC with a Single Public Subnet

IP CIDR Block:* 10.0.0.0/16 (65531 IP addresses available)

VPC Name:

Public Subnet:* 10.0.0.0/24 (251 IP addresses available)

Availability Zone:* No Preference

Subnet Name: Public Subnet

Additional subnets can be added after the VPC has been created.

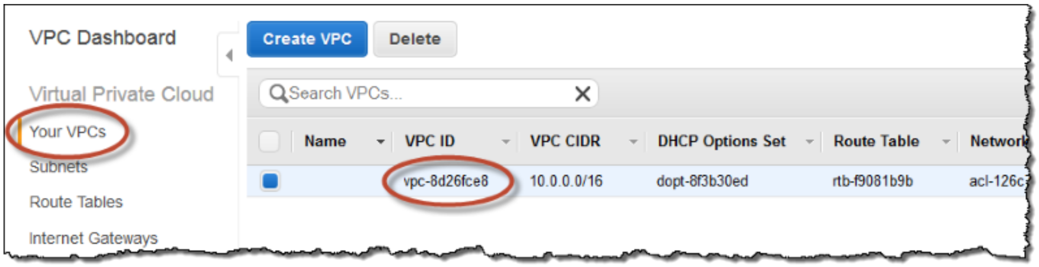
Enable DNS Hostnames:* ● Yes ○ No

Hardware Tenancy:* Default

4. Click **Create VPC**.

   AWS creates your VPC, subnet, Internet gateway, and route table. Click **Close** to end the wizard.

   After AWS successfully creates the VPC, it assigns the VPC a VPC ID. You will need this for this for the next step. To view your VPC ID, click **Your VPCs** in the left pane of the Amazon VPC console.



	Name	VPC ID	VPC CIDR	DHCP Options Set	Route Table	Network
■		vpc-8d26fce8	10.0.0.0/16	dopt-8f3b30ed	rtb-f9081b9b	acl-126c

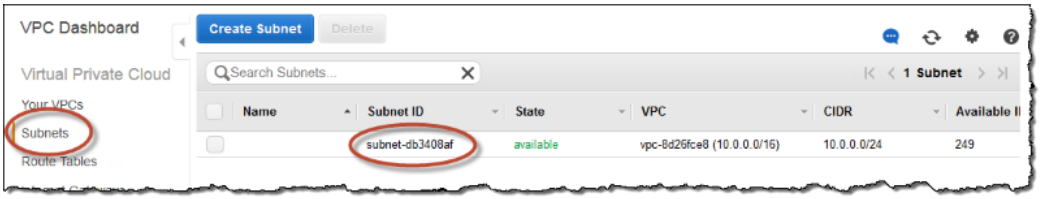
Step 3: Deploy to Elastic Beanstalk

After you set up your VPC, you can deploy your application to Elastic Beanstalk and create your environment inside your VPC. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, eb, API-based command line interface (CLI), or API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS Toolkit, eb, CLI, or API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk Console

When you create an Elastic Beanstalk application or launch an environment, the Elastic Beanstalk console walks you through creating your environment inside a VPC. For more information, see Create an Application (p. 34).

You'll need to select the VPC ID and subnet ID for your instance. By default, VPC creates a public subnet using 10.0.0.0/24. You can view your subnet ID by clicking **Subnets** in the Amazon VPC console.



	Name	Subnet ID	State	VPC	CIDR	Available I
☐		subnet-db3408af	available	vpc-8d26fce8 (10.0.0.0/16)	10.0.0.0/24	249

Deploying with the AWS Toolkits, Eb, CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, eb, CLI, or API, you need to specify your VPC option settings in a file and deploy it with your source bundle. To deploy your application using a toolkit or eb, create a configuration file with a `.config` extension (e.g., `myconfig.config`) and place it inside a directory named `.ebextensions` in the top-level directory of your source bundle. If you use the CLI or API, you can specify these settings in a file name and pass the file name in as a parameter. Deploy your application to Elastic Beanstalk using one of the following methods:

- Java — Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse (p. 576)
- The EB CLI — Tools (p. 384)
- CLI or API — Create an Application (p. 34) (see the CLI or API section)
- CLI or API — Launching a New AWS Elastic Beanstalk Environment (p. 55) (see the CLI or API section)

When you create your configuration file, you will need to specify at least the following:

- **VPCId** — Contains the ID of the VPC.
- **Subnets** — Contains the ID of the public subnet that contains the load balancer and the instances.
- **AssociatePublicIpAddress** — Specifies whether to launch instances with public IP addresses in your VPC. Instances with public IP addresses do not require a NAT instance to communicate with the Internet. You must set the value to `true` if you want to include your load balancer and instances in a single public subnet.

The following is an example of the settings you could set when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration Options (p. 103).

```
option_settings:
  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-8d26fce8

  - namespace: aws:ec2:vpc
    option_name: Subnets
    value: subnet-db3408af

  - namespace: aws:ec2:vpc
    option_name: AssociatePublicIpAddress
    value: true
```

Using Elastic Beanstalk with AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) helps you securely control access to your AWS resources. This section includes reference materials for working with IAM policies, instance profiles, and service roles.

For an overview of these subjects, see Service Roles, Instance Profiles, and User Policies (p. 19). For most environments, the service role and instance profile that the AWS Management Console prompts you to create with 1-click role creation when you launch your environment have all of the permissions that you need. Likewise, the managed policies (p. 326) provided by Elastic Beanstalk for full access and read-only access contain all of the user permissions required for daily use.

To learn how to apply a Elastic Beanstalk managed policy to a user or group, see Using Managed Policies to Control Access to All Elastic Beanstalk Resources (p. 326). To learn more about custom policies, and how to allow or deny permissions to perform specific actions on Elastic Beanstalk resources, see Creating Policies to Control Access to Specific Elastic Beanstalk Resources (p. 329). To learn how to use IAM roles with Elastic Beanstalk, see Using IAM Roles with Elastic Beanstalk (p. 332). For more information about permissions, see Permissions and Policies in the *IAM User Guide*. For more information about roles and how they differ from users or groups, see Roles in the *IAM User Guide*. To learn how to use policies to control access to resources, see Service Roles, Instance Profiles, and User Policies (p. 19).For instructions using the Elastic Beanstalk console, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34). For instructions using the EB CLI, see Tools (p. 384)

The IAM User Guide provides in-depth coverage of AWS permissions.

Using Managed Policies to Control Access to All Elastic Beanstalk Resources

AWS Elastic Beanstalk (Elastic Beanstalk) provides two managed policies that enable you to assign full access or read-only access to all Elastic Beanstalk resources. You can attach the policies to users or groups. Use these policies to grant broad permissions for all Elastic Beanstalk resources in your AWS account. To control permissions for specific resources, create a new policy as described in Creating Policies to Control Access to Specific Elastic Beanstalk Resources (p. 329).

The following table describes each managed policy.

Managed Policy	Description
AWSElasticBeanstalkFullAccess	This policy allows the user to create, modify, and delete Elastic Beanstalk applications, application versions, configuration settings, environments, and their underlying resources. For non-legacy container types, this policy also allows Elastic Beanstalk to provision and manage resources in other services on your behalf. For a list of supported non-legacy container types, see Why are some container types marked legacy? (p. 92).
AWSElasticBeanstalkReadOnlyAccess	This policy allows the user to view applications and environments, but not to perform operations on them. It provides read-only access to all Elastic Beanstalk resources.

To apply a managed policy to a user or group

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the check box next to the name of the policy to attach. To filter the list of policies, you can use the **Filter** menu or the **Search** box.
4. Choose **Policy Actions**, then **Attach**.
5. Choose the principal entities to attach the policy to. To filter the list of principal entities, you can use the **Filter** menu or the **Search** box. After choosing the principal entities to attach the policy to, choose **Attach Policy**.

Working with Service Roles

The following service role contains all of the permissions needed by Elastic Beanstalk to monitor environment health:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeInstanceHealth",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:GetConsoleOutput",
        "ec2:AssociateAddress",
        "ec2:DescribeAddresses",
        "ec2:DescribeSecurityGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeNotificationConfigurations"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

To allow Elastic Beanstalk to assume the role, the following trust policy must be attached:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
          "Service": "elasticbeanstalk.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "elasticbeanstalk"
          }
        }
      }
    ]
}
```

To assign a service role to a environment using the EB CLI (p. 384)

1. Use the `eb config` command to open the environment configuration file in the default text editor:

   ```
   ~/project$ eb config
   ```

2. In the **settings** section, locate the `aws:elasticbeanstalk:environment` namespace. Change the value of `ServiceRole` to the name of your role.

   ```
   settings
     aws:elasticbeanstalk:environment:
       EnvironmentType: LoadBalanced
       ServiceRole: aws-elasticbeanstalk-service-role
   ```

3. Save the configuration file, and then close the text editor.
4. The EB CLI starts an environment update to apply your configuration changes. Wait for the operation to complete or press **Ctrl**+**C** to exit safely.

Creating Policies to Control Access to Specific Elastic Beanstalk Resources

You can create your own IAM policy to allow or deny specific Elastic Beanstalk API actions on specific Elastic Beanstalk resources. To put the policy into effect, you attach it to a user or group using the IAM console, command line interface, or API. For more information about attaching a policy to a user or group, see Managing IAM Policies in *Using AWS Identity and Access Management*.

An IAM policy contains policy statements that describe the specific permissions you want to grant. When you create a policy statement for Elastic Beanstalk, there are four parts of a statement that you need to know how to use:

- **Effect** specifies whether to allow or deny the actions in the statement.
- **Action** specifies the actions you want to control. To specify Elastic Beanstalk actions, the action name must be prefixed with the lowercase string `elasticbeanstalk`. You use wildcards to specify all actions related to Elastic Beanstalk. The wildcard `"*"` matches zero or multiple characters. For example, to grant all create action permissions, you can specify `elasticbeanstalk:create*` in your IAM policy.

  Note

  If your policy uses a wildcard to specify all actions instead of explicitly listing each action, be aware that if an update to Elastic Beanstalk were to add any new actions, this policy would automatically give the grantee access to those new actions.

  For a complete list of Elastic Beanstalk actions, see the API action names in the Elastic Beanstalk API Reference. For more information about permissions and policies, go to Permissions and Policies in *Using AWS Identity and Access Management*.

  Users with permission to use specific Elastic Beanstalk API actions can perform those actions. Certain operations, such as creating an environment, may require additional permissions to perform those actions. To check if an API action depends on permissions to other actions and to ensure all required permissions are assigned, use the information in section Resources and Conditions for Elastic Beanstalk Actions (p. 345).
- **Resource** specifies the resources that you want to control access to. To specify Elastic Beanstalk resources, you list the Amazon Resource Name (ARN) of each resource. For more information, see Amazon Resource Name (ARN) Format for Elastic Beanstalk (p. 343). Each Elastic Beanstalk action operates on a specific resource. For example, the `UpdateApplicationVersion` action operates on application versions, which you would specify as one or more version resources. For more information, see Amazon Resource Name (ARN) Format for Elastic Beanstalk (p. 343). To specify multiple ARNs, you can list each resource's ARN or use the `"*"` wildcard, which matches zero or multiple characters.
- **Condition** specifies restrictions on the permission granted in the statement. As discussed earlier, an action operates on a specific resource. However, that action may have dependencies on other Elastic Beanstalk resources such as where the action occurs (for example, creating an environment within an application) or which other resources the action needs access to in order to complete its operation (for example, updating an environment from a configuration template or application version). For more information, see Resources and Conditions for Elastic Beanstalk Actions (p. 345).

IAM policies are expressed in JSON format. For information about the structure of IAM policies and statements, see Basic Policy Structure in *Using AWS Identity and Access Management*. The following example policy contains three statements that enable a user who has this policy to call the `CreateEnvironment` action to create an environment whose name begins with **`Test`** in the application **`My First Elastic Beanstalk Application`** using the application version **`First Release`**. The policy also allows the user to perform actions on the resources required to create the environment. The `CreateEnvironmentPerm` statement allows the `elasticbeanstalk:CreateEnvironment` action to create an environment with the constraints specified above. The `AllNonResourceCalls` statement allows `elasticbeanstalk:CreateEnvironment` to perform the Elastic Beanstalk actions required to create the environment. The `OtherServicePerms` statement allows `elasticbeanstalk:CreateEnvironment` to call the appropriate actions to create resources in other AWS services to complete the creation of the environment.

Note

The following policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":"CreateEnvironmentPerm",
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-1:123456789012:environment/My First
Elastic Beanstalk Application/Test*"
      ],
      "Condition": {
        "StringEquals": {
         "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-
1:123456789012:application/My First Elastic Beanstalk Application"],
          "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbean
stalk:us-east-1:123456789012:applicationversion/My First Elastic Beanstalk Ap
plication/First Release"]
        }
       }
      },
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      },
      {
      "Sid":"OtherServicePerms",
         "Effect":"Allow",
         "Action":[
            "ec2:*",
            "elasticloadbalancing:*",
            "autoscaling:*",
            "cloudwatch:*",
            "s3:*",
            "sns:*",
            "rds:*",
            "cloudformation:*"
         ],
         "Resource":"*"
      }
   ]
}
```

Note that the policy above enables the user to create an environment using the Elastic Beanstalk CreateEnvironment API and the elastic-beanstalk-create-environment (p. 483) command. However, if you want that user to be able to use the Elastic Beanstalk console to create an environment, you must also add the following policy to the user. When the user creates an environment in the Elastic Beanstalk console, the user must be able to navigate to the application **`My First Elastic Beanstalk Application`** (`elasticbeanstalk:DescribeApplications`). When the user clicks **Launch New Environment**, the Elastic Beanstalk console needs to get information about existing environments (`elasticbeanstalk:DescribeEnvironments`), the application version to use (`elasticbeanstalk:DescribeApplicationVersions`), and solution stacks (`elasticbeanstalk:ListAvailableSolutionStacks` and `elasticbeanstalk:DescribeConfigurationOptions`). If you want to enable specific actions within the Elastic Beanstalk console, you need to consider the types of dependencies described in this example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DescribeApplications"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-1:155363561088:application/My First
Elastic Beanstalk Application"
      ]
    },
    {
      "Action": "elasticbeanstalk:DescribeEnvironments",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-1:123456789012:environment/My First
Elastic Beanstalk Application/Test*"
      ]
    },
    {
      "Action": "elasticbeanstalk:DescribeApplicationVersions",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-1:123456789012:applicationversion/My
 First Elastic Beanstalk Application/First Release"
      ]
    },
    {
      "Action": [
        "elasticbeanstalk:ListAvailableSolutionStacks"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:elasticbeanstalk:us-east-1::solutionstack/*"
    },
    {
      "Action": "elasticbeanstalk:DescribeConfigurationOptions",
      "Effect": "Allow",
      "Resource": [
         "arn:aws:elasticbeanstalk:us-east-1::solutionstack/*"
      ]
    }
  ]
}
```

Using IAM Roles with Elastic Beanstalk

IAM roles control what actions and AWS services your Elastic Beanstalk application can access. With roles, you don't have to share long-term credentials or define permissions for each entity that requires access to a resource. To allow your application access to AWS resources, you attach a custom action policy to the IAM role and use the instance profile associated with that role to launch your Amazon EC2 instances. Examples of when Elastic Beanstalk uses IAM roles include when your application requires access to AWS resources such as DynamoDB or if you want Elastic Beanstalk to rotate your logs to Amazon S3.

This document describes how to configure your Elastic Beanstalk application to access AWS services using IAM roles. For more information about using IAM roles with temporary security credentials to access the Elastic Beanstalk API, see Creating Temporary Security Credentials for Delegating API Access in the *AWS Security Token Service User Guide*.

For an overview of the steps required to grant permissions to applications running in Elastic Beanstalk using IAM roles, see Service Roles, Instance Profiles, and User Policies (p. 19).

The following sections provides examples for using IAM roles with Elastic Beanstalk, including sample action policies.

- Example: Granting Permissions to Elastic Beanstalk Applications to Access DynamoDB (p. 335)
- Example: Granting Elastic Beanstalk Permission to Rotate Logs to Amazon S3 (p. 340)

Granting IAM Users Permissions to Create and Pass IAM Roles

You need to have the appropriate permissions so that Elastic Beanstalk can create a default role and instance profile for you, or to view the list of instance profiles available in your environment. If you tried to create or update your environment to use an instance profile, but you received an error, the error might have occurred because you do not have the correct permissions. Your account administrator should allow the following actions:

```
"iam:AddRoleToInstanceProfile",
"iam:CreateInstanceProfile",
"iam:CreateRole",
"iam:PassRole",
"iam:ListInstanceProfiles"
```

You require the create role, create instance profile, and add to instance profile actions in order to create a role. The list instance profiles actions allows you to list the instance profiles in the AWS account, and the pass role action allows you to associate a role to an environment.

The following example shows one statement that gives a broad set of permissions to AWS products that Elastic Beanstalk uses to manage applications and environments and includes permissions to create an instance profile and view a list of available instance profiles.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "iam:AddRoleToInstanceProfile",
        "iam:CreateInstanceProfile",
        "iam:CreateRole",
        "iam:PassRole",
        "iam:ListInstanceProfiles"
      ],
      "Resource": "*"
    }
  ]
}
```

Granting IAM Role Permissions for Worker Environment Tiers

The following example statement gives permissions to the IAM role in your instance profile to run the aws-sqsd daemon in the worker environment tier, and publish metrics to CloudWatch. For worker environment tiers with an application that performs periodic tasks, the statement also includes permissions to access DynamoDB.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QueueAccess",
      "Action": [
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "MetricsAccess",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*-your-account-ID-without-hyphens/*",
        "arn:aws:s3:::elasticbeanstalk-*-your-account-ID-without-hyphens-*/*"
      ]
    },
    {
      "Sid": "DynamoPeriodicTasks",
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:*:your-account-ID-without-hyphens:table/*-stack-
AWSEBWorkerCronLeaderRegistry*"
      ]
    }
  ]
}
```

If you are using a different account with an unmanaged Amazon SQS queue, you must also edit the policy on the queue to grant access to the queue to other accounts, such as the one that you use with the worker tier. For an example statement, see Example: Using a resource-based policy to delegate access to an Amazon SQS queue in another account in the AWS Identity and Access Management User Guide.

Granting IAM Role Permissions to Access an Amazon S3 Bucket

The following example policy grants read-only permission to the Amazon S3 bucket "my-bucket" to the IAM role with the role name "janedoe". In IAM, this policy is called a resource-based policy. The resource in the example is the Amazon S3 bucket. (In Amazon S3, this is referred to as a bucket policy.) You can use the IAM management console to create your own policy from scratch. Give the policy the name "S3ReadOnlyPerms" and replace the text "your-account-ID-without-hyphens" with your own account ID. For more information, see Creating Customer Managed Policies.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3ReadOnlyPerms",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::your-account-ID-without-hyphens{:role/janedoe"
      },
      "Action": [
        "s3:ListBucketVersions",
        "s3:GetObjectVersion",
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket/*",
        "arn:aws:s3:::my-bucket"
      ]
    }
  ]
}
```

Example: Granting Permissions to Elastic Beanstalk Applications to Access DynamoDB

This walkthrough shows you how to use IAM roles with Elastic Beanstalk to control access to DynamoDB and assumes that you have the necessary IAM permissions to create and pass roles. To learn more about developing applications with DynamoDB, see the Amazon DynamoDB Developer Guide. In this walkthrough, you will do the following:

1. Create and configure an IAM role for Elastic Beanstalk.
2. Update your application to obtain temporary security credentials to make AWS API calls to DynamoDB.
3. Deploy your application to Elastic Beanstalk.

In this walkthrough, you use the IAM console to create an IAM role. However, Elastic Beanstalk can also create a default instance profile for you when you launch or update your Elastic Beanstalk environment. To allow your application access to other AWS resources, you simply attach a new custom policy to the default role. For instructions on creating a new application using the Elastic Beanstalk console with a default instance profile, see Create an Application (p. 34). For instructions on updating an existing environment, see Instance Profiles (p. 202). For instructions for the EB CLI, see Tools (p. 384).

Step 1: Create and Configure an IAM Role for Elastic Beanstalk

When you create and configure an IAM role, you are creating a role and defining the policies for who can assume the role and granting permissions to perform actions on Elastic Beanstalk resources. There are two ways you can create and configure an IAM role:

- Elastic Beanstalk can create a default role when you deploy your application and attach an action policy.
- Use the IAM console to create an IAM role and attach an action policy.

This section walks you through both methods. If you use Elastic Beanstalk to create a default role, then Elastic Beanstalk will automatically update the Amazon S3 bucket policy to allow log rotation. If you use a custom role, you need to attach a policy that grants Elastic Beanstalk permissions to rotate logs. For more information about the policy, see Using a Custom Instance Profile (p. 341).

In this procedure, we use the Elastic Beanstalk to create a default role and then attach a custom policy for DynamoDB.

To attach an action policy to the Elastic Beanstalk default role

1. Deploy a sample application to Elastic Beanstalk using one of the following methods. When prompted, select to create a default instance profile.

   - Elastic Beanstalk console — Create an Application (p. 34)
   - The EB CLI — Tools (p. 384)
   - AWS Toolkit for Eclipse — Develop, Test, and Deploy (p. 576)
   - AWS Toolkit for Visual Studio — Develop, Test, and Deploy (p. 627)

   Elastic Beanstalk creates an IAM role called `aws-elasticbeanstalk-ec2-role`.
2. Attach an action policy to the `aws-elasticbeanstalk-ec2-role` role.

   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

   b. In the left pane, click **Roles**.

   c. From the **Roles** pane, click the `aws-elasticbeanstalk-ec2-role` role.

   d. Inside the **Permissions** tab, click **Attach Role Policy**.

   e. Click **Custom Policy** and then type the following action policy to give Elastic Beanstalk permission to get data from items in the AWS Account's tables.

      Note

      In the Amazon Resource Name (ARN), replace the example region `us-west-2` with the region that has the tables with the data you need. Similarly, replace the example account number `123456789012` with your AWS account ID.

      To find your AWS account number, go to the AWS Management Console and click **My Account**. Your AWS account number is shown in the upper right portion of the **Manage Your Account** page. Do not include dashes in your AWS account ID in the ARN in the policy document.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem"
       ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
    }
  ]
}
```

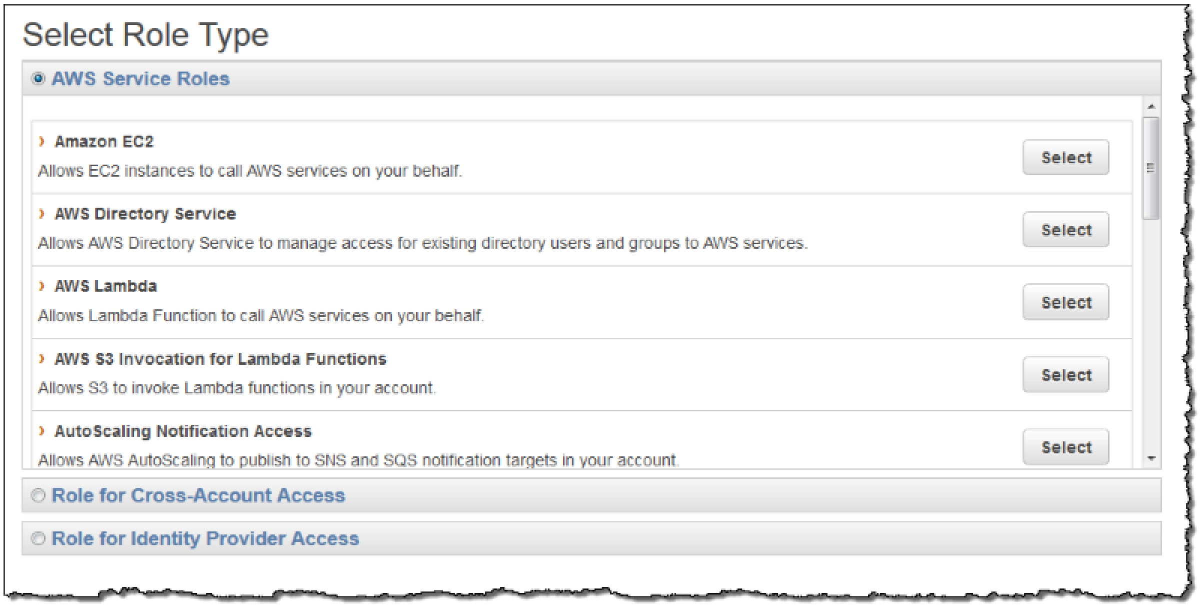
   f. Click **Apply Policy**.

   For more information on managing policies, go to Managing IAM Policies in *Using AWS Identity and Access Management*.

In this procedure, we use the IAM console to create an IAM role and a custom policy for DynamoDB.

To create a role and attach an action policy to the role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left pane, click **Roles**.
3. Click **Create New Role** to launch the Create Role Wizard.
4. On the **Set Role Name** page, in the **Role Name** box, enter the name of the role. Click **Next Step**.
5. On the **Configure Role** page, next to **Amazon EC2**, click **Select** to allow EC2 instances to call AWS services on your behalf.



Select Role Type

● AWS Service Roles

› Amazon EC2
Allows EC2 instances to call AWS services on your behalf.

› AWS Directory Service
Allows AWS Directory Service to manage access for existing directory users and groups to AWS services.

› AWS Lambda
Allows Lambda Function to call AWS services on your behalf.

› AWS S3 Invocation for Lambda Functions
Allows S3 to invoke Lambda functions in your account.

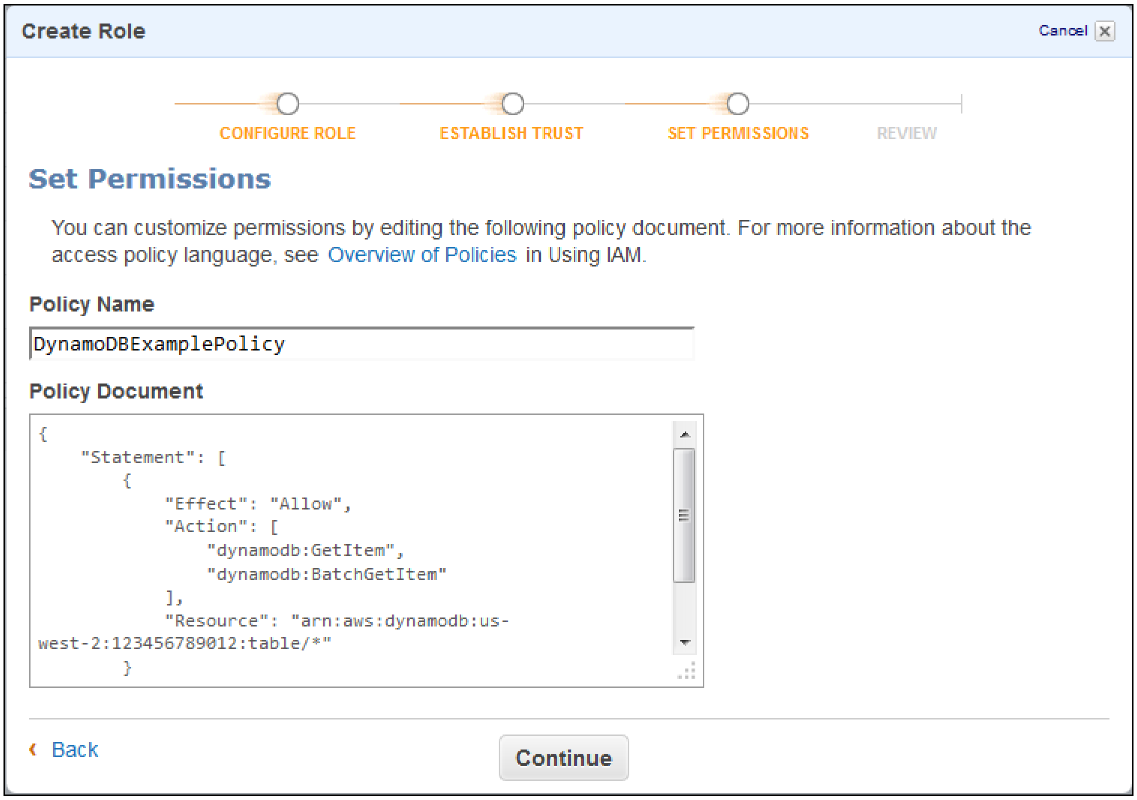
› AutoScaling Notification Access
Allows AWS AutoScaling to publish to SNS and SQS notification targets in your account.

○ Role for Cross-Account Access

○ Role for Identity Provider Access

6. On the **Attach Policy** page, click **Custom Policy**.
7. Next to the name of the policy that you created earlier, select the check box. Click **Next Step**.
8. On the **Review** page, verify the information that you entered, and then click **Create Role**.

   IAM creates a role and an instance profile associated with that IAM role. The name of the instance profile is the same as the role. This instance profile allows your application running on the Amazon EC2 instances to gain access to temporary security credentials so that it can make AWS API requests.
9. Type the name of the policy and the policy information, and click **Continue**. The following diagram shows an example policy that gives Elastic Beanstalk permission to get data from items in the AWS Account's tables.



Set Permissions

You can customize permissions by editing the following policy document. For more information about the access policy language, see Overview of Policies in Using IAM.

Policy Name: DynamoDBExamplePolicy

Policy Document

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem"
            ],
            "Resource": "arn:aws:dynamodb:us-
west-2:123456789012:table/*"
        }
```

For more example policies for DynamoDB, go to Example Policies for Amazon DynamoDB in *Amazon DynamoDB Developer Guide* .

10. On the **Review** page, click **Create Role**.

An IAM role and an instance profile associated with that role are created. The name of the instance profile is the same as the role. This instance profile allows your applications running on the EC2 instances to gain access to temporary security credentials so that it can make AWS API requests.

Step 2: Update Your Application to Access Temporary Credentials

Now that you have created and configured an IAM role, your application can use the instance profile associated with that role to obtain temporary security credentials to make AWS API calls. When you deploy your application to Elastic Beanstalk, Elastic Beanstalk launches the EC2 instances using the instance profile you specify. Your application uses the role credentials that are available on the EC2 instance. Your application retrieves the role credentials from the Instance Meta Data Service (IMDS), and then makes API calls to DyanmoDB using those credentials. For more information about using IAM roles with EC2, go to Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources in the *AWS Identity and Access Management Using IAM*.

If you use the AWS SDKs, the software constructs a client object for an AWS service, using an overload of the constructor that does not take any parameters. When this parameterless constructor executes, it searches the "credentials provider chain." The credentials provider chain is the set of places where the constructor attempts to find credentials if they are not specified explicitly as parameters. The sequence of places where the constructor will attempt to check varies depending on the programming language. Check the corresponding SDK documentation for details. You can also have the SDK automatically use the IAM role credentials from the IMDS by specifying a parameter. In this section, we provide code examples using the SDKs to obtain role credentials.

Java

```
AmazonDynamoDB client = new AmazonDynamoDBClient(new InstanceProfileCreden
tialsProvider());
```

For more information, go to Using IAM Roles for EC2 Instances with the SDK for Java in the *AWS SDK for Java Developer Guide*.

.NET

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient(new InstanceProfileAWSCre
dentials());
```

For more information, go to Using IAM Roles for EC2 Instances with the SDK for .NET in the *AWS SDK for .NET Developer Guide*.

PHP

```
$dynamoDB = new AmazonDynamoDB(array(
  'default_cache_config' => '/tmp/secure-dir'
));
```

For more information, go to Using IAM Roles for EC2 Instances with the SDK for PHP in the *AWS SDK for PHP Developer Guide*.

Python

```
import boto
conn = boto.connect_dynamodb()
```

For more information, go to boto: A Python interface to Amazon Web Services.

Ruby

Example for specifying role credentials:

```
AWS.config(:credential_provider => AWS::Core::CredentialProviders::EC2Pro
vider.new)
```

For Ruby, the credentials provider chain is static credentials in AWS.config, environment variables, and then the IMDS.

Example without specifying role credentials:

```
ddb = AWS::DynamoDB.new
```

For more information, go to Using IAM Roles for Amazon EC2 Instances with the AWS SDK for Ruby in the *AWS SDK for Ruby Developer Guide*.

Step 3: Deploy to Elastic Beanstalk

After you update your application, you can deploy it to Elastic Beanstalk using your instance profile.

If you used Elastic Beanstalk to create a default instance profile, then you can redeploy your updated application to your environment. For instructions, see one of the following:

- Elastic Beanstalk console — Create an Application Version (p. 39)
- The EB CLI — Tools (p. 384)
- AWS Toolkit for Eclipse — Edit the Application and Redeploy (p. 583)
- AWS Toolkit for Visual Studio — Edit the Application and Redeploy (p. 635)

If you used the IAM console to create the role, the IAM console automatically creates and manages the instance profile for you. The instance profile has the same name as the role you created. Use one of the following methods to deploy your application to Elastic Beanstalk. When prompted, select the instance profile you just created.

- Elastic Beanstalk console — Create an Application (p. 34)
- The EB CLI — Tools (p. 384)
- AWS Toolkit for Eclipse — Develop, Test, and Deploy (p. 576)
- AWS Toolkit for Visual Studio — Develop, Test, and Deploy (p. 627)

Example: Granting Elastic Beanstalk Permission to Rotate Logs to Amazon S3

Elastic Beanstalk can copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application. Elastic Beanstalk requires permissions to access this Amazon S3 bucket. To grant Elastic Beanstalk permission to rotate logs, you can do one of two things:

- Create a default instance profile using the Elastic Beanstalk console or eb.
- Use a custom IAM role and attach an action policy to grant permission to Elastic Beanstalk.

This topic walks you through granting Elastic Beanstalk permission to rotate logs using both methods and assumes that you have the necessary IAM permissions to create and pass roles.

Creating a Default Instance Profile

When you deploy your application to Elastic Beanstalk, Elastic Beanstalk can create a default instance profile for you. Elastic Beanstalk creates a default instance profile called `aws-elasticbeanstalk-ec2-role` and updates the Amazon S3 bucket policy to allow log rotation. When Elastic Beanstalk creates the default instance profile, it creates a trust policy with no action policies attached. If your application requires access to other AWS resources, you can attach action policies to the default instance profile.

> **Note**
> You must have permission to create a default profile when deploying your application using Elastic Beanstalk. For more information, see Granting IAM Users Permissions to Create and Pass IAM Roles (p. 332).

You can use the Elastic Beanstalk console, eb, or the AWS Toolkits to create a default instance profile when you deploy your application. For instructions, see one of the following:

- Elastic Beanstalk console — Create an Application (p. 34)
- The EB CLI — Tools (p. 384)
- AWS Toolkit for Eclipse — Develop, Test, and Deploy (p. 576)
- AWS Toolkit for Visual Studio — Develop, Test, and Deploy (p. 627)

If you already deployed your application to Elastic Beanstalk, you can also update your environment to use the default instance profile. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199).

Using a Custom Instance Profile

If you want to create an IAM role or use an existing IAM role where EC2 is the trusted entity, you can use that when deploying your application to Elastic Beanstalk. To enable log rotation, you will need to attach a policy to your IAM role to grant Elastic Beanstalk permission to rotate logs. When you deploy your application to Elastic Beanstalk, you use the instance profile associated with the IAM role. If you use the IAM console to create the IAM role, the instance profile is the same name as the role.

To create a custom action policy for Elastic Beanstalk to rotate logs

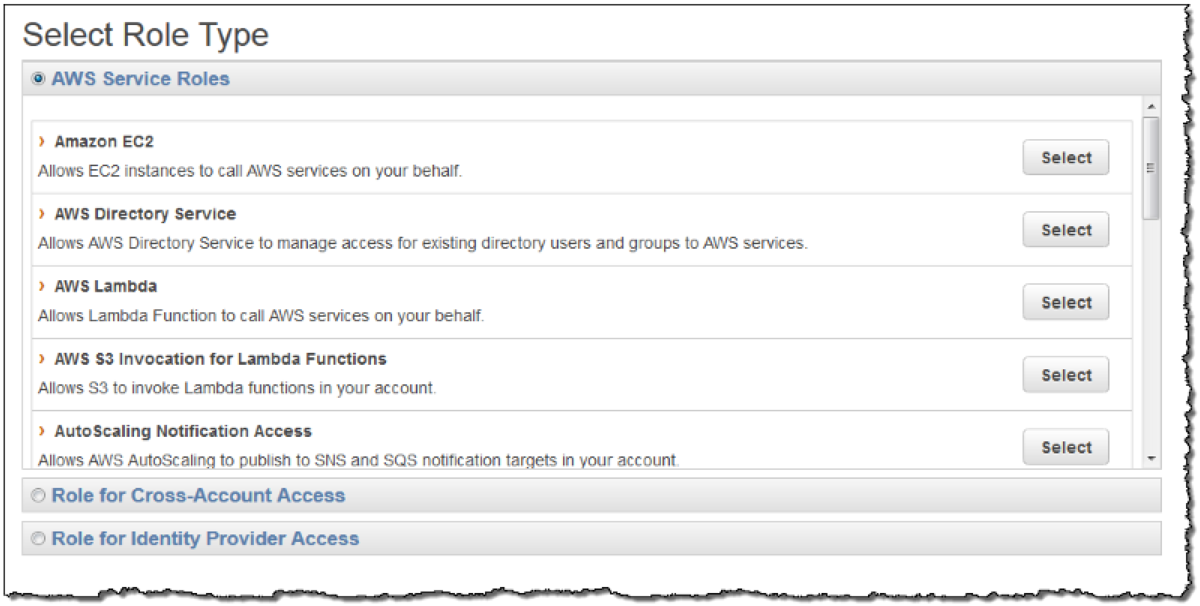
1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left pane, click **Policies**.
3. Click **Create Policy** to launch the Create Policy Wizard.
4. On the **Create Policy** page, next to **Create Your Own Policy**, click **Select**.
5. On the **Review Policy** page, enter a policy name that will help you identify the policy later. For example, **`ElasticBeanstalkRotatesLogs`**.
6. Click **Policy Document**, and then enter the following:
7.
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::elasticbeanstalk-*/resources/environ
ments/logs/*"
    }
  ]
}
```

   Note
   The example policy grants permissions to Amazon S3 buckets for all regions and account IDs. You can replace the example with your Amazon S3 bucket name for Elastic Beanstalk. The Amazon S3 bucket name will be **elasticbeanstalk–*`region`*–*`your account ID`***. The region is the region where you launched your Elastic Beanstalk environment.

8. (Optional) Click **Validate Policy**.
9. When you are ready, click **Create Policy**.

To grant Elastic Beanstalk permission to rotate logs using a new IAM role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left pane, click **Roles**.
3. Click **Create New Role** to launch the Create Role Wizard.
4. On the **Set Role Name** page, in the **Role Name** box, enter the name of the role. Click **Next Step**.
5. On the **Configure Role** page, next to **Amazon EC2**, click **Select** to allow EC2 instances to call AWS services on your behalf.



Select Role Type

● AWS Service Roles

› Amazon EC2
Allows EC2 instances to call AWS services on your behalf.

› AWS Directory Service
Allows AWS Directory Service to manage access for existing directory users and groups to AWS services.

› AWS Lambda
Allows Lambda Function to call AWS services on your behalf.

› AWS S3 Invocation for Lambda Functions
Allows S3 to invoke Lambda functions in your account.

› AutoScaling Notification Access
Allows AWS AutoScaling to publish to SNS and SQS notification targets in your account.

○ Role for Cross-Account Access

○ Role for Identity Provider Access

6. On the **Attach Policy** page, click **Custom Policy**.
7. Next to the name of the policy that you created earlier, select the check box. Click **Next Step**.
8. On the **Review** page, verify the information that you entered, and then click **Create Role**.

   IAM creates a role and an instance profile associated with that IAM role. The name of the instance profile is the same as the role. This instance profile allows your application running on the Amazon EC2 instances to gain access to temporary security credentials so that it can make AWS API requests.
9. Deploy your application to Elastic Beanstalk using one of the following, and when prompted, select the instance profile you just created.

   - Elastic Beanstalk console — Create an Application (p. 34)
   - The EB CLI — Tools (p. 384)
   - AWS Toolkit for Eclipse — Develop, Test, and Deploy (p. 576)
   - AWS Toolkit for Visual Studio — Develop, Test, and Deploy (p. 627)

   If you already deployed your application to Elastic Beanstalk, you can also update your environment to use the default instance profile. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199).

To grant Elastic Beanstalk permission to rotate logs using an existing IAM role

1. Make sure you created an IAM role where Amazon EC2 is the trusted entity. You can check your trusted entities using the IAM management console at https://console.aws.amazon.com/iam/. In the list of roles, click the IAM **Role Name**. If you do not see **ec2.amazonaws.com** listed as a trusted entity under **Trust Relationships**, then you need to update your trust policy. To update your trust policy, do the following:

   a. Under **Trust Relationships**, click **Edit Trust Relationship**.
   b. Update the trust policy to include **ec2.amazonaws.com** like in the following snippet.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ec2.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

c. Click **Update Trust Policy**.

2. Attach an action policy to the IAM role that grants Elastic Beanstalk permission to rotate logs. If you have not already created a policy with the appropriate permissions, follow the instructions in the procedure To create a custom action policy for Elastic Beanstalk to rotate logs.

   a. In the IAM management console, in the navigation pane, click **Roles**. on the **Roles** page, click the IAM **Role Name**.
   b. In the list of roles, click the IAM **Role Name** to which you want to attach the action policy.
   c. Under **Permissions**, click **Attach Policy**.
   d. Next to the name of the policy that you created earlier, select the check box, and then click **Attach Policy**.

   For more information on managing policies, go to Managing IAM Policies in *Using AWS Identity and Access Management*.

3. Deploy your application to Elastic Beanstalk using one of the following, and when prompted, select the instance profile.

   - Elastic Beanstalk console — Create an Application (p. 34)
   - The EB CLI — Tools (p. 384)
   - AWS Toolkit for Eclipse — Develop, Test, and Deploy (p. 576)
   - AWS Toolkit for Visual Studio — Develop, Test, and Deploy (p. 627)

   If you already deployed your application to Elastic Beanstalk, you can also update your environment to use the default instance profile. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199).

Amazon Resource Name (ARN) Format for Elastic Beanstalk

You specify a resource for an IAM policy using that resource's Amazon Resource Name (ARN). For Elastic Beanstalk, the ARN has the following format.

```
arn:aws:elasticbeanstalk:region:accountid:resourcetype/resourcepath
```

Where:

- *region* is the region the resource resides in (for example, **us-west-2**).
- *accountid* is the AWS account ID, with no hyphens (for example, **123456789012**)
- *resourcetype* identifies the type of the Elastic Beanstalk resource—for example, environment. See the table below for a list of all Elastic Beanstalk resource types.
- *resourcepath* is the portion that identifies the specific resource. An Elastic Beanstalk resource has a path that uniquely identifies that resource. See the table below for the format of the resource path for each resource type. For example, an environment is always associated with an application. The resource path for the environment **myEnvironment** in the application **myApp** would look like this:

```
myApp/myEnvironment
```

Elastic Beanstalk has several types of resources you can specify in a policy. The following table shows the ARN format for each resource type and an example.

Resource Type	Format for ARN
application	arn:aws:elasticbeanstalk:*region*:*accountid*:application/*applicationname* Example: **arn:aws:elasticbeanstalk:us-west-2:123456789012:application/My App**
applicationversion	arn:aws:elasticbeanstalk:*region*:*accountid*:applicationversion/*applicationname*/*versionlabel* Example: **arn:aws:elasticbeanstalk:us-west-2:123456789012:applicationversion/My App/My Version**
environment	arn:aws:elasticbeanstalk:*region*:*accountid*:environment/*applicationname*/*environmentname* Example: **arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My App/MyEnvironment**
solutionstack	arn:aws:elasticbeanstalk:*region*::solutionstack/*solutionstackname* Example: **arn:aws:elasticbeanstalk:us-west-2::solutionstack/32bit Amazon Linux running Tomcat 7**
template	arn:aws:elasticbeanstalk:*region*:*accountid*:template/*applicationname*/*templatename* Example: **arn:aws:elasticbeanstalk:us-west-2:123456789012:template/My App/My Template**

An environment, application version, and configuration template are always contained within a specific application. You'll notice that these resources all have an application name in their resource path so that they are uniquely identified by their resource name and the containing application. Although solution stacks are used by configuration templates and environments, solution stacks are not specific to an application or AWS account and do not have the application or AWS account in their ARNs.

Resources and Conditions for Elastic Beanstalk Actions

Topics

This section describes the resources and conditions that you can use in policy statements to grant permissions that allow specific Elastic Beanstalk actions to be performed on specific Elastic Beanstalk resources.

> **Note**
>
> Some Elastic Beanstalk actions may require permissions to other AWS services. For example, the following policy gives permissions for all Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) actions required to complete any Elastic Beanstalk action. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment. For a list of supported non-legacy container types, see Why are some container types marked legacy? (p. 92).
>
> The following policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
       "ec2:*",
       "elasticloadbalancing:*",
       "autoscaling:*",
       "cloudwatch:*",
       "s3:*",
       "sns:*",
       "rds:*",
       "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Policy Information for Elastic Beanstalk Actions

The following table lists all Elastic Beanstalk actions, the resource that each action acts upon, and the additional contextual information that can be provided using conditions.

Conditions enable you to specify permissions to resources that the action needs to complete. For example, when you can call the `CreateEnvironment` action, you must also specify the application version to deploy as well as the application that contains that application name. When you set permissions for the `CreateEnvironment` action, you specify the application and application version that you want the action to act upon by using the `InApplication` and `FromApplicationVersion` conditions. In addition, you can specify the environment configuration with a solution stack (`FromSolutionStack`) or a configuration template (`FromConfigurationTemplate`). The following policy statement allows the `CreateEnvironment` action to create an environment with the name **`myenv`** (specified by `Resource`) in the application **`My App`** (specified by the `InApplication` condition) using the application version **`My Version`** (`FromApplicationVersion`) with a **`32bit Amazon Linux running Tomcat 7`** configuration (`FromSolutionStack`):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
App/myenv"
      ],
      "Condition": {
        "StringEquals": {
         "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbean
stalk:us-west-2:123456789012:applicationversion/My App/My Version"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
west-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

As you can see in the preceding example, resources are specified using their Amazon Resource Name (ARN). For more information about the ARN format for Elastic Beanstalk resources, see Amazon Resource Name (ARN) Format for Elastic Beanstalk (p. 343).

The Comments column contains a simple example statement that grants permission to use the action on a specific resource with the appropriate contextual information provided through one or more conditions. The Comments column also lists dependencies that the action may have on permissions to perform other actions or to access other resources.

> **Note**
>
> If you set a policy on `elasticbeanstalk:Describe*` actions, those actions return only values that are permitted through the policy. For example, the following policy allows the `elasticbeanstalk:DescribeEvents` action to return a list of event descriptions for the environment **`myenv`** in the application **`My App`**. If you applied this policy to a user, that user could successfully perform the `elasticbeanstalk:DescribeEvents` action using **`myenv`** for the `EnvironmentName` parameter to get the list of events for **`myenv`**. However, if the user used another environment name for `EnvironmentName` or specified different parameters such as one for a specific application version, the action would return no event descriptions because the user has permission to view only**`myenv`** events. If the user specified no parameters for `elasticbeanstalk:DescribeEvents`, the action would return only the events for **`myenv`** because that is the only resource the user has permissions for.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "elasticbeanstalk:DescribeEvents",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
 App/myenv"
      ],
      "Condition": {
        "StringEquals": {
         "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

Policy information for Elastic Beanstalk actions, including resources, conditions, examples, and dependencies

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td colspan="3"><b>Action:</b> <code>CheckDNSAvailability</code></td></tr>
<tr><td><code>"*"</code></td><td>N/A</td><td>This example allows the <code>CheckDNSAvailability</code> action to check if a CNAME is available. Note that permission to a resource is not required for this action and the resource should be specified by <code>"*"</code>.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CheckDNSAvailabil
ity"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> <code>CreateApplication</code></td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>application</td><td>N/A</td><td>

This example allows the CreateApplication action to create applications whose names begin with **DivA**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateApplication"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/DivA*"
      ]
    }
  ]
}
```

</td></tr>
<tr><td colspan="3"><b>Action:</b> CreateApplicationVersion</td></tr>
<tr><td>applicationversion</td><td>InApplication</td><td>

This example allows the CreateApplicationVersion action to create application versions with any name (*****) in the application **My App**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateApplication
Version"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

</td></tr>
<tr><td colspan="3"><b>Action:</b> CreateConfigurationTemplate</td></tr>
</table>

Resource	Conditions	Comments

`configuration-template`	`InApplication`
	`FromApplication`
	`FromApplicationVersion`
	`FromConfigurationTemplate`
	`FromEnvironment`
	`FromSolutionStack`
		This example allows the `CreateConfigurationTemplate` action to create configuration templates whose name begins with **`My Template`** (`My Template*`) in the application **`My App`**:
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`       "elasticbeanstalk:CreateConfiguration`
		`Template"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-west-`
		`2:123456789012:configurationtemplate/My`
		`App/My Template*"`
		`      ],`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "elasticbeanstalk:InApplication":`
		`["arn:aws:elasticbeanstalk:us-west-`
		`2:123456789012:application/My App"],`
		`          "elasticbeanstalk:FromSolution`
		`Stack": ["arn:aws:elasticbeanstalk:us-west-`
		`2::solutionstack/32bit Amazon Linux running`
		`Tomcat 7"]`
		`        }`
		`      }`
		`    }`
		`  ]`
		`}`

Action: `CreateEnvironment`		

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>environment</td><td>InApplication
FromApplicationVersion
FromConfigurationTemplate
FromSolutionStack</td><td>

This example allows the CreateEnvironment action to create an environment whose name is **myenv** in the application **My App** and using the solution stack **32bit Amazon Linux running Tomcat 7**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateEnvironment"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplication
Version": ["arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"],
          "elasticbeanstalk:FromSolution
Stack": ["arn:aws:elasticbeanstalk:us-west-
2::solutionstack/32bit Amazon Linux running
Tomcat 7"]
        }
      }
    }
  ]
}
```

</td></tr>
<tr><td colspan="3"><b>Action:</b> CreateStorageLocation</td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>"*"</td><td>N/A</td><td>This example allows the CreateStorageLocation action to create an Amazon S3 storage location. Note that permission to an Elastic Beanstalk resource is not required for this action, and the resource should be specified by "*".
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateStorageLoca
tion"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> DeleteApplication</td></tr>
<tr><td>application</td><td>N/A</td><td>This example allows the DeleteApplication action to delete the application <b>My App</b>:
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteApplication"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"
      ]
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> DeleteApplicationVersion</td></tr>
</table>

Resource	Conditions	Comments
applicationversion	InApplication	This example allows the DeleteApplicationVersion action to delete an application version whose name is **My Version** in the application **My App**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteApplication
Version"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

Action: DeleteConfigurationTemplate

Resource	Conditions	Comments
configuration-template	InApplication (Optional)	This example allows the DeleteConfigurationTemplate action to delete a configuration template whose name is **My Template** in the application **My App**. Specifying the application name as a condition is optional.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
       "elasticbeanstalk:DeleteConfiguration
Template"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:configurationtemplate/My
App/My Template"
      ]
    }
  ]
}
```

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td colspan="3"><b>Action:</b> <code>DeleteEnvironmentConfiguration</code></td></tr>
<tr><td><code>environment</code></td><td><code>InApplication</code> (Optional)</td><td>This example allows the <code>DeleteEnvironmentConfiguration</code> action to delete a draft configuration for the environment <b><code>myenv</code></b> in the application <b><code>My App</code></b>. Specifying the application name as a condition is optional.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteEnvironment
Configuration"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ]
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> <code>DescribeApplicationVersions</code></td></tr>
<tr><td><code>applicationversion</code></td><td><code>InApplication</code> (Optional)</td><td>This example allows the <code>DescribeApplicationVersions</code> action to describe the application version <b><code>My Version</code></b> in the application <b><code>My App</code></b>. Specifying the application name as a condition is optional.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
       "elasticbeanstalk:DescribeApplication
Versions"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"
      ]
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> <code>DescribeApplications</code></td></tr>
</table>

Resource	Conditions	Comments

application	N/A	This example allows the DescribeApplications action to describe the application My App.
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:DescribeApplica
		tions"
		      ],
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-west-
		2:123456789012:application/My App"
		      ]
		    }
		  ]
		}

Action: DescribeConfigurationOptions		

environment, configuration-template, solutionstack	InApplication (Optional)	This example allows the DescribeConfigurationOptions action to describe the configuration options for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		     "Action": "elasticbeanstalk:DescribeCon
		figurationOptions",
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-west-
		2:123456789012:environment/My App/myenv"
		      ]
		    }
		  ]
		}

Action: DescribeConfigurationSettings		

Resource	Conditions	Comments

environment, configuration-template	InApplication (Optional)	This example allows the DescribeConfigurationSettings action to describe the configuration settings for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`     "Action": "elasticbeanstalk:DescribeConfigurationSettings",`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My App/myenv"`
		`      ]`
		`    }`
		`  ]`
		`}`

Action: DescribeEnvironmentResources		

environment	InApplication (Optional)	This example allows the DescribeEnvironmentResources action to return list of AWS resources for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`     "Action": "elasticbeanstalk:DescribeEnvironmentResources",`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My App/myenv"`
		`      ]`
		`    }`
		`  ]`
		`}`

Action: DescribeEnvironments		

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>environment</td><td>InApplication
(Optional)</td><td>This example allows the DescribeEnvironments action to describe the environments <b>myenv</b> and <b>myotherenv</b> in the application <b>My App</b>. Specifying the application name as a condition is optional.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
     "Action": "elasticbeanstalk:DescribeEn
vironments",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv",
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App2/myother
env"
      ]
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> DescribeEvents</td></tr>
<tr><td>application, applicationversion, configurationtemplate, environment</td><td>InApplication</td><td>This example allows the DescribeEvents action to list event descriptions for the environment <b>myenv</b> and the application version <b>My Version</b> in the application <b>My App</b>.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "elasticbeanstalk:Descri
beEvents",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv",
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
</pre></td></tr>
</table>

Resource	Conditions	Comments
Action: ListAvailableSolutionStacks		
solutionstack	N/A	This example allows the ListAvailableSolutionStacks action to return only the solution stack **32bit Amazon Linux running Tomcat 7**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:ListAvailableSolu
tionStacks"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:elasticbean
stalk:us-west-2::solutionstack/32bit Amazon
Linux running Tomcat 7"
    }
  ]
}
```

Resource	Conditions	Comments
Action: RebuildEnvironment		
environment	InApplication	This example allows the RebuildEnvironment action to rebuild the environment **myenv** in the application **My App**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
       "elasticbeanstalk:RebuildEnvironment"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

Resource	Conditions	Comments
Action: RequestEnvironmentInfo		

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>environment</td><td>InApplication</td><td>This example allows the RequestEnvironmentInfo action to compile information about the environment <b>myenv</b> in the application <b>My App</b>.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:RequestEnvironmentInfo"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-west-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> RestartAppServer</td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td><code>environment</code></td><td><code>InApplication</code></td><td>This example allows the <code>RestartAppServer</code> action to restart the application container server for the environment <b><code>myenv</code></b> in the application <b><code>My App</code></b>.

<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:RestartAppServer"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> <code>RetrieveEnvironmentInfo</code></td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>environment</td><td>InApplication</td><td>

This example allows the RetrieveEnvironmentInfo action to retrieve the compiled information for the environment **myenv** in the application **My App**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:RetrieveEnviron
mentInfo"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

</td></tr>
<tr><td colspan="3">

Action: SwapEnvironmentCNAMEs

</td></tr>
<tr><td>environment</td><td>InApplication
(Optional)
FromEnvironment
(Optional)</td><td>

This example allows the SwapEnvironmentCNAMEs action to swap the CNAMEs for the environments **mysrcenv** and **mydestenv**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:SwapEnvironmentC
NAMEs"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/mysrcenv",

        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/mydestenv"

      ]
    }
  ]
}
```

</td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td colspan="3"><b>Action:</b> <code>TerminateEnvironment</code></td></tr>
<tr><td><code>environment</code></td><td><code>InApplication</code></td><td>This example allows the <code>TerminateEnvironment</code> action to terminate the environment <b><code>myenv</code></b> in the application <b><code>My App</code></b>.
<pre><code>{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:TerminateEnviron
ment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}</code></pre></td></tr>
<tr><td colspan="3"><b>Action:</b> UpdateApplication</td></tr>
<tr><td><code>application</code></td><td>N/A</td><td>This example allows the <code>UpdateApplication</code> action to update properties of the application <b><code>My App</code></b>.
<pre><code>{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplication"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"
      ]
    }
  ]
}</code></pre></td></tr>
<tr><td colspan="3"><b>Action:</b> <code>UpdateApplicationVersion</code></td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>applicationversion</td><td>InApplication</td><td>

This example allows the UpdateApplicationVersion action to update the properties of the application version **My Version** in the application **My App**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplication
Version"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

</td></tr>
<tr><td colspan="3"><b>Action:</b> UpdateConfigurationTemplate</td></tr>
</table>

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>configuration-template</td><td>InApplication</td><td>This example allows the UpdateConfigurationTemplate action to update the properties or options of the configuration template <b>My Template</b> in the application <b>My App</b>.
<pre>
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
       "elasticbeanstalk:UpdateConfigurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:configurationtemplate/My App/My Template"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-west-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
</pre></td></tr>
<tr><td colspan="3"><b>Action:</b> UpdateEnvironment</td></tr>
</table>

Resource	Conditions	Comments
environment	InApplication FromApplicationVersion FromConfigurationTemplate	This example allows the UpdateEnvironment action to update the environment **myenv** in the application **My App** by deploying the application version **My Version**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"

      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplication
Version": ["arn:aws:elasticbeanstalk:us-west-
2:123456789012:applicationversion/My App/My
Version"]
        }
      }
    }
  ]
}
```

Action: ValidateConfigurationSettings

<table>
<tr><th>Resource</th><th>Conditions</th><th>Comments</th></tr>
<tr><td>template, environment</td><td>InApplication</td><td>This example allows the ValidateConfigurationSettings action to validates configuration settings against the environment <b>myenv</b> in the application <b>My App</b>.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:ValidateConfigura
tionSettings"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-
2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication":
["arn:aws:elasticbeanstalk:us-west-
2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```
</td></tr>
</table>

Condition Keys for Elastic Beanstalk Actions

Keys enable you to specify conditions that express dependencies, restrict permissions, or specify constraints on the input parameters for an action. Elastic Beanstalk supports the following keys.

InApplication

Specifies the application that contains the resource that the action operates on.

The following example allows the UpdateApplicationVersion action to update the properties of the application version **My Version**. The InApplication condition specifies **My App** as the container for **My Version**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplicationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
       "arn:aws:elasticbeanstalk:us-west-2:123456789012:applicationversion/My
 App/My Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

`FromApplicationVersion`

Specifies an application version as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **myenv** in the application **My App**. The `FromApplicationVersion` condition constrains the `VersionLabel` parameter to allow only the application version **My Version** to update the environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:application/My App"],
         "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbean
stalk:us-west-2:123456789012:applicationversion/My App/My Version"]
        }
      }
    }
  ]
}
```

`FromConfigurationTemplate`

Specifies a configuration template as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **myenv** in the application **My App**. The `FromConfigurationTemplate` condition constrains the `TemplateName` parameter to allow only the configuration template **My Template** to update the environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:application/My App"],
          "elasticbeanstalk:FromConfigurationTemplate": ["arn:aws:elastic
beanstalk:us-west-2:123456789012:configurationtemplate/My App/My Template"]

        }
      }
    }
  ]
}
```

`FromEnvironment`

Specifies an environment as a dependency or a constraint on an input parameter.

The following example allows the `SwapEnvironmentCNAMEs` action to swap the CNAMEs in **`My App`** for all environments whose names begin with **`mysrcenv`** and **`mydestenv`** but not those environments whose names begin with **`mysrcenvPROD*`** and **`mydestenvPROD*`**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:SwapEnvironmentCNAMEs"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
App/mysrcenv*",
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:environment/My
App/mydestenv*"
      ],
      "Condition": {
        "StringNotLike": {
         "elasticbeanstalk:FromEnvironment": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:environment/My App/mysrcenvPROD*"],
         "elasticbeanstalk:FromEnvironment": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:environment/My App/mydestenvPROD*"
          ]
        }
      }
    }
  ]
}
```

`FromSolutionStack`

Specifies a solution stack as a dependency or a constraint on an input parameter.

This example allows the `CreateConfigurationTemplate` action to create configuration templates whose name begins with **`My Template`** (`My Template*`) in the application **`My App`**. The

FromSolutionStack condition constrains the solutionstack parameter to allow only the solution stack **32bit Amazon Linux running Tomcat 7** as the input value for that parameter.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateConfigurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-west-2:123456789012:configurationtem
plate/My App/My Template*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
west-2:123456789012:application/My App"],
         "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
west-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

Example Policies Based on Managed Policies

This section demonstrates how to control user access to AWS Elastic Beanstalk (Elastic Beanstalk) and includes example policies that provide the required access for common scenarios. These policies are derived from the Elastic Beanstalk managed policies. For information about attaching managed policies to users and groups, see Using Managed Policies to Control Access to All Elastic Beanstalk Resources (p. 326).

In this scenario, Example Corp. is a software company with three teams responsible for the company website: administrators who manage the infrastructure, developers who build the software for the website, and a QA team that tests the website. To help manage permissions to their Elastic Beanstalk resources, Example Corp. creates three groups to which members of each respective team belong: Admins, Developers, and Testers. Example Corp. wants the Admins group to have full access to all applications, environments, and their underlying resources so that they can create, troubleshoot, and delete all Elastic Beanstalk assets. Developers require permissions to view all Elastic Beanstalk assets and to create and deploy application versions. Developers should not be able to create new applications or environments or terminate running environments. Testers need to view all Elastic Beanstalk resources to monitor and test applications. The Testers should not be able to make changes to any Elastic Beanstalk resources.

The following example policies provide the required permissions for each group.

Example 1: Allow the Admins group to use all Elastic Beanstalk and related service APIs

The following policy gives users permissions for all actions required to use Elastic Beanstalk. This policy also allows Elastic Beanstalk to provision and manage resources on your behalf in the following services. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment.

- Amazon Elastic Compute Cloud
- Elastic Load Balancing
- Auto Scaling
- Amazon CloudWatch
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Relational Database Service
- AWS CloudFormation

> **Note**
>
> The following policy is an example. It gives a broad set of permissions to the AWS services that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an AWS Identity and Access Management (IAM) user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"
      ],
      "Resource" : "*"
    }
  ]
}
```

Example 2: Allow the Developers group to perform all actions except highly privileged operations, such as creating applications and environments

The following policy denies permission to create applications and environments, but allows all other Elastic Beanstalk actions.

> **Note**
>
> The following policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:DeleteApplication",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment"],
      "Effect" : "Deny",
      "Resource" : "*"
    },
    {
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"],
      "Effect" : "Allow",
      "Resource" : "*"
    }
  ]
}
```

Example 3: Allow the Testers group to view all Elastic Beanstalk assets, but not to perform any actions

The following policy allows read-only access to all applications, application versions, events, and environments.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "rds:Describe*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*"
      ],
      "Resource" : "*"
    }
  ]
}
```

Example Policies Based on Resource Permissions

This section walks through a use case for controlling user permissions for Elastic Beanstalk actions that access specific Elastic Beanstalk resources. We'll walk through the sample policies that support the use case. For more information policies on Elastic Beanstalk resources, see Creating Policies to Control Access to Specific Elastic Beanstalk Resources (p. 329). For information about attaching policies to users and groups, go to Managing IAM Policies in *Using AWS Identity and Access Management*.

In our use case, Example Corp. is a small consulting firm developing applications for two different customers. John is the development manager overseeing the development of the two Elastic Beanstalk applications, app1 and app2. John does development and some testing on the two applications, and only he can update the production environment for the two applications. These are the permissions that he needs for app1 and app2:

- View application, application versions, environments, and configuration templates
- Create application versions and deploy them to the staging environment
- Update the production environment

- Create and terminate environments

Jill is a tester who needs access to view the following resources in order to monitor and test the two applications: applications, application versions, environments, and configuration templates. However, she should not be able to make changes to any Elastic Beanstalk resources.

Jack is the developer for app1 who needs access to view all resources for app1 and also needs to create application versions for app1 and deploy them to the staging environment.

Joe is the administrator of the AWS account for Example Corp. He has created IAM users for John, Jill, and Jack and attaches the following policies to those users to grant the appropriate permissions to the app1 and app2 applications.

Example 1: Policies that allow John to perform his development, test, and deployment actions on app1 and app2

We have broken down John's policy into three separate policies so that they are easier to read and manage. Together, they give John the permissions he needs to perform the Elastic Beanstalk actions on the two applications.

The first policy specifies actions for Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types). Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment. For a list of supported non-legacy container types, see Why are some container types marked legacy? (p. 92).

> **Note**
> The following policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:*",
            "elasticloadbalancing:*",
            "autoscaling:*",
            "cloudwatch:*",
            "s3:*",
            "sns:*",
            "rds:*",
            "cloudformation:*"
         ],
         "Resource":"*"
      }
   ]
}
```

The second policy specifies the Elastic Beanstalk actions that John is allowed to perform on the app1 and app2 resources. The `AllCallsInApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) performed on all resources within app1 and app2 (for example, `elasticbeanstalk:CreateEnvironment`). The `AllCallsOnApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) on the app1 and app2 application resources (for example, `elasticbeanstalk:DescribeApplications`, `elasticbeanstalk:UpdateApplication`, etc.). The `AllCallsOnSolutionStacks` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) for solution stack resources (for example, `elasticbeanstalk:ListAvailableSolutionStacks`).

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllCallsInApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app1",
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app2"
               ]
            }
         }
      },
      {
         "Sid":"AllCallsOnApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
           "arn:aws:elasticbeanstalk:us-west-2:123456789012:application/app1",

            "arn:aws:elasticbeanstalk:us-west-2:123456789012:application/app2"

         ]
      },
      {
         "Sid":"AllCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-west-2::solutionstack/*"
         ]
      }
   ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateApplication`, `elasticbeanstalk:CreateEnvironment`, and other actions.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 2: Policies that allow Jill to test and monitor app1 and app2

We have broken down Jill's policy into three separate policies so that they are easier to read and manage. Together, they give Jill the permissions she needs to perform the Elastic Beanstalk actions on the two applications.

The first policy specifies `Describe*`, `List*`, and `Get*` actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to retrieve the relevant information about the underlying resources of the app1 and app2 applications.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:Describe*",
            "elasticloadbalancing:Describe*",
            "autoscaling:Describe*",
            "cloudwatch:Describe*",
            "cloudwatch:List*",
            "cloudwatch:Get*",
            "s3:Get*",
            "s3:List*",
            "sns:Get*",
            "sns:List*",
            "rds:Describe*",
            "cloudformation:Describe*",
         "cloudformation:Get*",
         "cloudformation:List*",
         "cloudformation:Validate*",
         "cloudformation:Estimate*"
         ],
         "Resource":"*"
      }
   ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jill is allowed to perform on the app1 and app2 resources. The `AllReadCallsInApplications` statement allows her to call the `Describe*` actions and the environment info actions. The `AllReadCallsOnApplications` statement allows her to call the `DescribeApplications` and `DescribeEvents` actions on the app1 and app2 application resources. The `AllReadCallsOnSolutionStacks` statement allows viewing actions that involve solution stack resources (`ListAvailableSolutionStacks`, `DescribeConfigurationOptions`, and `ValidateConfigurationSettings`).

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllReadCallsInApplications",
         "Action":[
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:RequestEnvironmentInfo",
            "elasticbeanstalk:RetrieveEnvironmentInfo"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app1",
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app2"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnApplications",
         "Action":[
            "elasticbeanstalk:DescribeApplications",
            "elasticbeanstalk:DescribeEvents"
         ],
         "Effect":"Allow",
         "Resource":[
           "arn:aws:elasticbeanstalk:us-west-2:123456789012:application/app1",

            "arn:aws:elasticbeanstalk:us-west-2:123456789012:application/app2"

         ]
      },
      {
         "Sid":"AllReadCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:ListAvailableSolutionStacks",
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-west-2::solutionstack/*"
         ]
      }
   ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required for some viewing actions.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 3: Policies that allow Jack to access app1 to test, monitor, create application versions, and deploy to the staging environment

We have broken down Jack's policy into three separate policies so that they are easier to read and manage. Together, they give Jack the permissions he needs to perform the Elastic Beanstalk actions on the app1 resource.

The first policy specifies the actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to view and work with the underlying resources of app1. For a list of supported non-legacy container types, see Why are some container types marked legacy? (p. 92).

Note

The following policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:*",
            "elasticloadbalancing:*",
            "autoscaling:*",
            "cloudwatch:*",
            "s3:*",
            "sns:*",
            "rds:*",
            "cloudformation:*"
         ],
         "Resource":"*"
      }
   ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jack is allowed to perform on the app1 resource.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllReadCallsAndAllVersionCallsInApplications",
         "Action":[
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:RequestEnvironmentInfo",
            "elasticbeanstalk:RetrieveEnvironmentInfo",
            "elasticbeanstalk:CreateApplicationVersion",
            "elasticbeanstalk:DeleteApplicationVersion",
            "elasticbeanstalk:UpdateApplicationVersion"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app1"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnApplications",
         "Action":[
            "elasticbeanstalk:DescribeApplications",
            "elasticbeanstalk:DescribeEvents"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-west-2:123456789012:application/app1"

         ]
      },
      {
         "Sid":"UpdateEnvironmentInApplications",
         "Action":[
            "elasticbeanstalk:UpdateEnvironment"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-west-2:123456789012:environ
ment/app1/app1-staging*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-west-2:123456789012:applica
tion/app1"
               ]
            },
            "StringLike":{
               "elasticbeanstalk:FromApplicationVersion":[
                 "arn:aws:elasticbeanstalk:us-west-2:123456789012:application
version/app1/*"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:ListAvailableSolutionStacks",
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
         ],
         "Effect":"Allow",
         "Resource":[
                "arn:aws:elasticbeanstalk:us-west-2::solutionstack/*"
            ]
        }
    ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateEnvironment`, and other actions.

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

IAM Roles for Elastic Beanstalk Environment Tiers

Elastic Beanstalk recommends as a best practice an IAM role with permissions that an application must have when the application makes calls to other AWS resources. When Elastic Beanstalk launches Amazon EC2 instances, it uses the instance profile associated with an IAM role. All applications that run on the instances can use the role credentials to sign requests. Because role credentials are temporary and rotated automatically, you don't have to worry about long-term security risks.

When you create an application in a web server environment tier, you choose an existing IAM role. When you create an application in a worker environment tier, you can choose an existing IAM or create a new one. Each type of environment tier requires different permissions.

> **Note**
> This topic assumes that you have the necessary IAM permissions to create and instance profile and pass roles. For more information, see Granting IAM Users Permissions to Create and Pass IAM Roles (p. 332).

For an overview of the steps required to grant permissions to applications running in Elastic Beanstalk using IAM roles, see Service Roles, Instance Profiles, and User Policies (p. 19).

The following sections provide example policies for using IAM roles with Elastic Beanstalk environment tiers.

- Granting IAM Role Permissions for Web Server Environment Tiers (p. 382)
- Granting IAM Role Permissions for Worker Environment Tiers (p. 382)

Granting IAM Role Permissions for Web Server Environment Tiers

The first time you launch an Elastic Beanstalk environment on a web server environment tier, Elastic Beanstalk creates a default IAM role called `aws-elasticbeanstalk-ec2-role` in a default instance profile. If this is not your first deployment, the default instance profile already exists and you can choose it during application deployment.

The following statement shows an action policy with the permissions that Elastic Beanstalk grants to the default role to rotate logs to Amazon S3. If you use a different instance profile, ensure that it has these permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Allow",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::elasticbeanstalk-us-east-1-012345678901/re
sources/environments/logs/*"
    }
  ]
}
```

Note
Replace the example Amazon S3 bucket name with your Amazon S3 bucket name for Elastic Beanstalk. The Amazon S3 bucket name will be elasticbeanstalk–*`region`*–*`your-account-ID`*. The region is the region where you launched your Elastic Beanstalk environment.

Granting IAM Role Permissions for Worker Environment Tiers

The following statement shows the permissions for the default `aws-elasticbeanstalk-ec2-worker-role` IAM role for worker environment tiers that you can create and attach to your instance profile. If this is not your first deployment, this instance profile already exists. The permissions enable you to run the aws-sqsd daemon in the worker environment tier, and publish metrics to CloudWatch. For worker environment tiers with an application that performs periodic tasks, the statement also includes permissions to access DynamoDB.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QueueAccess",
      "Action": [
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "MetricsAccess",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*-your-account-ID-without-hyphens/*",
        "arn:aws:s3:::elasticbeanstalk-*-your-account-ID-without-hyphens-*/*"
      ]
    },
    {
      "Sid": "DynamoPeriodicTasks",
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:*:your-account-ID-without-hyphens:table/*-stack-
AWSEBWorkerCronLeaderRegistry*"
      ]
    }
  ]
}
```

If you are using a different account with an unmanaged Amazon SQS queue, you must also edit the policy on the queue to grant access to the queue to other accounts, such as the one that you use with the worker tier. For an example statement, see Example: Using a resource-based policy to delegate access to an Amazon SQS queue in another account in the AWS Identity and Access Management User Guide.

Tools

Topics

The EB Command Line Interface (EB CLI) is a command line interface for Elastic Beanstalk that provides high level commands that simplify creating, updating and monitoring environments from a local repository. Use the EB CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

For information about installing the EB CLI, configuring a repository, and working with environments, see EB Command Line Interface (p. 384).

The AWS Command Line Interface (AWS CLI) is a unified client for AWS services that provides commands for all public API operations. These commands are lower level than those provided by the EB CLI, so it often takes more commands to do an operation with the AWS CLI. On the other hand, the AWS CLI allows you to work with any application or environment running in your account without setting up a repository on your local machine. Use the AWS CLI to create scripts that simplify or automate operational tasks.

For more information about supported services and to download the AWS Command Line Interface, see AWS Command Line Interface. For information about how API-based CLI commands and AWS CLI commands correspond to each other, see AWS Command Line Interface (p. 523).

EB Command Line Interface

Topics

Install the EB Command Line Interface (CLI)

This topic describes how to install the EB Command Line Interface (CLI) version 3. The EB CLI is an interactive command line tool that makes it easy to create an application, configure it for use with Elastic Beanstalk, and deploy it to an Elastic Beanstalk environment. The EB CLI is developed in Python and requires Python version 2.7 or 3.4 in order to run.

The primary distribution method for the EB CLI on Linux, Windows, and OS X is pip, a package manager for Python that provides an easy way to install, upgrade, and remove Python packages and their dependencies.

For OS X, updates to the EB CLI are distributed to both pip and Homebrew repositories. You can use either package manager to install the awsebcli package.

Install the EB CLI with Python and pip

The latest version of the EB CLI is always available in the pip repository. If you already have pip and a supported version of Python, you can install the EB CLI with the following command. Run this command using `sudo` (Linux, OS X, or Unix) or an administrator command prompt (Windows).

```
# pip install awsebcli
```

Note
Amazon Linux comes with Python 2.7 and pip starting with version 2015.03.

The EB CLI is updated regularly to add functionality that supports the latest Elastic Beanstalk features. When you need to update to the latest version of the EB CLI, use the `upgrade` option:

```
# pip install --upgrade awsebcli
```

If you don't have Python and pip, use the procedure for your operating system, as follows.

Topics

Install Python, pip, and the EB CLI on Linux

The EB CLI requires Python 2.7 or later. If your distribution did not come with Python, or came with an older version, install Python before installing pip and the EB CLI.

To install Python 2.7 on Linux

1. Check to see if Python is already installed:

   ```
   $ python --version
   ```

   Note
   If your Linux distribution came with Python, you may need to install the Python developer package in order to get the headers and libraries required to compile extensions and install the EB CLI. Install the developer package (typically named python-dev or python-devel) using your package manager.

2. If Python 2.7 or later is not installed, install it with your distribution's package manager. The command and package name varies:

   - On Debian derivatives such as Ubuntu, use APT:

     ```
     $ sudo apt-get install python2.7
     ```

   - On Red Hat and derivatives, use yum:

     ```
     $ sudo yum install python27
     ```

   - On SUSE and derivatives, use zypper:

     ```
     $ sudo zypper install python
     ```

3. Open a command prompt or shell and run the following command to verify that Python installed correctly:

   ```
   $ python --version
   Python 2.7.9
   ```

Install pip by using the script provided by the Python Packaging Authority, and then install the EB CLI.

To install pip and the EB CLI

1. Download the installation script from pypa.io:

   ```
   $ curl -O https://bootstrap.pypa.io/get-pip.py
   ```

   The script downloads and installs the latest version of pip and another required package named setuptools.
2. Run the script with Python:

   ```
   $ sudo python27 get-pip.py
   Collecting pip
     Downloading pip-6.1.1-py2.py3-none-any.whl (1.1MB)
   Collecting setuptools
     Downloading setuptools-15.0-py2.py3-none-any.whl (501kB)
   Installing collected packages: pip, setuptools
   Successfully installed pip-6.1.1 setuptools-15.0
   ```

   Invoking version 2.7 of Python directly by using the `python27` command instead of `python` ensures that pip is installed in the proper location, even if an older system version of Python is present on your system.
3. Finally, use pip to install the EB CLI.

   ```
   $ sudo pip install awsebcli
   Collecting awsebcli
   ```

```
  Downloading awsebcli-3.2.2.tar.gz (828kB)
Collecting pyyaml>=3.11 (from awsebcli)
  Downloading PyYAML-3.11.tar.gz (248kB)
Collecting cement==2.4 (from awsebcli)
  Downloading cement-2.4.0.tar.gz (129kB)
Collecting python-dateutil<3.0.0,>=2.1 (from awsebcli)
  Downloading python_dateutil-2.4.2-py2.py3-none-any.whl (188kB)
Collecting jmespath>=0.6.1 (from awsebcli)
  Downloading jmespath-0.6.2.tar.gz
Collecting six>=1.5 (from python-dateutil<3.0.0,>=2.1->awsebcli)
  Downloading six-1.9.0-py2.py3-none-any.whl
Installing collected packages: pyyaml, cement, six, python-dateutil, jmespath,
 awsebcli
  Running setup.py install for pyyaml
  Running setup.py install for cement
  Running setup.py install for jmespath
  Running setup.py install for awsebcli
Successfully installed awsebcli-3.2.2 cement-2.4.0 jmespath-0.6.2 python-
dateutil-2.4.2 pyyaml-3.11 six-1.9.0
```

Note

If you installed a new version of Python alongside an older version that came with your distribution, you may get an error like the following when trying to invoke pip with sudo:

```
sudo: pip: command not found
```

To work around this issue, use `which pip` to locate the executable, and then invoke it directly by using an absolute path when installing the EB CLI:

```
$ which pip
/usr/local/bin/pip
$ sudo /usr/local/bin/pip install awsebcli
```

4. Verify that the EB CLI installed correctly by checking its version:

```
$ eb --version
EB CLI 3.2.2 (Python 2.7.9)
```

Install Python, pip, and the EB CLI on Windows

The Python Software Foundation provides installers for Windows that include pip.

To install Python 3.4, pip, and the EB CLI on Windows

1. Install Python 3.4 from the downloads page of Python.org.
2. Add the Python home and scripts directories to the Windows `Path` system variable:

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Python34;C:\Python34\Scripts
```

3. Open the Windows Command Processor from the Start menu.
4. Verify that Python and pip are both installed correctly with the following commands:

```
C:\Windows\System32> python --version
Python 3.4.3
C:\Windows\System32> pip --version
pip 6.0.8 from C:\Python34\lib\site-packages (python 3.4)
```

5. Install the EB CLI using pip:

```
C:\Windows\System32> pip install awsebcli
Collecting awsebcli
  Downloading awsebcli-3.2.2.tar.gz (828kB)
Collecting pyyaml>=3.11 (from awsebcli)
  Downloading PyYAML-3.11.tar.gz (248kB)
Collecting cement==2.4 (from awsebcli)
  Downloading cement-2.4.0.tar.gz (129kB)
Collecting python-dateutil<3.0.0,>=2.1 (from awsebcli)
  Downloading python_dateutil-2.4.2-py2.py3-none-any.whl (188kB)
Collecting jmespath>=0.6.1 (from awsebcli)
  Downloading jmespath-0.6.2.tar.gz
Collecting six>=1.5 (from python-dateutil<3.0.0,>=2.1->awsebcli)
  Downloading six-1.9.0-py2.py3-none-any.whl
Installing collected packages: six, jmespath, python-dateutil, cement,
pyyaml, awsebcli
  Running setup.py install for jmespath
  Running setup.py install for cement
  Running setup.py install for pyyaml
    checking if libyaml is compilable
    Microsoft Visual C++ 10.0 is required (Unable to find vcvarsall.bat).
    skipping build_ext
  Running setup.py install for awsebcli
    Installing eb-script.py script to C:\Python34\Scripts
    Installing eb.exe script to C:\Python34\Scripts
    Installing eb.exe.manifest script to C:\Python34\Scripts
Successfully installed awsebcli-3.2.2 cement-2.4.0 jmespath-0.6.2 python-
dateutil-2.4.2 pyyaml-3.11 six-1.9.0
```

6. Verify that the EB CLI is installed correctly:

```
C:\Windows\System32> eb --version
EB CLI 3.2.2 (Python 3.4.3)
```

Install the EB CLI on OS X

OS X 10.7 and later versions include Python 2.7. With Python 2.7 installed, you can use a script provided by AWS to install the EB CLI. If you have an older version of OS X, you can install Python and pip, and then use pip to install the EB CLI. If you use the Homebrew package manager, you can also install the EB CLI with the brew command (p. 389).

Install the EB CLI on OS X 10.7 or later

AWS provides a script that lets you install the EB CLI on OS x 10.7 and later with a single command. The script installs pip and virtualenv and uses them to install the EB CLI in a virtual environment, avoiding common issues that can occur when using pip with OS X.

To install the EB CLI on OS X 10.7 or later

1. Open a terminal and use the following command to download and run the installation script provided by AWS.

   ```
   $ curl -s https://s3.amazonaws.com/elasticbeanstalk-cli-resources/install-
   ebcli.py | python
   ```

2. Verify that the EB CLI is installed correctly:

   ```
   $ eb --version
   EB CLI 3.2.2 (Python 3.4.3)
   ```

The installation script can be used to upgrade to the latest version of the EB CLI. Run the `curl` command again or save the script locally and run it with Python whenever a new version is available.

If you encounter any issues when using this script, let us know by using the feedback link at the top of the page.

Install Python, pip, and the EB CLI on OS X 10.6 or earlier

If you have an older version of OS X and Python, you can install the latest version of Python and pip and then use them to install the EB CLI.

To install the EB CLI on OS X 10.6 or earlier

1. Install Python 3.4 from the downloads page of Python.org.
2. Install pip by using the script provided by the Python Packaging Authority.

   ```
   $ curl -O https://bootstrap.pypa.io/get-pip.py
   $ python3 get-pip.py
   ```

3. Use pip to install the EB CLI.

   ```
   $ sudo pip install awsebcli
   ```

4. Verify that the EB CLI is installed correctly:

   ```
   $ eb --version
   EB CLI 3.2.2 (Python 3.4.3)
   ```

Install the EB CLI with Homebrew

If you have Homebrew, you can use it to install the EB CLI. The latest version of the EB CLI is typically available from Homebrew a couple of days after it appears in pip.

To install the EB CLI with Homebrew

1. Run `brew install awsebcli`:

```
$ brew install awsebcli
```

2. Verify that the EB CLI is installed correctly:

```
$ eb --version
EB CLI 3.2.2 (Python 3.4.3)
```

Configure the EB CLI

After installing the EB CLI (p. 385), you are ready to configure your project folder with `eb init`.

Topics

Initializing an EB CLI Project

Run `eb init` in your application's project directory to configure the EB CLI and your project.

The following example shows the configuration steps when running `eb init` for the first time in a project folder named `eb`:

To initialize an EB CLI project

1. First, the EB CLI prompts you to select a region. Type the number that corresponds to the region that you would like to use and press **Enter**.

```
~/eb $ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-southeast-1 : Asia Pacific (Singapore)
7) ap-southeast-2 : Asia Pacific (Sydney)
8) ap-northeast-1 : Asia Pacific (Tokyo)
9) sa-east-1 : South America (Sao Paulo)
(default is 3): 3
```

2. Next, provide your access key and secret key so that the EB CLI can manage resources for you. Access keys are created in the AWS Identity and Access Management management console. If you don't have keys, see How Do I Get Security Credentials? in the Amazon Web Services General Reference.

```
You have not yet set up your credentials or your credentials are incorrect
You must provide your credentials.
(aws-access-id): AKIAJOUAASEXAMPLE
(aws-secret-key): 5ZRIrtTM4ciIAvd4EXAMPLEDtm+PiPSzpoK
```

3. An application in Elastic Beanstalk is a resource that contains a set of application versions (source), environments, and saved configurations that are associated with a single web application. Each time you deploy your source code to Elastic Beanstalk using the EB CLI, a new application version is created and added to the list.

   ```
   Select an application to use
   1) [ Create new Application ]
   (default is 1): 1
   ```

4. The default application name is the name of the folder in which you run `eb init`. Enter any name that describes your project.

   ```
   Enter Application Name
   (default is "eb"): eb
   Application eb has been created.
   ```

5. Select a platform that matches the language or framework that your web application is developed in. If you haven't started developing an application yet, choose a platform that you are interested in. You will see how to launch a sample application shortly, and you can always change this setting later.

   ```
   Select a platform.
   1) Node.js
   2) PHP
   3) Python
   4) Ruby
   5) Tomcat
   6) IIS
   7) Docker
   8) Multi-container Docker
   9) GlassFish
   10) Go
   (default is 1): 1
   ```

6. Choose yes to assign an SSH key pair to the instances in your Elastic Beanstalk environment, allowing you to connect directly to them for troubleshooting.

   ```
   Do you want to set up SSH for your instances?
   (y/n): y
   ```

7. Choose an existing key pair or create a new one. If you create a key pair using the EB CLI, the CLI registers the key pair with EC2 for you and stores it locally in a folder named `.ssh` in your user directory.

   ```
   Select a keypair.
   1) [ Create new KeyPair ]
   (default is 1): 1
   ```

Your EB CLI installation is now configured and ready to use. See Using the EB CLI (p. 394) for instructions on creating and working with an Elastic Beanstalk environment.

Ignoring files with .ebignore

You can tell the EB CLI to ignore certain files in your project directory with a `.ebignore` file. This file works like a `.gitignore`. When you deploy your project directory to Elastic Beanstalk and create a new application version, the EB CLI will not include files specified by the `.ebignore` in the source bundle that it creates.

> **Note**
> If no `.ebignore` is present, but a `.gitignore` is, the EB CLI will ignore files specified in the `.gitignore`. If an `.ebignore` file is present, the EB CLI will not read the `.gitignore`.

Using Named Profiles

If you store your credentials as a named profile in a `credentials` or `config` file, you can use the `--profile` (p. 399) option to explicitly specify a profile. For example, the following command creates a new application using the `user2` profile.

```
$ eb init --profile user2
```

You can also change the default profile by setting the `AWS_EB_PROFILE` environment variable. When this variable is set, the EB CLI will read credentials from the specified profile instead of `default` or `eb-cli`.

Linux, OS X, or Unix

```
$ export AWS_EB_PROFILE=user2
```

Windows

```
> set AWS_EB_PROFILE=user2
```

Deploying an Artifact Instead of the Project Folder

You can tell the EB CLI to deploy a ZIP or WAR file that you generate as part of a separate build process by adding the following lines to `.elasticbeanstalk/config.yml` in your project folder.

```
deploy:
  artifact: path/to/buildartifact.zip
```

Access Key Provider Chain

You must provide the EB CLI with a set of AWS credentials—an access key ID and a secret key. The credentials must have permissions that allow the EB CLI to act on your behalf to create the AWS resources for your application's environment, such as Amazon EC2 instances and Amazon S3 buckets. You can use an existing set of credentials if they have appropriate permissions, or you can create a new IAM user and attach a policy that grants the necessary permissions. For more information, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326).

> **Important**
> For security reasons, we strongly recommend that you do *not* provide the EB CLI with your account's root credentials. Instead, grant an IAM user the appropriate permissions and provide

those credentials to the EB CLI. For more information on managing AWS credentials, see Best Practices for Managing AWS Access Keys.

You can provide credentials to the EB CLI in either of the following ways.

- Store the credentials on your system, as described later.

  When you use the EB CLI to create an environment, it automatically uses stored credentials if they exist and have the correct permissions. You can also explicitly specify a particular set of stored credentials.
- Provide the credentials to the EB CLI on the command line.

  If you do not have stored credentials, or the stored credentials lack the required permissions, the EB CLI prompts you for credentials when you create your first environment. For an example, see Using the EB CLI (p. 394).

Note
You are usually prompted for credentials only once. The EB CLI then stores those credentials so you don't need to explicitly provide them again. This means that you will usually use the EB CLI with stored credentials, as described in the following sections.

Storing Credentials

The EB CLI uses the same credentials storage as the AWS CLI and AWS SDKs. The following summarizes the available storage options:

- The AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables.

  You can use these variables to store a single set of credentials.
- An `AWS credentials` file, which is located at `~/.aws/credentials` on Linux and OS X systems or at `C:\Users\USERNAME\.aws\credentials` on Windows systems.

  This file can contain multiple sets of credentials, called profiles, which are identified by name. The default EB CLI profile is named `eb-cli`.
- An AWS CLI `config` file, which is typically `~/.aws/config` on Linux and OS X systems or `C:\Users\`*`USERNAME`*`\.aws\config` on Windows systems.

  The `config` file can also contain multiple profiles, and the default EB CLI profile is also named `eb-cli`. A `config` profile can include additional CLI-specific configuration settings. For example, you can use the `region` setting to specify a default region.

Note
We recommend storing new credentials using one of the preceding approaches. However, if you have an existing EB CLI `config` file (`~/.elasticbeanstalk/config`), the EB CLI can also obtain credentials from that file, as described later. It then creates a profile for those credentials in the AWS CLI `config` file for use by all subsequent commands.

Using Stored Credentials

If you do not specify a profile, the EB CLI uses the following procedure to obtain credentials. Note that if you are familiar with the AWS CLI, the search procedure is similar but not identical.

1. Look for environment variables
Look for `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

- If the variables are not defined, go to Step 2.

- If the variables are defined and the credentials have the required permissions, use them to create the environment.
- If the variables are defined but the credentials lack the required permissions, terminate the search and prompt for credentials (Step 5).

2. Look in the credentials file

Look for the `credentials` file. The credentials must be stored as an `eb-cli` or `default` profile.

- If the credentials file does not exist, or exists but does not include an `eb-cli` or `default` profile, go to Step 3.
- If the file contains an `eb-cli` profile and the credentials have the required permissions, use them to create the environment.
- If the file lacks an `eb-cli` profile but includes a `default` profile, and the `default` credentials have the required permissions, use them to create the environment.
- If either or both profiles exist, but neither has credentials with the required permissions, terminate the search and prompt for credentials (Step 5).

3. Look in the AWS CLI config file

Look for the AWS CLI `config` file. The credentials must be stored as an `eb-cli` or `default` profile. Use the same procedure as Step 2 to evaluate the file.

4. Look for a legacy EB CLI config file

Look for a legacy `config` file (`~/.elasticbeanstalk/config`).

- If this file contains credentials with the required permissions, use them to create the environment.
- Create an `eb-cli` profile with these credentials in the AWS CLI `config` file for use by all subsequent commands.

5. Display a command-line prompt

If the attempt to find appropriate stored credentials fails, the EB CLI prompts for credentials, as follows:

```
You have not yet set up your credentials or your credentials are incorrect
You must provide your credentials.
(aws-access-id): AKIAIOSFODNN7EXAMPLE
(aws-secret-key): wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

- Use these credentials to create the environment.
- Create an `eb-cli` profile with these credentials in the AWS CLI `config` file for use by all subsequent commands.

Using the EB CLI

After installing the EB CLI (p. 385) and configuring your project directory (p. 390), you are ready to create an Elastic Beanstalk environment using the EB CLI.

Topics

Working with Environments

After initializing a project directory with `eb init`, you can use the EB CLI to create and manage environments, deploy source and configuration updates, and pull logs and events.

The following examples use an empty project folder named `eb` that was initialized with the EB CLI for use with a sample Docker application.

Basic Commands

eb create

To create your first environment, run `eb create` and follow the prompts. If your project directory has source code in it, the EB CLI will bundle it up and deploy it to your environment. Otherwise, a sample application will be used.

```
~/eb$ eb create
Enter Environment Name
(default is eb-dev): eb-dev
Enter DNS CNAME prefix
(default is eb-dev): eb-dev
WARNING: The current directory does not contain any source code. Elastic Bean
stalk is launching the sample application instead.
Environment details for: elasticBeanstalkExa-env
  Application name: elastic-beanstalk-example
  Region: us-west-2
  Deployed Version: Sample Application
  Environment ID: e-j3pmc8tscn
  Platform: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
  Tier: WebServer-Standard
  CNAME: eb-dev.elasticbeanstalk.com
  Updated: 2015-06-27 01:02:24.813000+00:00
Printing Status:
INFO: createEnvironment is starting.
 -- Events -- (safe to Ctrl+C) Use "eb abort" to cancel the command.
```

Your environment can take several minutes to become ready. Press **Ctrl**+**C** to return to the command line while the environment is created.

eb status

Run `eb status` to see the current status of your environment. When the status is `ready`, the sample application is available at elasticbeanstalk.com and the environment is ready to be updated.

```
~/eb$ eb status
Environment details for: elasticBeanstalkExa-env
  Application name: elastic-beanstalk-example
  Region: us-west-2
  Deployed Version: Sample Application
  Environment ID: e-gbzqc3jcra
  Platform: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
  Tier: WebServer-Standard
  CNAME: elasticbeanstalkexa-env.elasticbeanstalk.com
  Updated: 2015-06-30 01:47:45.589000+00:00
  Status: Ready
  Health: Green
```

eb events

Use `eb events` to see a list of events output by Elastic Beanstalk.

```
~/eb$ eb open
2015-06-29 23:21:09    INFO    createEnvironment is starting.
2015-06-29 23:21:10    INFO    Using elasticbeanstalk-us-west-2-EXAMPLE as
Amazon S3 storage bucket for environment data.
2015-06-29 23:21:23    INFO    Created load balancer named: awseb-e-g-AWSEBLoa-
EXAMPLE
2015-06-29 23:21:42    INFO    Created security group named: awseb-e-gbzqc3jcra-
stack-AWSEBSecurityGroup-EXAMPLE
2015-06-29 23:21:45    INFO    Created Auto Scaling launch configuration named:
 awseb-e-gbzqc3jcra-stack-AWSEBAutoScalingLaunchConfiguration-EXAMPLE
...
```

eb logs

Use `eb logs` to pull logs from an instance in your environment. By default, `eb logs` pull logs from the first instance launched and displays them in standard output. You can specify an instance ID with the `--instance` option to get logs from a specific instance.

The `--all` option pulls logs from all instances and saves them to subdirectories under `.elasticbeanstalk/logs`.

```
~/eb$ eb logs --all
Retrieving logs...
Logs were saved to /home/local/ANT/mwunderl/ebcli/environments/test/.elastic
beanstalk/logs/150630_201410
Updated symlink at /home/local/ANT/mwunderl/ebcli/environments/test/.elastic
beanstalk/logs/latest
```

eb open

To open your environment's website in a browser, use `eb open`:

```
~/eb$ eb open
```

In a windowed environment, your default browser will open in a new window. In a terminal environment, a command line browser (e.g. w3m) will be used if available.

eb deploy

Once the environment is up and ready, you can update it using `eb deploy`.

This command works better with some source code to bundle up and deploy, so for this example we've created a `Dockerfile` in the project directory with the following content:

~/eb/Dockerfile

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/gab
rielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -
rf 2048-master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

This `Dockerfile` deploys an image of Ubuntu 12.04 and installs the game `2048`. Run `eb deploy` to upload the application to your environment:

```
~/eb$ eb deploy
Creating application version archive "app-150630_014338".
Uploading elastic-beanstalk-example/app-150630_014338.zip to S3. This may take
 a while.
Upload Complete.
INFO: Environment update is starting.
 -- Events -- (safe to Ctrl+C) Use "eb abort" to cancel the command.
```

When you run `eb deploy`, the EB CLI bundles up the contents of your project directory and deploys it to your environment.

> **Note**
> If you have initialized a git repository in your project folder, the EB CLI will always deploy the latest commit, even if you have pending changes. Commit your changes prior to running `eb deploy` to deploy them to your environment.

eb config

Take a look at the configuration options available for your running environment with the `eb config` command:

```
~/eb$ eb config
ApplicationName: elastic-beanstalk-example
DateUpdated: 2015-06-30 02:12:03+00:00
EnvironmentName: elasticBeanstalkExa-env
SolutionStackName: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
settings:
  AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
    LowerBreachScaleIncrement: '-1'
  AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
    UpperBreachScaleIncrement: '1'
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
...
```

This command populates a list of available configuration options in a text editor. Many of the options shown have a `null` value, these are not set by default but can be modified to update the resources in your environment. See Configuration Options (p. 103) for more information about these options.

eb terminate

If you are done using the environment for now, use `eb terminate` to terminate it.

```
~/eb$ eb terminate
The environment "eb-dev" and all associated instances will be terminated.
To confirm, type the environment name: eb-dev
INFO: terminateEnvironment is starting.
INFO: Deleted CloudWatch alarm named: awseb-e-jc8t3pmscn-stack-AWSEBCloud
watchAlarmHigh-1XLMU7DNCBV6Y
INFO: Deleted CloudWatch alarm named: awseb-e-jc8t3pmscn-stack-AWSEBCloud
watchAlarmLow-8IVI04W2SCXS
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:210774411744:scalingPolicy:1753d43e-ae87-4df6-a405-11d31f4c8f97:autoScal
ingGroupName/awseb-e-jc8t3pmscn-stack-AWSEBAutoScalingGroup-90TTS2ZL4MXV:poli
cyName/awseb-e-jc8t3pmscn-stack-AWSEBAutoScalingScaleUpPolicy-A070H1BMUQAJ
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:210774411744:scalingPolicy:1fd24ea4-3d6f-4373-affc-4912012092ba:autoScal
ingGroupName/awseb-e-jc8t3pmscn-stack-AWSEBAutoScalingGroup-90TTS2ZL4MXV:poli
cyName/awseb-e-jc8t3pmscn-stack-AWSEBAutoScalingScaleDownPolicy-LSWFUMZ46H1V
INFO: Waiting for EC2 instances to terminate. This may take a few minutes.
 -- Events -- (safe to Ctrl+C)
```

For a full list of available EB CLI commands, check out the EB CLI Command Reference (p. 399).

Using Git with the EB CLI

The EB CLI provides integration with Git. This section provides an overview of how to use Git with the EB CLI.

To install Git and initialize your Git repository

1. Download the most recent version of Git by going to http://git-scm.com
2. Initialize your Git repository by typing the following:

   ```
   $ git init
   ```

   EB CLI will now recognize that your application is set up with Git.
3. If you haven't already run `eb init`, do that now:

   ```
   $ eb init
   ```

To use different Git branches

- You can associate your environment with different branches of your code so that when you work in a new branch, your default environment also uses that branch. For example, you can type the following to associate the running environment with your master and develop branches:

  ```
  $ git checkout master
  $ eb use prod
  $ git checkout develop
  $ eb use dev
  ```

To assign Git tags to your application version

- You can use a Git tag as your version label to identify what application version is running in your environment. For example, type the following:

```
$ git tag -a v1.0 -m "My version 1.0"
```

Note
If you have already deployed this version, EB CLI will deploy that version to your environment instead of uploading a new application version.

To use Git with EB CLI to deploy only code under source control

1. Make any change to your code, and then type the following:

```
$ git commit
```

Note
EB CLI uses your commit ID and message as the application version label and description, respectively.

2. Deploy your updated code.

Note
Now, when you run `eb deploy`, EB CLI will deploy only the code that was under source control.

EB CLI Command Reference

You can use the AWS Elastic Beanstalk Command Line Interface (EB CLI) to perform a variety of operations to deploy and manage your Elastic Beanstalk applications and environments. EB CLI integrates with Git if you want to deploy application source code that is under Git source control. For more information, see EB Command Line Interface (p. 384) and Using Git with the EB CLI (p. 398).

Common Options

The following options can be used with all EB CLI commands.

Name	Description
`--debug`	Print information for debugging.
`-h`, `--help`	Show the Help message. Type: String Default: None
`--no-verify-ssl`	Skip SSL certificate verification. Use this option if you have issues using the CLI with a proxy.
`--profile`	Use a specific profile from your AWS credentials file.
`--quiet`	Suppress all output from the command.

Name	Description
`--region`	Use the specified region.
`-v`, `--verbose`	Display verbose information.

Commands

abort

Description

Cancels an upgrade when environment configuration changes to instances are still in progress.

> **Note**
> If you have more than two environments that are undergoing a update, you are prompted to select the name of the environment for which you want to roll back changes.

Syntax

`eb abort`

`eb abort` ***`environment_name`***

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Output

The command shows a list of environments currently being updated and prompts you to choose the update that you want to abort. If only one environment is currently being updated, you do not need to specify the environment name. If successful, the command reverts environment configuration changes. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails.

Example

The following example cancels the platform upgrade.

```
$ eb abort
Aborting update to environment "tmp-dev".
<list of events>
```

clone

Description

Clones an environment to a new environment so that both have identical environment settings.

> **Note**
> By default, regardless of the solution stack version of the environment from which you create the clone, the `eb clone` command creates the clone environment with the most recent solution stack. You can suppress this by including the `--exact` option when you run the command.

Syntax

`eb clone`

`eb clone` ***`environment_name`***

Options

Name	Description	Required

`-n` *`string`*
or
`--clone_name` *`string`*
	Desired name for the cloned environment.	No

`-c` *`string`*
or
`--cname` *`string`*
	Desired CNAME prefix for the cloned environment.	No

`--envvars`	Environment variables in a comma-separated list with the format ***`name`*****`=`*****`value`***. Type: String Constraints: • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / + \ - @ • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either **`aws:`** or **`elasticbeanstalk:`**. • The combined size of all environment variables cannot exceed 4096 bytes.	No

Name	Description	Required
`--exact`	Prevents Elastic Beanstalk from updating the solution stack version for the new clone environment to the most recent version available (for the original environment's platform).	No
`--scale` *`number`*	The number of instances to run in the clone environment when it is launched.	No
`--tags` *`name`*`=`*`value`*	Amazon EC2 tags for the environment in a comma-separated list with the format ***`name`*****`=`*****`value`***. Type: String Constraints: • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / + \ - @ • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either **`aws:`** or **`elasticbeanstalk:`**.	No
`--timeout`	The number of minutes before the command times out.	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command creates an environment that has the same settings as the original environment or with modifications to the environment as specified by any **eb clone** options.

Example

The following example clones the specified environment.

```
$ eb clone
Enter name for Environment Clone
(default is tmp-dev-clone):
Enter DNS CNAME prefix
(default is tmp-dev-clone):
Environment details for: tmp-dev-clone
  Application name: tmp
  Region: us-west-2
  Deployed Version: app-141029_144740
  Environment ID: e-vjvrqnn5pv
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev-clone.elasticbeanstalk.com
  Updated: 2014-10-29 22:00:23.008000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-888214631909 as Amazon S3 storage bucket
 for environment data.
INFO: Created load balancer named: awseb-e-v-AWSEBLoa-4X0VL5UVQ353
INFO: Created security group named: awseb-e-vjvrqnn5pv-stack-AWSEBSecurityGroup-
18AV9FGCH2HZM
INFO: Created Auto Scaling launch configuration named: awseb-e-vjvrqnn5pv-stack-
AWSEBAutoScalingLaunchConfiguration-FDUWRSZZ6L3Z
INFO: Waiting for EC2 instances to launch. This may take a few minutes.
INFO: Created Auto Scaling group named: awseb-e-vjvrqnn5pv-stack-AWSEBAutoScal
ingGroup-69DN6PO5TISM
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:addb18d0-7088-402f-90ae-43be7c8d40cb:autoScalingGroup
Name/awseb-e-vjvrqnn5pv-stack-AWSEBAutoScalingGroup-69DN6PO5TISM:policy
Name/awseb-e-vjvrqnn5pv-stack-AWSEBAutoScalingScaleDownPolicy-I8GFGQ8T8MOV
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:fdcee817-e687-4fce-adc3-376995b3fef5:autoScalingGroup
Name/awseb-e-vjvrqnn5pv-stack-AWSEBAutoScalingGroup-69DN6PO5TISM:policy
Name/awseb-e-vjvrqnn5pv-stack-AWSEBAutoScalingScaleUpPolicy-1R312293DFY24
INFO: Created CloudWatch alarm named: awseb-e-vjvrqnn5pv-stack-AWSEBCloud
watchAlarmLow-1M67HXZH1U9K3
INFO: Created CloudWatch alarm named: awseb-e-vjvrqnn5pv-stack-AWSEBCloud
watchAlarmHigh-1K5CI7RVGV8ZJ
INFO: Added EC2 instance 'i-cf30e1c5' to Auto Scaling Group 'awseb-e-vjvrqnn5pv-
stack-AWSEBAutoScalingGroup-69DN6PO5TISM'.
INFO: Successfully launched environment: tmp-dev-clone
```

config

Description

Changes the environment configuration settings. This command saves the environment configuration settings as well as uploads, downloads, or lists saved configurations.

Syntax

eb config

eb config *environment_name*

The following describes the syntax for using the `eb config` command to work with saved configurations. For examples, see the see the Examples (p. 406) section later in this topic.

- **eb config delete *filename*** – Deletes the named saved configuration.
- **eb config get *filename*** – Downloads the named saved configuration.
- **eb config list** – Lists the saved configurations that you have in Amazon S3.
- **eb config put *filename*** – Uploads the named saved configuration to an Amazon S3 bucket. The **filename** must have the file extension `.cfg.yml`. To specify the file name without a path, you can save the file to the `.elasticbeanstalk` folder or to the `.elasticbeanstalk/saved_configs/` folder before you run the command. Alternatively, you can specify the **filename** by providing the full path.
- **eb config save** – Saves the environment configuration settings for the current running environment to `.elasticbeanstalk/saved_configs/` with the filename `[configuration-name].cfg.yml`. By default, EB CLI 3.1 saves the configuration settings with a *configuration-name* based on the environment name. You can specify a different configuration name by including the `--cfg` option with your desired configuration name when you run the command.

Options

Name	Description	Required
`--cfg`	The name to use for a saved configuration (which you can later specify to create or update an environment from a saved configuration).	No
`--timeout`	The number of minutes before the command times out.	No
Common options	For more information, see Common Options (p. 399).	No

Output

If the command runs successfully with no parameters, the command displays your current option settings in the text editor that you configured as the EDITOR environment variable. (If you have not configured an EDITOR environment variable, then EB CLI displays your option settings in your computer's default editor for YAML files.) When you save changes to the file and close the editor, the environment is updated with the option settings in the file.

If the command runs successfully with the `get` parameter, the command displays the location of the local copy that you downloaded.

If the command runs successfully with the `save` parameter, the command displays the location of the saved file.

Examples

This section describes how to change the text editor that you use to view and edit your option settings file.

For Linux/UNIX, the following example changes the editor to vim:

```
$ export EDITOR=vim
```

For Linux/UNIX,the following example changes the editor to what is installed at `/usr/bin/kate`.

```
$ export EDITOR=/usr/bin/kate
```

For Windows, the following example changes the editor to Notepad++.

```
> set EDITOR="C:\Program Files\Notepad++\Notepad++.exe
```

This section provides examples for the `eb config` command when it is run with parameters.

The following example deletes the saved configuration named app-tmp.

```
$ eb config delete app-tmp
```

The following example downloads the saved configuration with the name app-tmp from your Amazon S3 bucket.

```
$ eb config get app-tmp
```

The following example lists the names of saved configurations that are stored in your Amazon S3 bucket.

```
$ eb config list
```

The following example uploads the local copy of the saved configuration named app-tmp to your Amazon S3 bucket.

```
$ eb config put app-tmp
```

The following example saves configuration settings from the current running environment. If you do not provide a name to use for the saved configuration, then Elastic Beanstalk names the configuration file according to the environment name. For example, an environment named tmp-dev would be called `tmp-dev.cfg.yml`. Elastic Beanstalk saves the file to the folder `/.elasticbeanstalk/saved_configs/`.

```
$ eb config save
```

The following example shows how to use the `--cfg` option to save the configuration settings from the environment tmp-dev to a file called `v1-app-tmp.cfg.yml`. Elastic Beanstalk saves the file to the folder `/.elasticbeanstalk/saved_configs/`. If you do not specify an environment name, Elastic Beanstalk saves configuration settings from the current running environment.

```
$ eb config save tmp-dev --cfg v1-app-tmp
```

console

Description

Opens a browser to display the environment configuration dashboard in the Elastic Beanstalk Management Console.

Syntax

```
eb console
```

```
eb console environment_name
```

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

create

Description

Creates a new environment and deploys the current application or the sample application to the environment.

Syntax

```
eb create
```

```
eb create environment_name
```

> **Note**
> The *environment_name* is the environment in which you want to create or start the application. Environment names must be between 4 and 23 characters long and can only contain letters, numbers, and hyphens. Environment names can't begin or end with a hyphen. If you don't provide *environment_name* as a command line parameter, EB CLI prompts you for the environment name you want to use.
> If you include this parameter in the command line, then EB CLI does not prompt you to provide a CNAME. Instead, EB CLI autogenerates a CNAME.

Options

None of these options are required. If you run **eb create** without any options, you are prompted to enter or select a value for each setting.

Name	Description	Required

`-d`
or
`--branch_default`
	Sets the environment as the default environment.	No

`--cfg`	Creates an environment by using a saved configuration from either the `.elasticbeanstalk/saved_configs/` folder or your Amazon S3 bucket as a template for the new environment's configuration settings.	No

`-c` *`CNAME_prefix`*
or
`--cname` *`CNAME_prefix`*
	The prefix for the CNAME.
	Type: String
	Default: The environment name
		No

`-db`
or
`--database`
	Attaches a database to the environment. If you run `eb create` with the `--database` option, but without the `--database.username` and `--database.password` options, then EB CLI prompts you for the master database user name and password.	No

`-db.engine` *`engine`*
or
`--database.engine` *`engine`*
	The database engine type. If you run `eb create` with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option.
	Type: String
		No

Name	Description	Required
`-db.i` *instance_type* or `--database.instance` *instance_type*	The type of Amazon EC2 instance to use for the database. If you run `eb create` with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option. Type: String Valid values: See Option Values.	No
`-db.pass` *password* or `--database.password` *password*	The password for the database. If you run `eb create` with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option.	No
`-db.size` *number_of_gigabytes* or `--database.size` *number_of_gigabytes*	The number of gigabytes (GB) to allocate for database storage. If you run `eb create` with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option. Type: Number Valid values: • **MySQL** – **`5`** to **`1024`**. The default is **`5`**. • **Oracle** – **`10`** to **`1024`**. The default is **`10`**. • **Microsoft SQL Server Express Edition** – **`30`**. • **Microsoft SQL Server Web Edition** – **`30`**. • **Microsoft SQL Server Standard Edition** – **`200`**.	No
`-db.user` *username* or `--database.username` *username*	The user name for the database. If you run `eb create` with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option. If you run `eb create` with the `--database` option, but without the `--database.username` and `--database.password` options, then EB CLI prompts you for the master database user name and password.	No
`-db.version`*version* or `--database.version`*version*	Allows you to specify the database engine version. If this flag is present, the environment will launch with a database with the specified version number, even if the `--database` flag is not present.	No

Name	Description	Required
`--envvars`	Environment variables in a comma-separated list with the format ***`name`*****=*****`value`***. Type: String Constraints: • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / = + \ - @ • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either **`aws:`** or **`elasticbeanstalk:`**. • The combined size of all environment variables cannot exceed 4096 bytes.	No

`-ip` *`profile_name`*
or
`--instance_profile` *`profile_name`*
	The instance profile with the IAM role with the temporary security credentials that your application needs to access AWS resources.	No

`-i`
or
`--instance_type`
	The type of Amazon EC2 instance to use in the environment.
	Type: String
	Valid values: See Option Values.
		No

`-k` *`key_name`*
or
`--keyname` *`key_name`*
	The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application. If you include this option with the `eb create` command, the value you provide overwrites any key name that you might have specified with `eb init`.
	Type: String
	Valid values: An existing key name that is registered with Amazon EC2
		No

Name	Description	Required
`-p` *`platform`* (for example, **`php`**, **`PHP`**, **`php5.5`**, **`PHP 5.5`**, **`64bit Amazon Linux 2014.03 v1.0.7 running PHP 5.5`**) or `--platform` *`platform`*	The default platform (also known as the solution stack). If you specify this option with the `eb create` command, the value you provide overwrites any platform that you might have specified with `eb init`. If you do not specify a version, EB CLI uses the most recent container type. If you run `eb init` without this option, you are prompted to choose from a list of supported platforms. Names for container types for each supported platform change when AMIs are updated. Type: String Default: None Valid values: See Supported Platforms.	No

`-r` *`region`*
or
`--region` *`region`*
	The AWS region in which you want to deploy the application. If you include this option with the `eb create` command, the value you provide overwrites any region that you might have specified with `eb init`.
	For the list of values you can specify for this option, see AWS Elastic Beanstalk in the Regions and Endpoints topic in the *Amason Web Services General Reference*.
		No

`--sample`	Launches the Elastic Beanstalk sample application for the platform you select instead of using the application source code in the local project directory.	No
`--scale` *`scale`*	The number of instances to run when the environment launches.	No
`--service-role` *`servicerole`*	The service role to apply to the environment.	No
`--single`	Launches a single-instance environment. If not specified, EB CLI launches a load-balancing environment.	No
`--tags` *`name`*`=`*`value`*	Amazon EC2 tags for the environment, in a comma-separated list in the format ***`name`*`=`*`value`***. Type: String Constraints: • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / = + \ - @ • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either **`aws:`** or **`elasticbeanstalk:`**.	No

Name	Description	Required

`-t`
or
`--tier`
	Specifies the environment tier. If you don't specify the environment tier, EB CLI uses the most recent version of the **`webserver`** tier.
	Type: String
	Default: **`webserver`**
	Valid values: **`worker`** or **`webserver`**
		No

`--timeout`	The number of minutes before the command times out.	No
`--version` *`version_label`*	Specifies the application version that you want deployed to the environment instead of the application source code in the local project directory. Type: String Valid values: An existing application version label	No
`--vpc`	Configure a VPC for your environment. When you include this optino, the EB CLI prompts you to enter all required settings prior to launching the environment.	No
`--vpc.dbsubnets` *`subnet1,subnet2`*	Specifies subnets for database instances in a VPC.	Required when `--vpc.id` is specified
`--vpc.ec2subnets` *`subnet1,subnet2`*	Specifies subnets for Amazon EC2 instances in a VPC.	Required when `--vpc.id` is specified
`--vpc.elbpublic`	Launches your Elastic Load Balancing load balancer in a public subnet in your VPC.	No
`--vpc.elbsubnets` *`subnet1,subnet2`*	Specifies subnets for the Elastic Load Balancing load balancer in a VPC.	No
`--vpc.id` *`ID`*	Launches your environment in the specified VPC.	No, unless you want to launch your environment in a VPC
`--vpc.publicip`	Launches your Amazon EC2 instances in a public subnet in your VPC.	No
`--vpc.securitygroups` *`securitygroup1,securitygroup2`*	Specifies the security group ID or security group name.	No, unless you included the `--vpc.id` option with the command
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command prompts you with questions and then returns the status of the create operation. If there were problems during the launch, you can use the events (p. 417) operation to get more details.

Example 1

The following example creates an environment.

```
$ eb create
Enter Environment Name
(default is tmp-dev):
Enter DNS CNAME prefix
(default is tmp-dev):
Environment details for: tmp-dev
  Application name: tmp
  Region: us-west-2
  Deployed Version: app-141029_145448
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:54:51.063000+00:00
Printing Status:
INFO: createEnvironment is starting.
INFO: Using elasticbeanstalk-us-west-2-888214631909 as Amazon S3 storage bucket
 for environment data.
INFO: Created load balancer named: awseb-e-u-AWSEBLoa-AS5T4LHMG62Z
INFO: Created security group named: awseb-e-um3yfrzq22-stack-AWSEBSecurityGroup-
10MV688E4994W
INFO: Created Auto Scaling launch configuration named: awseb-e-um3yfrzq22-stack-
AWSEBAutoScalingLaunchConfiguration-LAYGQA7S0WLB
INFO: Waiting for EC2 instances to launch. This may take a few minutes.
INFO: Created Auto Scaling group named: awseb-e-um3yfrzq22-stack-AWSEBAutoScal
ingGroup-NAOJALR7EVNB
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:2642e591-3ce9-4b05-94c2-2defe59fe362:autoScalingGroup
Name/awseb-e-um3yfrzq22-stack-AWSEBAutoScalingGroup-NAOJALR7EVNB:policy
Name/awseb-e-um3yfrzq22-stack-AWSEBAutoScalingScaleDownPolicy-186ZHALUU40NL
INFO: Created Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:a2f0135c-a87d-47c8-b02f-f91e8ba01cc2:autoScalingGroup
Name/awseb-e-um3yfrzq22-stack-AWSEBAutoScalingGroup-NAOJALR7EVNB:policy
Name/awseb-e-um3yfrzq22-stack-AWSEBAutoScalingScaleUpPolicy-ALX8R1PE048
INFO: Created CloudWatch alarm named: awseb-e-um3yfrzq22-stack-AWSEBCloud
watchAlarmLow-F054729P40PL
INFO: Created CloudWatch alarm named: awseb-e-um3yfrzq22-stack-AWSEBCloud
watchAlarmHigh-VIH77T5YPDPP
INFO: Added EC2 instance 'i-4d133742' to Auto Scaling Group 'awseb-e-um3yfrzq22-
stack-AWSEBAutoScalingGroup-NAOJALR7EVNB'.
INFO: Adding instance 'i-4d133742' to your environment.
INFO: Successfully launched environment: tmp-dev
```

deploy

Description

Deploys the application source bundle from the initialized project directory to the running application.

If git is installed, EB CLI uses the `git archive` command to create a `.zip` file from the contents of the most recent `git commit` command.

> **Note**
> You can configure the EB CLI to deploy an artifact from your build process instead of creating a ZIP file of your project folder. See Deploying an Artifact Instead of the Project Folder (p. 392) for details.

Syntax

`eb deploy`

`eb deploy` ***`environment_name`***

Options

Name	Description	Required

`-l` *`version_label`*
or
`--label` *`version_label`*
	The version label for the application version that EB CLI deploys.
	Type: String
		No

`-m` "*`version_description`*" or `--message` "*`version_description`*"	The description for the application version, enclosed in double quotation marks. Type: String	No
`--staged`	Deploy files staged in the git index instead of the HEAD commit.	No
`--timeout` *`minutes`*	The number of minutes before the command times out.	No
`--version` *`version_label`*	An existing application version to deploy. Type: String	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command returns the status of the `deploy` operation.

Example

The following example deploys the current application.

```
$ eb deploy
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instances.
INFO: Environment update completed successfully.
```

events

Description

Returns the most recent events for the environment.

Syntax

eb events

eb events *environment_name*

Options

Name	Description	Required

`-f`
or
`--follow`
	Streams events. To cancel, press CTRL+C.	No

Output

If successful, the command returns recent events.

Example

The following example returns the most recent events.

```
$ eb events
2014-10-29 21:55:39     INFO    createEnvironment is starting.
2014-10-29 21:55:40     INFO    Using elasticbeanstalk-us-west-2-169465803350
as Amazon S3 storage bucket for environment data.
2014-10-29 21:55:57     INFO    Created load balancer named: awseb-e-r-AWSEBLoa-
NSKUOK5X6Z9J
2014-10-29 21:56:16     INFO    Created security group named: awseb-e-rxgrhjr9bx-
stack-AWSEBSecurityGroup-1UUHU5LZ20ZY7
2014-10-29 21:56:20     INFO    Created Auto Scaling launch configuration
named:awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingLaunchConfiguration-AG68JQHE9NWO
2014-10-29 21:57:18     INFO    Waiting for EC2 instances to launch. This may
take a few minutes.
2014-10-29 21:57:18     INFO    Created Auto Scaling group named: awseb-e-rx
grhjr9bx-stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
arn:aws:autoscaling:us-west-2:11122223333:scalingPolicy:2cced9e6-859b-421a-be63-
8ab34771155a:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-
1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingScaleUpPolicy-
1I2ZSNVU4APRY
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
arn:aws:autoscaling:us-west-2:11122223333:scalingPolicy:1f08b863-bf65-415a-b584-
b7fa3a69a0d5:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-
1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingScaleDown
Policy-1E3G7PZKZPSOG
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rx
grhjr9bx-stack-AWSEBCloudwatchAlarmLow-VF5EJ549FZBL
```

```
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rx
grhjr9bx-stack-AWSEBCloudwatchAlarmHigh-LA9YEW3O6WJO
2014-10-29 21:58:50     INFO    Added EC2 instance 'i-c7ee492d' to Auto Scal
ingGroup 'awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD'.
2014-10-29 21:58:53     INFO    Successfully launched environment: tmp-dev
2014-10-29 21:59:14     INFO    Environment health has been set to GREEN
2014-10-29 21:59:43     INFO    Adding instance 'i-c7ee492d' to your environment.
```

health

Description

Returns the most recent health for the environment.

Syntax

eb health

eb health ***environment_name***

Options

Name	Description	Required

`-r`
or
`--refresh`
	Show health information interactively and update every ten seconds as new information is reported.	No

`--mono`	Don't display color in output.	No

Output

If successful, the command returns recent health.

Example

The following example returns the most recent health.

```
$ eb health
 elasticBeanstalkExa-env                                  Ok
     2015-07-08 23:13:20
WebServer
         Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  id             status     cause
    Overall      Ok
  i-d581497d     Ok
  i-d481497c     Ok
  i-136e00c0     Ok
  i-126e00c1     Ok
  i-8b2cf575     Ok

  id             r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
     p50     p10
    Overall      0.0         -      -      -      -         -        -       -
       -       -
  i-d581497d     0.0         -      -      -      -         -        -       -
       -       -
  i-d481497c     0.0         -      -      -      -         -        -       -
       -       -
  i-136e00c0     0.0         -      -      -      -         -        -       -
       -       -
  i-126e00c1     0.0         -      -      -      -         -        -       -
       -       -
  i-8b2cf575     0.0         -      -      -      -         -        -       -
       -       -

  id             az            running     load 1  load 5      user%  nice%
system%  idle%   iowait%
  i-d581497d     us-east-1a    12 mins        0.0    0.03        0.2    0.0
   0.0   99.7       0.1
  i-d481497c     us-east-1a    12 mins        0.0    0.03        0.3    0.0
   0.0   99.7       0.0
  i-136e00c0     us-east-1b    12 mins        0.0    0.04        0.1    0.0
   0.0   99.9       0.0
  i-126e00c1     us-east-1b    12 mins       0.01    0.04        0.2    0.0
   0.0   99.7       0.1
  i-8b2cf575     us-east-1c    1 hour         0.0    0.01        0.2    0.0
   0.1   99.6       0.1
```

init

Description

Sets default values for Elastic Beanstalk applications created with EB CLI by prompting you with a series of questions.

> **Note**
> The values you set with `init` apply only to the current directory and repository. Until you run the `init` command, the current running environment is unchanged. Each time you run the `init` command, new settings get appended to the `config` file.

Syntax

`eb init -i`

`eb init -i` ***`application_name`***

> **Note**
> If you don't specify *`application_name`* as a command line parameter when you run `eb init`, EB CLI prompts you for the application name that you want to use.

Options

None of these options are required. If you run `eb init` without any options, you are prompted to enter or select a value for each setting.

Name	Description	Required
`-i` or `--interactive`	Forces EB CLI to prompt you to provide a value for every `eb init` command option. **Note** The `init` command prompts you to provide values for `eb init` command options that do not have a (default) value. After the first time you run the `eb init` command in a directory, EB CLI might not prompt you about any command options. Therefore, use the `--interactive` option when you want to change a setting that you previously set.	No

`-k` *`keyname`*
or
`--keyname` *`keyname`*
	The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.	No

Name	Description	Required
`-p` *`platform`* (for example, **`php`**, **`PHP`**, **`php5.5`**, **`"PHP 5.5"`**, **`node.js`**, **`"64bit Amazon Linux 2014.03 v1.0.7 running PHP 5.5"`**) or `--platform` *`platform`*	The default platform (also known as the solution stack). If you do not specify a version, EB CLI uses the most recent container type. If you run `eb init` without this option, you are prompted to choose from a list of supported platforms. Names for container types for each supported platform change when AMIs are updated. **Note** If you specify this option, then EB CLI does not prompt you for values for any other options. Instead, it assumes default values for each option. You can specify options for anything for which you do not want to use default values. Type: String Default: None	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command guides you through setting up a new AWS Elastic Beanstalk application through a series of prompts.

Example

The following example request initializes EB CLI and prompts you to enter information about your application. Replace the red placeholder text with your own values.

```
$ eb init -i
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-southeast-1 : Asia Pacific (Singapore)
7) ap-southeast-2 : Asia Pacific (Sydney)
8) ap-northeast-1 : Asia Pacific (Tokyo)
9) sa-east-1 : South America (Sao Paulo)
(default is 3): 3

Select an application to use
1) HelloWorldApp
2) NewApp
3) [ Create new Application ]
(default is 3): 3

Enter Application Name
(default is "tmp"):
Application tmp has been created.

It appears you are using PHP. Is this correct?
(y/n): y
```

```

Select a platform version.
1) PHP 5.5
2) PHP 5.4
3) PHP 5.3
(default is 1): 1
Do you want to set up SSH for your instances?
(y/n): y

Select a keypair.
1) aws-eb
2) [ Create new KeyPair ]
(default is 2): 1
```

list

Description

Lists all environments in the current application or all environments in all applications, as specified by the `--all` option.

Syntax

`eb list`

Options

Name	Description	Required

`-a`
or
`--all`
	Lists all environments from all applications.	No

`-v`
or
`--verbose`
	Provides more detailed information about all environments, including instances.	No

Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command returns a list of environment names in which your current environment is marked with an asterisk (*).

Example 1

The following example lists your environments and indicates that tmp-dev is your default environment.

```
$ eb list
* tmp-dev
```

Example 2

The following example lists your environments with additional details.

```
$ eb list --verbose
Region: us-west-2
Application: tmp
    Environments: 1
        * tmp-dev : ['i-c7ee492d']
```

local

Description

Use `eb local run` to run your application's containers locally in Docker. Check the application's container status with `eb local status`. Open the application in a web browser with `eb local open`. Retrieve the location of the application's logs with `eb local logs`.

`eb local setenv` and `eb local printenv` let you set and view environment variables that are provided to the Docker containers that you run locally with `eb local run`.

You must run all `eb local` commands in the project directory of a Docker application that has been initialized as an EB CLI repository by using `eb init`.

Syntax

`eb local run`

`eb local status`

`eb local open`

`eb local logs`

`eb local setenv`

`eb local printenv`

Options

`eb local run`

Name	Description	Required
`--envvars` *key1=value1,key2=value2*	Sets environment variables that the EB CLI will pass to the local Docker containers. In multicontainer environments, all variables are passed to all containers.	No
`--port` *hostport*	Maps a port on the host to the exposed port on the container. If you don't specify this option, the EB CLI uses the same port on both host and container. This option works only with single container applications.	No
Common options	For more information, see Common Options (p. 399).	No

`eb local status`

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

`eb local open`

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

`eb local logs`

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

`eb local setenv`

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

`eb local printenv`

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Output

`eb local run`

Status messages from Docker. Remains active as long as application is running. Press **Ctrl**+**C** to stop the application.

`eb local status`

The status of each container used by the application, running or not.

`eb local open`

Opens the application in a web browser and exits.

`eb local logs`

The location of the logs generated in your project directory by applications running locally under `eb local run`.

`eb local setenv`

None

`eb local printenv`

The name and values of environment variables set with `eb local setenv`.

Examples

`eb local run`

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

eb local status

View the status of your local containers:

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3
 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): None
Full local URL(s): None
```

eb local logs

View the log path for the current project:

```
~/project$ eb local logs
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbean
stalk/logs/local.
Logs were most recently created 3 minutes ago and written to /home/user/pro
ject/.elasticbeanstalk/logs/local/150420_234011665784.
```

eb local setenv

Set environment variables for use with `eb local run`.

```
~/project$ eb local setenv PARAM1=value
```

Print environment variables set with `eb local setenv`.

```
~/project$ eb local printenv
Environment Variables:
PARAM1=value
```

logs

Description

Returns logs for the specified or default environment. Relevant logs vary by container type.

Syntax

eb logs

eb logs *environment_name*

Options

Name	Description	Required

`-a`
or
`--all`
	Retrieves all logs and saves them to the `.elasticbeanstalk/logs` directory.	No

`--instance` *instance-id*	Retrieve logs for the specified instance only.	No
`--zip`	Retrieves all logs, compresses them into a `.zip` file, and then saves the file to the `.elasticbeanstalk/logs` directory.	No
Common options	For more information, see Common Options (p. 399).	No

Output

Shows the logs directly in the terminal by default (press q to close). `--all` and `--zip` options save the logs locally and output the location of the file(s).

Example

```
$ eb logs --zip
Retrieving logs...
Logs were saved to /home/workspace/environment/.elasticbean
stalk/logs/150622_173444.zip
```

open

Description

Opens the application in the default browser at the environment CNAME.

Syntax

eb open

eb open ***environment_name***

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Output

The command `eb open` does not have output. Instead, it opens the application in a browser window.

platform

Description

Lists supported platforms and enables you to set the default platform and platform version to use when you launch an environment. Use **`eb platform list`** to view a list of all supported platforms. Use **`eb platform select`** to change the platform for your project. Use **`eb platform show`** to view your project's selected platform.

Syntax

`eb platform list`

`eb platform select`

`eb platform show`

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Example 1

The following example provides the full name of containers for all platforms that Elastic Beanstalk currently supports.

```
$ eb platform list
docker-1.5.0
glassfish-4.0-java-7-(preconfigured-docker)
glassfish-4.1-java-8-(preconfigured-docker)
go-1.3-(preconfigured-docker)
go-1.4-(preconfigured-docker)
iis-7.5
iis-8
iis-8.5
multi-container-docker-1.3.3-(generic)
node.js
php-5.3
php-5.4
php-5.5
python
python-2.7
python-3.4
python-3.4-(preconfigured-docker)
ruby-1.9.3
ruby-2.0-(passenger-standalone)
ruby-2.0-(puma)
ruby-2.1-(passenger-standalone)
ruby-2.1-(puma)
ruby-2.2-(passenger-standalone)
ruby-2.2-(puma)
tomcat-6
tomcat-7
tomcat-7-java-6
```

```
tomcat-7-java-7
tomcat-8-java-8
```

Example 2

The following example prompts you to choose from a list of platforms and the version that you want to deploy for the specified platform.

```
$ eb platform select
Select a platform.
1) PHP
2) Node.js
3) IIS
4) Tomcat
5) Python
6) Ruby
7) Docker
8) Multi-container Docker
9) GlassFish
10) Go
(default is 1): 5

Select a platform version.
1) Python 2.7
2) Python
3) Python 3.4 (Preconfigured - Docker)
```

Example 3

The following example lists your current default platform and the default platform that Elastic Beanstalk will launch by default for environments that you create later.

```
$ eb platform show
Current default platform: Python 2.7
New environments will be running:  64bit Amazon Linux 2014.09 v1.2.0 running
Python 2.7

Platform info for environment "tmp-dev":
Current: 64bit Amazon Linux 2014.09 v1.2.0 running Python
Latest:  64bit Amazon Linux 2014.09 v1.2.0 running Python
```

printenv

Description

Prints all the environment variables in the command window.

Syntax

```
eb printenv
```

```
eb printenv environment_name
```

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command returns the status of the `printenv` operation.

Example

The following example prints environment variables for the specified environment.

```
$ eb printenv
Environment Variables:
     PARAM1 = None
     PARAM4 = None
     PARAM2 = None
     PARAM5 = None
     AWS_ACCESS_KEY_ID = None
     ExampleVar = ExampleValue
     AWS_SECRET_KEY = None
     PARAM3 = None
```

scale

Description

Scales the environment to always run on a specified number of instances, setting both the minimum and maximum number of instances to the specified number.

Syntax

```
eb scale scale
```

```
eb scale scale environment_name
```

Options

Name	Description	Required
--timeout	The number of minutes before the command times out.	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command updates the number of minimum and maximum instances to run to the specified number.

Example

The following example sets the number of instances to 2.

```
$ eb scale 2
INFO: Environment update is starting.
INFO: Updating environment tmp-dev's configuration settings.
INFO: Added EC2 instance 'i-5fce3d53' to Auto Scaling Group 'awseb-e-2cpfjbra9a-
stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E'.
INFO: Successfully deployed new configuration to environment.
INFO: Environment update completed successfully.
```

setenv

Description

Sets environment variables for the default environment.

Syntax

```
eb setenv key=value
```

You can include as many variables as you want, but the total size of all variables cannot exceed 4096 bytes. You can delete a variable by leaving the value blank.

Options

Name	Description	Required
--timeout	The number of minutes before the command times out.	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command displays that the environment update succeeded.

Example

The following example sets the environment variable ExampleVar.

```
$ eb setenv ExampleVar=ExampleValue
INFO: Environment update is starting.
INFO: Updating environment tmp-dev's configuration settings.
INFO: Successfully deployed new configuration to environment.
INFO: Environment update completed successfully.
```

The following command sets multiple environment variables. It adds the environment variable named `foo` and sets its value to `bar`, changes the value of the `JDBC_CONNECTION_STRING` variable, and deletes the `PARAM4` and `PARAM5` variables.

```
eb setenv foo=bar JDBC_CONNECTION_STRING=hello PARAM4= PARAM5=
```

ssh

Description

Note

This command does not work with environments running Windows Server instances.

Connect to a Linux Amazon EC2 instance in your environment using Secure Shell (SSH). If an environment has multiple running instances, EB CLI prompts you to specify which instance you want to connect to. To use this command, SSH must be installed on your local machine and available from the command line. Private key files must be located in a folder named `.ssh` under your user directory.

SSH Keys

If you have not previously configured SSH, you can use the EB CLI to create a key when running eb init. If you have already run `eb init`, run it again with the `--interactive` option and select **Yes** and **Create New Keypair** when prompted to set up SSH. Keys created during this process will be stored in the proper folder by the EB CLI.

This command temporarily opens port 22 in your environment's security group for incoming traffic from 0.0.0.0/0 (all IP addresses) if no rules for port 22 are already in place. If you have configured your environment's security group to open port 22 to a restricted CIDR range for increased security, the EB CLI will respect that setting and forgo any changes to the security group. To override this behavior and force the EB CLI to open port 22 to all incoming traffic, use the `--force` option.

See Amazon EC2 Security Groups (p. 201) for information on configuring your environment's security group.

Syntax

```
eb ssh
```

```
eb ssh environment_name
```

Options

Name	Description	Required

`-n`
or
`--number`
	Specifies which instance (that you choose from a list of all instances) that you want to use SSH to connect to.	No

`-i`
or
`--instance`
	Specifies the instance ID of the instance to which you connect. We recommend that you use this option.	No

`-o`
or
`--keep_open`
	Specifies that EB CLI should not close port 22 after the SSH session ends.	No

Name	Description	Required
`--setup`	Sets up the specified environment to use SSH. (This is an alternative to the `eb init` command that sets up SSH for all future environments.) **Note** Because Amazon EC2 keys cannot be changed, if you run this command, your environment will be restarted with all new instances. Your environment will be unavailable while this command is being executed.	No
`--force`	Open port 22 to incoming traffic from 0.0.0.0/0 in the environment's security group, even if the security group is already configured for SSH. Use this option if your environment's security group is configured to open port 22 to a restricted CIDR range that does not include the IP address that you are trying to connect from.	No
Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command opens an SSH connection to the instance.

Example

The following example connects you to the specified environment.

```
$ eb ssh
Select an instance to ssh into
1) i-96133799
2) i-5931e053
(default is 1): 1
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '54.191.45.125 (54.191.45.125)' can't be established.
RSA key fingerprint is ee:69:62:df:90:f7:63:af:52:7c:80:60:1b:3b:51:a9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.191.45.125' (RSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-8-185 ~]$ ls
[ec2-user@ip-172-31-8-185 ~]$ exit
logout
Connection to 54.191.45.125 closed.
INFO: Closed port 22 on ec2 instance security group
```

status

Description

Provides information about the status of the environment.

Syntax

eb status

eb status *environment_name*

Options

Name	Description	Required

`-v`
or
`--verbose`
	Provides more information about individual instances, such as their status with the Elastic Load Balancing load balancer.	No

Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command returns the following information about the environment:

- Environment name
- Application name
- Deployed application version
- Environment ID
- Platform
- Environment tier
- CNAME
- Time the environment was last updated
- Status
- Health

If you use verbose mode, EB CLI also provides you with the number of running Amazon EC2 instances.

Example

The following example shows the status for the environment tmp-dev.

```
$ eb status
Environment details for: tmp-dev
  Application name: tmp
  Region: us-west-2
  Deployed Version: None
  Environment ID: e-2cpfjbra9a
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
```

```
CNAME: tmp-dev.elasticbeanstalk.com
Updated: 2014-10-29 21:37:19.050000+00:00
Status: Launching
Health: Grey
```

swap

Description

Swaps the environment's CNAME with the CNAME of another environment (for example, to avoid downtime when you update your application version).

> **Note**
> If you have more than two environments, you are prompted to select the name of the environment that is currently using your desired CNAME from a list of environments. To suppress this, you can specify the name of the environment to use by including the `-n` option when you run the command.

Syntax

`eb swap`

`eb swap` ***`environment_name`***

> **Note**
> The *`environment_name`* is the environment for which you want a different CNAME. If you don't specify *`environment_name`* as a command line parameter when you run `eb swap`, EB CLI updates the CNAME of the default environment.

Options

Name	Description	Required

`-n`
or
`--destination_name`
	Specifies the name of the environment with which you want to swap CNAMEs. If you run `eb swap` without this option, then EB CLI prompts you to choose from a list of your environments.	No

Common options	For more information, see Common Options (p. 399).	No

Output

If successful, the command returns the status of the `swap` operation.

Examples

The following example swaps the environment tmp-dev with live-env.

```
$ eb swap
Select an environment to swap with.
1) staging-dev
2) live-env
(default is 1): 2
INFO: swapEnvironmentCNAMEs is starting.
INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-AWSEBLoa-
M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
INFO: Completed swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
```

The following example swaps the environment tmp-dev with the environment live-env but does not prompt you to enter or select a value for any settings.

```
$ eb swap tmp-dev --destination_name live-env
INFO: swapEnvironmentCNAMEs is starting.
INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-AWSEBLoa-
M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
INFO: Completed swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
```

terminate

Description

Terminates the running environment so that you do not incur charges for unused AWS resources.

> **Note**
> You can always launch a new environment using the same version later. If you have data from an environment that you would like to preserve, create a snapshot of your current database instance before you terminate the environment. You can later use it as the basis for new DB instance when you create a new environment. For more information, see Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Syntax

`eb terminate`

`eb terminate` ***`environment_name`***

Options

Name	Description	Required
`--all`	Terminates the environment, application, and all resources.	No
`--force`	Proceeds with termination without requiring you to confirm the environment name and that you want to proceed.	No
`--timeout`	The number of minutes before the command times out.	No

Output

If successful, the command returns the status of the `terminate` operation.

Example

The following example request terminates the environment tmp-dev.

```
$ eb terminate
The environment "tmp-dev" and all associated instances will be terminated.
To confirm, type the environment name: tmp-dev
INFO: terminateEnvironment is starting.
INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-AWSEBCloud
watchAlarmHigh-16V08YOF2KQ7U
INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-AWSEBCloud
watchAlarmLow-6ZAWH9F20P7C
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:5d7d3e6b-d59b-47c5-b102-3e11fe3047be:autoScalingGroup
Name/awseb-e-2cpfjbra9a-stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policy
Name/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleUpPolicy-1876U27JEC34J
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-west-
2:11122223333:scalingPolicy:29c6e7c7-7ac8-46fc-91f5-cfabb65b985b:autoScalingGroup
Name/awseb-e-2cpfjbra9a-stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policy
Name/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleDownPolicy-SL4LHODMOMU
```

```
INFO: Waiting for EC2 instances to terminate. This may take a few minutes.
INFO: Deleted Auto Scaling group named: awseb-e-2cpfjbra9a-stack-AWSEBAutoScal
ingGroup-7AXY7U13ZQ6E
INFO: Deleted Auto Scaling launch configuration named: awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingLaunchConfiguration-19UFHYGYWORZ
INFO: Deleted security group named: awseb-e-2cpfjbra9a-stack-AWSEBSecurityGroup-
XT4YYGFL7I99
INFO: Deleted load balancer named: awseb-e-2-AWSEBLoa-AK6RRYFQVV3S
INFO: Deleting SNS topic for environment tmp-dev.
INFO: terminateEnvironment completed successfully.
```

upgrade

Description

Upgrades the platform of your environment to the most recent version of the solution stack it is currently running.

Syntax

eb upgrade

eb upgrade *environment_name*

Options

Name	Description	Required
`--force`	Upgrades without requiring you to confirm the environment name before starting the upgrade process.	No
`--noroll`	Updates all instances without using rolling updates to keep some instances in service during the upgrade.	No
Common options	For more information, see Common Options (p. 399).	No

Output

The command shows an overview of the change and prompts you to confirm the upgrade by typing the environment name. If successful, your environment is updated and then launched with the most recent version of the platform.

Example

The following example upgrades the current platform version of the specified environment to the most recently available platform version.

```
$ eb upgrade
Current platform: 64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7
Latest platform:  64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

WARNING: This operation replaces your instances with minimal or zero downtime.
 You may cancel the upgrade after it has started by typing "eb abort".
You can also change your platform version by typing "eb clone" and then "eb
swap".

To continue, type the environment name:
```

use

Description

Sets the specified environment as the default environment.

When using Git, `eb use` sets the default environment for the current branch. Run this command once in each branch that you want to deploy to Elastic Beanstalk.

Syntax

```
eb use environment_name
```

Options

Name	Description	Required
Common options	For more information, see Common Options (p. 399).	No

Eb CLI 2.6 (deprecated)

Important

> This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

This section describes how to set up eb 2.6 and how to create a sample application using eb. This section also includes a command reference for eb 2.6.

Topics

Differences from version 3 of the EB CLI

EB and Eb are command line interface (CLI) tools for Elastic Beanstalk that you can use to deploy applications quickly and more easily. The most recent tool that Elastic Beanstalk supports is EB CLI 3. Elastic Beanstalk also supports eb 2.6 for customers who previously installed and continue to use it. You can use EB CLI 3 to manage environments that you launched using eb 2.6 or earlier versions of eb. EB CLI will automatically retrieve settings from an environment created using eb if the environment is running. Unlike eb, EB CLI does not store option settings locally.

EB CLI introduces the commands `eb create`, `eb deploy`, `eb open`, `eb console`, `eb scale`, `eb setenv`, `eb config`, `eb terminate`, `eb clone`, `eb list`, `eb use`, `eb printenv`, and `eb ssh`. In EB CLI 3.1 or later, you can also use the `eb swap` command. In EB CLI 3.2 only, you can use the `eb abort`, `eb platform`, and `eb upgrade` commands. In addition to these new commands, EB CLI 3 commands differ from eb 2.6 commands in several cases:

- **`eb init`** – You use `eb init` to create an `.elasticbeanstalk` directory in an existing project directory and create a new Elastic Beanstalk application for the project. Unlike with eb, running `eb init` with EB CLI does not prompt you to create an environment.
- **`eb start`** – EB CLI does not include the command `eb start`. Instead, you use `eb create` to create an environment.
- **`eb stop`** – EB CLI does not include the command `eb stop`. Instead, you use `eb terminate` to completely terminate an environment and clean up.
- **`eb push`** and **`git aws.push`** – EB CLI does not include the commands `eb push` or `git aws.push`. The commands have been replaced with the command `eb deploy`.
- **`eb update`** – EB CLI does not include the command `eb update`. You use the command `eb config` to update an environment.
- **`eb branch`** – EB CLI does not include the command `eb branch`.

For more information about using EB CLI 3 commands to create and manage an application, go to EB CLI Command Reference (p. 399). For a command reference for eb 2.6, see Eb Operations (p. 453). For a walkthrough of how to deploy a sample application using EB CLI 3, see Using the EB CLI (p. 394). For a walkthrough of how to deploy a sample application using eb 2.6, see Getting Started with Eb (p. 445). For a walkthrough of how to use eb 2.6 to map a Git branch to a specific environment, see Deploying a Git Branch to a Specific Environment (p. 451).

Getting Started with Eb

Important

> This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Eb is a command line interface (CLI) tool that asks you a series of questions and uses your answers to deploy and manage Elastic Beanstalk applications. This section provides an end-to-end walkthrough using eb to launch a sample application, view it, update it, and then delete it.

To complete this walkthrough, you will need to download the command line tools at the AWS Sample Code & Libraries website. For a complete CLI reference for more advanced scenarios, see Operations (p. 475), and see Getting Set Up (p. 472) for instructions on how to get set up.

Step 1: Initialize Your Git Repository

Eb is a command line interface that you can use with Git to deploy applications quickly and more easily. Eb is available as part of the Elastic Beanstalk command line tools package. Follow the steps below to install eb and initialize your Git repository.

To install eb, its prerequisite software, and initialize your Git repository

1. Install the following software onto your local computer:

   a. Linux/Unix/Mac

      - Download and unzip the Elastic Beanstalk command line tools package at the AWS Sample Code & Libraries website.
      - Git 1.6.6 or later. To download Git, go to http://git-scm.com/.
      - Python 2.7 or 3.0.

   b. Windows

- Download and unzip the Elastic Beanstalk command line tools package at the AWS Sample Code & Libraries website.
- Git 1.6.6 or later. To download Git, go to http://git-scm.com/.
- PowerShell 2.0.

  Note
  Windows 7 and Windows Server 2008 R2 come with PowerShell 2.0. If you are running an earlier version of Windows, you can download PowerShell 2.0. Visit http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx for more details.

2. Initialize your Git repository.

```
git init .
```

Step 2: Configure Elastic Beanstalk

Elastic Beanstalk needs the following information to deploy an application:

- AWS access key ID
- AWS secret key
- Service region
- Application name
- Environment name
- Solution stack

When you use the `init` command, Elastic Beanstalk will prompt you to enter this information. If a default value or current setting is available, and you want to use it, press `Enter`.

Before you use eb, set your PATH to the location of eb. The following table shows an example for Linux/UNIX and Windows.

In Linux and UNIX	In Windows
`$ export PATH=$PATH:<path to unzipped eb CLI package>/eb/linux/python2.7/` If you are using Python 3.0, the path will include `python3` rather than `python2.7`.	`C:\> set PATH=%PATH%;<path to unzipped eb CLI package>\eb\windows\`

To configure Elastic Beanstalk

Note
The EB CLI stores your credentials in a file named `credentials` in a folder named `.aws` in your user directory.

1. From the directory where you created your local repository, type the following command:

```
eb init
```

2. When you are prompted for the access key ID, type your access key ID. To get your access key ID, see How Do I Get Security Credentials? in the *AWS General Reference*.

   ```
   Enter your AWS Access Key ID (current value is "AKIAIOSFODNN7EXAMPLE"):
   ```

3. When you are prompted for the secret access key, type your secret access key. To get your secret access key, see How Do I Get Security Credentials? in the *AWS General Reference*.

   ```
   Enter your AWS Secret Access Key (current value is "wJalrXUtnFEMI/K7MDENG/bPxR
   fiCYEXAMPLEKEY"):
   ```

4. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
5. When you are prompted for the Elastic Beanstalk application name, type the name of the application. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use HelloWorld.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "windows"): HelloWorld
   ```

   Note

   If you have a space in your application name, make sure you do not use quotation marks.

6. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter an AWS Elastic Beanstalk environment name (current value is "HelloWorld-
   env"):
   ```

   Note

   If you have a space in your application name, make sure you do not have a space in your environment name.

7. When you are prompted, choose an environment tier. For more information about environment tiers, see Architectural Overview (p. 16). For this example, we'll use **1**.

   ```
   Available environment tiers are:
   1) WebServer::Standard::1.0
   2) Worker::SQS/HTTP::1.0
   ```

8. When you are prompted for the solution stack, type the number of the solution stack you want. For more information about solution stacks, see Supported Platforms (p. 24). For this example, we'll use **64bit Amazon Linux running PHP 5.4**.
9. When you are prompted, choose an environment type. For this example, we'll use **2**.

   ```
   Available environment types are:
   1) LoadBalanced
   2) SingleInstance
   ```

10. When you are prompted to create an Amazon RDS DB instance, type **y** or **n**. For more information about using Amazon RDS, see Using Elastic Beanstalk with Amazon RDS (p. 294). For this example, we'll type **y**.

    ```
    Create an Amazon RDS DB Instance? [y/n]:
    ```

11. When you are prompted to create the database from scratch or a snapshot, type your selection. For this example, we'll use **`No snapshot`**.
12. When you are prompted to enter your RDS user master password, type your password containing 8 to 16 printable ASCII characters (excluding /, \, and @).

    ```
    Enter an Amazon RDS DB master password:
    ```

    ```
    Retype password to confirm:
    ```

13. When you are prompted to create a snapshot if you delete the Amazon RDS DB instance, type **y** or **n**. For this example, we'll type **n**. If you type **n**, then your RDS DB instance will be deleted and your data will be lost if you terminate your environment.

    By default, eb sets the following default values for Amazon RDS.

    - **Database engine** — MySQL
    - **Default version:** — 5.5
    - **Database name:** — ebdb
    - **Allocated storage** — 5GB
    - **Instance class** — db.t1.micro
    - **Deletion policy** — delete
    - **Master username** — ebroot

14. When you are prompted to enter your instance profile name, you can choose to create a default instance profile or use an existing instance profile. Using an instance profile enables IAM users and AWS services to gain access to temporary security credentials to make AWS API calls. Using instance profiles prevents you from having to store long-term security credentials on the EC2 instance. For more information about instance profiles, see Service Roles, Instance Profiles, and User Policies (p. 19). For this example, we'll use **`Create a default instance profile`**.

    You should see a confirmation that your AWS Credential file was successfully updated.

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key. If you want to update your Amazon RDS DB configuration settings, you can update your `optionsettings` file in the `.elasticbeanstalk` directory, and then use the `eb update` command to update your Elastic Beanstalk environment.

> **Note**
> You can set up multiple directories for use with eb—each with its own Elastic Beanstalk configuration—by repeating the preceding two steps in each directory: first initialize a Git repository, and then use **init** to configure eb.

Step 3: Create the Application

Next, you need to create and deploy a sample application. For this step, you use a sample application that is already prepared. Elastic Beanstalk uses the configuration information you specified in the previous step to do the following:

- Create an application using the application name you specified.
- Launch an environment using the environment name you specified that provisions the AWS resources to host the application.
- Deploy the application into the newly created environment.

Use the `start` command to create and deploy a sample application.

To create the application

- From the directory where you created your local repository, type the following command:

```
eb start
```

It may take several minutes to complete this process. Elastic Beanstalk provides status updates during the process. If at any time you want to stop polling for status updates, press **Ctrl+C**. When the environment status is Green, Elastic Beanstalk outputs a URL for the application.

Step 4: View the Application

In the previous step, you created an application and deployed it to Elastic Beanstalk. After the environment is ready and its status is Green, Elastic Beanstalk provides a URL to view the application. In this step, you can check the status of the environment to make sure it is set to Green and then copy and paste the URL to view the application.

Use the `status` command to check the environment status, and then use the URL to view the application.

To view the application

1. From the directory where you created your local repository, type the following command:

   ```
   eb status --verbose
   ```

   Elastic Beanstalk displays the environment status. If the environment is set to Green, Elastic Beanstalk displays the URL for the application. If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB information is displayed.
2. Copy and paste the URL into your web browser to view your application.

Step 5: Update the Application

After you have deployed a sample application, you can update the sample application with your own application. In this step, we'll update the sample PHP application with a simple HelloWorld application.

To update the sample application

1. Create a simple PHP file that displays "Hello World" and name it `index.php`.

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Next, add your new program to your local Git repository, and then commit your change.

```
git add index.php
git commit -m "initial check-in"
```

> **Note**
> For information about Git commands, go to Git - Fast Version Control System.

2. Deploy to Elastic Beanstalk.

```
eb push
```

3. View your updated application. Copy and paste the same URL in your web browser as you did in Step 4: View the Application (p. 449).

Step 6: Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `stop` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

1. From the directory where you created your local repository, type the following command:

```
eb stop
```

This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

> **Note**
> If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

2. From the directory where you installed the command line interface, type the following command:

```
eb delete
```

Elastic Beanstalk displays a message once it has successfully deleted the application.

Deploying a Git Branch to a Specific Environment

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Developers often use branching in a project to manage code intended for different target environments. For example, you might have a test branch where you perform component or integration testing and a prod branch where you manage the code for your live or production code. With version 2.3 and later of the eb command line interface and AWS DevTools, you can use the `eb init` command to configure the `eb push` command to push your current git branch to a specific Elastic Beanstalk environment.

To set up a Git branch to deploy to a specific environment

1. Make sure you have version 2.3 of the Elastic Beanstalk command line tools installed.

   To check what version you have installed, use the following command:

   ```
   eb --version
   ```

   To download the command line tools, go to Elastic Beanstalk Command Line Tool page and follow the instructions in the README.txt file in the `.zip` file.
2. From a command prompt, change directories to the location of the local repository containing the code you want to deploy.

   If you have not set up a Git repository, you need to create one to continue. For information about how to use Git, see the Git documentation.
3. Make sure that the current branch for your local repository is the one you want to map to an Elastic Beanstalk environment.

   To switch to a branch, you use the `git checkout` command. For example, you would use the following command to switch to the prod branch.

   ```
   git checkout prod
   ```

   For more information about creating and managing branches in Git, see the Git documentation.
4. If you have not done so already, use the `eb init` command to configure eb to use Elastic Beanstalk with a specific settings for credentials, application, region, environment, and solution stack. The values set with `eb init` will be used as defaults for the environments that you create for your branches. For detailed instructions, see Step 2: Configure Elastic Beanstalk (p. 446).
5. Use the `eb branch` command to map the current branch to a specific environment.

   1. Type the following command.

      ```
      eb branch
      ```

   2. When prompted for an environment name, enter the name of the environment that you want to map to the current branch.

      The eb command will suggest a name in parentheses and you can accept that name by pressing the **Enter** key or type the name that you want.

```
The current branch is "myotherbranch".
    Enter an Elastic Beanstalk environment name (auto-generated value is
 "test-myotherbranch-en"):
```

You'll notice that eb displays the current branch in your Git repository so you know which branch you're working with. You can specify an existing environment or a new one. If you specify a new one, you'll need to create it with the `eb start` command.

3. When prompted about using the settings from the default environment, type **y** unless you explicitly don't want to use the `optionsettings` file from the default environment for the environment for this branch.

```
Do you want to copy the settings from the default environment "main-env"
 for the new branch? [y/n]: y
```

6. If you specified a new environment for your branch, use the `eb start` command to create and start the environment.

   When this command is successful, you're ready for the next step.
7. Use the `eb push` command to deploy the changes in the current branch to the environment that you mapped to the branch.

Eb Common Options

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

This section describes options common to all eb operations.

Name	Description
`-f`, `--force`	Skip the confirmation prompt.
`- h`, `--help`	Show the Help message. Type: String Default: None
`--verbose`	Display verbose information.
`--version`	Show the program's version number and exit.

Eb Operations

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

You can use the eb command line interface to perform a wide variety of operations.

Topics

Eb stores environment settings in the `.elasticbeanstalk/optionsettings` file for the repository. It is designed to read only from local files. When you run `eb start` or `eb update`, Elastic Beanstalk reads the `.elasticbeanstalk/optionsettings` file and provides its contents as parameters to the `CreateEnvironment` or `UpdateEnvironment` API actions.

You can use a configuration file in an `.ebextensions/*.conf` directory to configure some of the same settings that are in an `.elasticbeanstalk/optionsettings` file. However, the values for the settings in `.elasticbeanstalk/optionsettings` will take precedence over anything in `.ebextensions/*.conf` if the settings are configured in both. Additionally, any option setting that is specified using the API, including through eb, cannot later be changed in an environment using `.ebextensions` configuration files.

When you run `eb branch`, Elastic Beanstalk will either add a section to the values in `.elasticbeanstalk/optionsettings` or create a new one for a new environment. The command does not affect any running environments.

To view your current settings, run `eb status --verbose`. You might also want to use eb in conjunction with the Elastic Beanstalk console to get a complete picture of your applications and environments.

branch

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Maps a Git branch to a new or existing Elastic Beanstalk environment and configures the mapped environment through a series of prompts. You must first create the Git branch. If no branches exist in the Git repository, eb displays a message that prompts you to run the `branch` command. Eb then attempts to start the application specified in the default settings in the optionsettings file.

To map a Git branch, first run `git checkout` *<branch>*, specifying the name of the Git branch you want to map. Then run `eb branch`. If the branch has never been mapped to an Elastic Beanstalk environment, you'll have the option to copy the most current environment settings to the new environment.

Consider the following additional information about using `branch`:

- If you run `eb init` on an existing repository and change the application name, region, or solution stack, the command resets all existing branch mappings. Run `branch` again to map each branch to an environment.
- You can map different Git branches to the same Elastic Beanstalk environment but in most cases maintain one-to-one relationships between branches and environments.

For a tutorial that describes how to use eb to deploy a Git branch to Elastic Beanstalk, see Deploying a Git Branch to a Specific Environment (p. 451).

Syntax

`eb branch`

Options

Name	Description	Required
`-e` or `--environment name` *ENVIRONMENT_NAME*	The environment to which you want to map the current Git branch. If you do not use this option, you'll be prompted to accept the autogenerated environment name or enter a new one. Type: String Default: *<Git-branch-name>*-env	No
Common options	For more information, see Eb Common Options (p. 453).	No

Output

None

Example

The following example maps the Git branch `master` to a new environment called MyApp-env-test, using the same settings as a previously created environment called Myapp-env. Replace the red placeholder text with your own values.

```
PROMPT> eb branch

The current branch is "master".
Enter an AWS Elastic Beanstalk environment name (auto-generated value is "MyApp-
master-env"): MyApp-env-test
Do you want to copy the settings from environment "MyApp-env" for the new branch?
 [y/n]: y
PROMPT> eb status
Environment "MyApp-env-test" is not running.
```

delete

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Deletes the current application, or an application you specify, along with all associated environments, versions, and configurations. For a tutorial that includes a description of how to use `eb delete` to delete an application, see Getting Started with Eb (p. 445).

Note
The `delete` operation applies to an application and all of its environments. To stop only a single environment rather than an entire application, use `eb stop (p. 469)`.

Syntax

`eb delete`

Options

Name	Description	Required

`-a`
or
`--application-name` *`APPLICATION_NAME`*
	The application that you want to delete. If you do not use this option, eb will delete the application currently specified in `.elasticbeanstalk/optionsettings`. To verify your current settings, run `eb init` (the current values will be displayed; press Enter at each prompt to keep the current value).
	Type: String
	Default: *`Current setting`*
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns confirmation that the application was deleted.

Example

The following example request deletes the specified application and all of its environments. Replace the red placeholder text with your own values.

```
PROMPT> delete -a MyApp

Delete application? [y/n]: y
Deleted application "MyApp".
```

events

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Returns the most recent events for the environment.

Syntax

eb events

Options

Name	Description	Required
[*number*] *NUMBER*	The number of events to return. Valid values range from 1 to 1000. Type: Integer Default: 10	Yes

Output

If successful, the command returns the specified number of recent events.

Example

The following example returns the 15 most recent events.

```
PROMPT> eb events 15

2014-05-19 08:44:51     INFO     terminateEnvironment completed successfully.
2014-05-19 08:44:50     INFO     Deleting SNS topic for environment MyApp-test-
env.
2014-05-19 08:44:38     INFO     Deleted security group named: awseb-e-
fEXAMPLEre-stack-AWSEBSecurityGroup-1DEXAMPLEKI
2014-05-19 08:44:32     INFO     Deleted RDS database named: aa1k8EXAMPLEdxl
2014-05-19 08:38:33     INFO     Deleted EIP: xx.xx.xxx.xx
2014-05-19 08:37:04     INFO     Waiting for EC2 instances to terminate. This
may take a few minutes.
2014-05-19 08:36:45     INFO     terminateEnvironment is starting.
2014-05-19 08:27:54     INFO     Adding instance 'i-fEXAMPLE7' to your environ
ment.
2014-05-19 08:27:50     INFO     Successfully launched environment: MyApp-test-
env
2014-05-19 08:27:50     INFO     Application available at MyApp-test-en
vmEXAMPLEst.elasticbeanstalk.com.
2014-05-19 08:24:04     INFO     Waiting for EC2 instances to launch. This may
 take a few minutes.
2014-05-19 08:23:21     INFO     Created RDS database named: aa1k8EXAMPLEdxl
```

```
2014-05-19 08:17:07     INFO     Creating RDS database named: aa1k8EXAMPLEdxl.
 This may take a few minutes.
2014-05-19 08:16:58     INFO     Created security group named: awseb-e-
fEXAMPLEre-stack-AWSEBSecurityGroup-1D6HEXAMPLEKI
2014-05-19 08:16:54     INFO     Created EIP: 50.18.181.66
```

init

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Sets various default values for AWS Elastic Beanstalk environments created with eb, including your AWS credentials and region. The values you set with `init` apply only to the current directory and repository. You can override some defaults with operation options (for example, use `-e` or `--environment-name` to target `branch (p. 455)` to a specific environment).

Note

Until you run the `init` command, the current running environment is unchanged. Each time you run the `init` command, new settings get appended to the `config` file.

For a tutorial that shows you how to use `eb init` to deploy a sample application, see Getting Started with Eb (p. 445).

Syntax

`eb init`

Options

None of these options are required. If you run `eb init` without any options, you will be prompted to enter or select a value for each setting.

Name	Description	Required

`-a`
or
`--application-name` *`APPLICATION_NAME`*
	The application managed by the current repository.
	Type: String
	Default: None
		No

`--aws-credential-file` *`FILE_PATH_NAME`*	The file location where your AWS credentials are saved. (You can use the environment variable `AWS_CREDENTIAL_FILE` to set the file location.) Type: String Default: None	No

`-e`
or
`--environment-name` *`ENVIRONMENT_NAME`*
	The environment on which you want to perform operations.
	Type: String
	Default: *`<application-name>-env`*
		No

Name	Description	Required

`-I`
or
`--access-key-id` *`ACCESS_KEY_ID`*
	Your AWS access key ID.
	Type: String
	Default: None
		No

`-S`
or
`--secret-key` *`SECRET_ACCESS_KEY`*
	Your AWS secret access key.
	Type: String
	Default: None
		No

`-s`
or
`--solution-stack` *`SOLUTION_STACK_LABEL`*
	The solution stack used as the application container type. If you run `eb init` without this option, you will be prompted to choose from a list of supported solution stacks. Solution stack names change when AMIs are updated.
	Type: String
	Default: None
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command guides you through setting up a new AWS Elastic Beanstalk application through a series of prompts.

Example

The following example request initializes eb and prompts you to enter information about your application. Replace the red placeholder text with your own values.

```
PROMPT> eb init

C:\>eb init
For information about your AWS access key ID and secret access key,
  go to http://docs.aws.amazon.com/general/latest/gr/getting-aws-sec-creds.html.
Enter your AWS Access Key ID (current value is "AKIAI*****5ZB7Q"):
Enter your AWS Secret Access Key (current value is "DHSAi*****xKPo6"):
Select an AWS Elastic Beanstalk service region (current value is "US East (N.
Virginia)".
Available service regions are:
1) US East (Virginia)
2) US West (Oregon)
3) US West (North California)
4) EU West (Ireland)
5) Asia Pacific (Singapore)
6) Asia Pacific (Tokyo)
7) Asia Pacific (Sydney)
8) South America (Sao Paulo)
Select (1 to 8): 2
Enter an AWS Elastic Beanstalk application name (current value is "MyApp"):
```

```
MyApp
Enter an AWS Elastic Beanstalk environment name (current value is "MyApp-env"):
 MyApp-env
Select a solution stack (current value is "64bit Amazon Linux running Python").
Available solution stacks are:
1) 32bit Amazon Linux running PHP 5.4
2) 64bit Amazon Linux running PHP 5.4
3) 32bit Amazon Linux running PHP 5.3
4) 64bit Amazon Linux running PHP 5.3
5) 32bit Amazon Linux running Node.js
6) 64bit Amazon Linux running Node.js
7) 64bit Windows Server 2008 R2 running IIS 7.5
8) 64bit Windows Server 2012 running IIS 8
9) 32bit Amazon Linux running Tomcat 7
10) 64bit Amazon Linux running Tomcat 7
11) 32bit Amazon Linux running Tomcat 6
12) 64bit Amazon Linux running Tomcat 6
13) 32bit Amazon Linux running Python
14) 64bit Amazon Linux running Python
15) 32bit Amazon Linux running Ruby 1.8.7
16) 64bit Amazon Linux running Ruby 1.8.7
17) 32bit Amazon Linux running Ruby 1.9.3
18) 64bit Amazon Linux running Ruby 1.9.3
[...]
Select (1 to 70): 60
Select an environment type (current value is "LoadBalanced").
Available environment types are:
1) LoadBalanced
2) SingleInstance
Select (1 to 2): 1
Create an RDS DB Instance? [y/n] (current value is "Yes"): y
Create an RDS BD Instance from (current value is "[No snapshot]"):
1) [No snapshot]
2) [Other snapshot]
Select (1 to 2): 1
Enter an RDS DB master password (current value is "******"):
Retype password to confirm:
If you terminate your environment, your RDS DB Instance will be deleted and you
 will lose your data.
Create snapshot? [y/n] (current value is "Yes"): y
Attach an instance profile (current value is "aws-elasticbeanstalk-ec2-role"):
1) [Create a default instance profile]
2) AppServer-AppServerInstanceProfile-TK2exampleHP
3) AppServer-AppServerInstanceProfile-1G2exampleK8
4) aws-opsworks-ec2-role
5) aws-elasticbeanstalk-ec2-role
6) [Other instance profile]
Select (1 to 6): 5
Updated AWS Credential file at "C:\Users\YourName\.elasticbeanstalk\aws_creden
tial_file".
```

logs

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Returns logs for the environment. Relevant logs vary by container type.

Syntax

```
eb logs
```

Options

Name	Description	Required
Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns environment logs.

push

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Deploys the current application to the AWS Elastic Beanstalk environment from the Git repository.

Note

- The `eb push` operation does not push to your remote repository, if any. Use a standard `git push` or similar command to update your remote repository.
- The `-e` or `--environment-name` options are not not valid for `eb push`. To push to a different environment from the current one (based on either the `eb init` default settings or the Git branch that is currently checked out), run `eb branch` before running `eb push`.

Syntax

`eb push`

Options

Name	Description	Required
Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns the status of the `push` operation.

Example

The following example deploys the current application.

```
PROMPT> eb push
Pushing to environment: MyApp-env
remote:
To https://AKI
AXXXXXXXX5ZB7Q:2013092XXXXXXXXXXXXf502a780888b0a49899798aa6cbeaef690c0b
525d0f090c7338cbead589bf14f@git.elasticbeanstalk.us-west-2.amazonaws.com/v1/re
pos/417
0705XXXXXXXX23632303133/commitid/33626435366339626230646332656366376339 3EX
AMPLExxxxx5
3165643137343939EXAMPLExx036/environment/417070536570743236323031332d6d6173EX
AMPLEx65
2013-09-26 17:35:37     INFO    Adding instance 'i-5EXAMPLE' to your environment.
2013-09-26 17:36:12     INFO    Deploying new version to instance(s).
2013-09-26 17:36:20     INFO    New application version was deployed to running
 EC2 instances.
2013-09-26 17:36:20     INFO    Environment update completed successfully.
Update of environment "MyApp-env" has completed.
```

start

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Creates and deploys the current application into the specified environment. For a tutorial that includes a description of how to deploy a sample application using `eb start`, see Getting Started with Eb (p. 445).

Syntax

`eb start`

Options

Name	Description	Required

`-e`
or
`--environment-name` *`ENVIRONMENT_NAME`*
	The environment into which you want to create or start the current application.
	Type: String
	Default: Current setting
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns the status of the start operation. If there were issues during the launch, you can use the events (p. 458) operation to get more details.

Example 1

The following example starts the environment.

```
PROMPT> start

Starting application "MyApp".
Waiting for environment "MyApp-env" to launch.
2014-05-13 07:25:33 INFO createEnvironment is starting.
2014-05-13 07:25:39 INFO Using elasticbeanstalk-us-west-2-8EXAMPLE3 as Amazon
S3 storage bucket for environment data.
2014-05-13 07:26:07 INFO Created EIP: xx.xx.xxx.xx
2014-05-13 07:26:09 INFO Created security group named: awseb-e-vEXAMPLErp-stack-
AWSEBSecurityGroup-1GCEXAMPLEG0
2014-05-13 07:26:17 INFO Creating RDS database named: aavcEXAMPLE5y. This may
take a few minutes.
2014-05-13 07:32:36 INFO Created RDS database named: aavcEXAMPLE5y
2014-05-13 07:34:08 INFO Waiting for EC2 instances to launch. This may take a
few minutes.
2014-05-13 07:36:24 INFO Application available at MyApp-env-z4vsuuxh36.elastic
beanstalk.com.
```

```
2014-05-13 07:36:24 INFO Successfully launched environment: MyApp-env
Application is available at "MyApp-env-z4EXAMPLE6.elasticbeanstalk.com"
```

Example 2

The following example starts the current application into an environment called *MyApp-test-env*.

```
PROMPT> start -e MyApp-test-env

Starting application "MyApp".
Waiting for environment "MyApp-test-env" to launch.
2014-05-13 07:25:33 INFO createEnvironment is starting.
2014-05-13 07:25:39 INFO Using elasticbeanstalk-us-west-2-8EXAMPLE3 as Amazon
S3 storage bucket for environment data.
2014-05-13 07:26:07 INFO Created EIP: xx.xx.xxx.xx
2014-05-13 07:26:09 INFO Created security group named: awseb-e-vEXAMPLErp-stack-
AWSEBSecurityGroup-1GCEXAMPLEG0
2014-05-13 07:26:17 INFO Creating RDS database named: aavcEXAMPLE5y. This may
take a few minutes.
2014-05-13 07:32:36 INFO Created RDS database named: aavcEXAMPLE5y
2014-05-13 07:34:08 INFO Waiting for EC2 instances to launch. This may take a
few minutes.
2014-05-13 07:36:24 INFO Application available at MyApp-test-envz4vsuuxh36.
elasticbeanstalk.com.
2014-05-13 07:36:24 INFO Successfully launched environment: MyApp-test-env
Application is available at "MyApp-test-env-z4EXAMPLE6.elasticbeanstalk.com"
```

status

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Describes the status of the specified environment. For a tutorial that includes a description of how to view an environment's status using `eb status`, see Getting Started with Eb (p. 445).

Syntax

`eb status`

Options

You might want to use the `--verbose` option with `status`.

Name	Description	Required

`-e`
or
`--environment-name` *`ENVIRONMENT_NAME`*
	The environment for which you want to display status.
	Type: String
	Default: Current setting
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns the status of the environment.

Example 1

The following example request returns the status of the environment.

```
PROMPT> eb status --verbose
Retrieving status of environment "MyNodeApp-env".
URL      : MyNodeApp-env-tnEXAMPLEcf.elasticbeanstalk.com
Status   : Ready
Health   : Green
Environment Name: MyNodeApp-env
Environment ID : e-vmEXAMPLEp
Environment Tier: WebServer::Standard::1.0
Solution Stack : 64bit Amazon Linux 2014.02 running Node.js
Version Label : Sample Application
Date Created : 2014-05-14 07:25:35
Date Updated : 2014-05-14 07:36:24
Description :

RDS Database: AWSEBRDSDatabase | aavcEXAMPLEd5y.clak1.us-west-2.rds.amazon
aws.com:3306
```

```
Database Engine: mysql 5.5.33
Allocated Storage: 5
Instance Class: db.t1.micro
Multi AZ: False
Master Username: ebroot
Creation Time: 2014-05-15 07:29:39
DB Instance Status: available
```

Example 2

The following example request returns the status of an application named *MyNodeApp* in an environment called *MyNodeApp-test-env*.

```
PROMPT> eb status -e MyNodeApp-test-env -a MyNodeApp
Retrieving status of environment "MyNodeApp-test-env".
URL : MyNodeApp-test-env-tnEXAMPLEcf.elasticbeanstalk.com
Status  : Ready
Health  : Green

RDS Database: AWSEBRDSDatabase | aavcEXAMPLEd5y.clak1.us-west-2.rds.amazon
aws.com:3306
```

stop

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Terminates the environment. For a tutorial that includes a description of how to terminate an environment using `eb stop`, see Getting Started with Eb (p. 445).

Note
The `stop` operation applies to environments, not applications. To delete an application along with its environments, use `eb delete`.

Syntax

`eb stop`

Options

Name	Description	Required

`-e`
or
`--environment-name` *`ENVIRONMENT_NAME`*
	The environment you want to terminate. The environment must contain the current application; you cannot specify an application other than the one in the repository you're currently working in.
	Type: String
	Default: Current setting
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns the status of the `stop` operation.

Example1

The following example request terminates the environment.

```
PROMPT> eb stop

If you terminate your environment, your RDS DB Instance will be deleted and you
 will lose your data.
Terminate environment? [y/n]: y
Stopping environment "MyApp-env". This may take a few minutes.
2014-05-13 07:18:10 INFO terminateEnvironment is starting.
2014-05-13 07:18:17 INFO Waiting for EC2 instances to terminate. This may take
a few minutes.
2014-05-13 07:19:43 INFO Deleted EIP: xxx.xxx.xxx.xx
2014-05-13 07:19:43 INFO Deleted security group named: awseb-e-zEXAMPLEng-stack-
AWSEBSecurityGroup-MEEXAMPLENHQ
```

```
2014-05-13 07:19:51 INFO Deleting SNS topic for environment MyApp-env.
2014-05-13 07:19:52 INFO terminateEnvironment completed successfully.
Stop of environment "MyApp-env" has completed.
```

Example 2

The following example request terminates the environment named *MyApp-test-env*.

```
PROMPT> eb stop -e MyApp-test-env

If you terminate your environment, your RDS DB Instance will be deleted and you
 will lose your data.
Terminate environment? [y/n]: y
Stopping environment "MyApp-test-env". This may take a few minutes.
2014-05-15 17:27:09 INFO terminateEnvironment is starting.
2014-05-15 17:27:16 INFO Waiting for EC2 instances to terminate. This
may take a few minutes.
2014-05-15 17:27:42 INFO Deleted EIP: xxx.xxx.xxx.xx
2014-05-15 17:27:42 INFO Deleted security group named: awseb-e-zEXAMPLEngstack-
AWSEBSecurityGroup-MEEXAMPLENHQ
2014-05-15 17:34:50 INFO Deleting SNS topic for environment MyApp-testenv.
2014-05-15 17:34:51 INFO terminateEnvironment completed successfully.
2013-05-15 17:29:55     INFO    Deleted Auto Scaling group named: awseb-e-
mqmp6mmcpk-stack-AWSEBAutoScalingGroup-QALO012HZJVJ
2013-05-15 17:29:56    INFO   Deleted Auto Scaling launch configuration named:
 awseb-e-mqmp6mmcpk-stack-AWSEBAutoScalingLaunchConfiguration-1DBGPQ99YFX08
2013-05-15 17:34:11     INFO    Deleted RDS database named: aauel5gap2gqb4
2013-05-15 17:34:16    INFO   Deleted security group named: awseb-e-mqmp6mmcpk-
stack-AWSEBSecurityGroup-1LDYFT0256P0B
2013-05-15 17:34:17    INFO   Deleted load balancer named: awseb-e-m-AWSEBLoa-
CT74SPXN541T
2013-05-15 17:34:29     INFO    Deleting SNS topic for environment MyOtherApp-
env.
2013-05-15 17:34:30     INFO    terminateEnvironment completed successfully.
Stop of environment "MyApp-test-env" has completed.
```

update

Important

This version of the EB CLI and its documentation have been replaced with version 3. Version 3 has different commands and is not backwards compatible with version 2. For information on the new version, see EB Command Line Interface (p. 384).

Description

Updates the specified environment by reading the `.elasticbeanstalk/optionsettings`. (Setting values in `.elasticbeanstalk/optionsettings` take precedence over the values specified for the same settings specified in `.ebextensions/*.conf` if the settings are configured in both places.) Use this operation after making changes to your settings (for example, via `init` or `branch`).

Syntax

`eb update`

Options

Name	Description	Required

`-e`
or
`--environment-name` *`ENVIRONMENT_NAME`*
	The environment you want to update.
	Type: String
	Default: Current setting
		No

Common options	For more information, see Eb Common Options (p. 453).	No

Output

If successful, the command returns the status of the update operation.

Example

The following example request updates the environment.

```
PROMPT> eb update

Update environment? [y/n]: y
Updating environment "MyApp-env". This may take a few minutes.
2014-05-15 17:10:34 INFO Updating environment MyApp-env's configuration
settings.
2014-05-15 17:11:12 INFO Successfully deployed new configuration to en
vironment.
2014-05-15 17:11:12 INFO Environment update completed successfully.
Update of environment "MyApp-env" has completed.
```

Elastic Beanstalk API Command Line Interface (deprecated)

Important

> This tool, the Elastic Beanstalk API CLI, and its documentation have been replaced with the AWS CLI. See EB Command Line Interface (p. 384) for information on using the AWS CLI with Elastic Beanstalk.

Topics

You can use the command line to create and deploy applications to Elastic Beanstalk. This section contains a complete reference for the API command line interface. If you want a quick and easy way to deploy your applications without needing to know which command line operations to use, you can use eb or EB. Eb and EB are command line interface tools for Elastic Beanstalk that are interactive and ask you the necessary questions to deploy your application. For more information, see Using the EB CLI (p. 394). This section discusses how to set up the API command line interface and the common options. For a complete reference list, see Operations (p. 475).

Getting Set Up

Important

> This tool, the Elastic Beanstalk API CLI, and its documentation have been replaced with the AWS CLI. See EB Command Line Interface (p. 384) for information on using the AWS CLI with Elastic Beanstalk.

Elastic Beanstalk provides a command line interface (CLI) to access Elastic Beanstalk functionality without using the AWS Management Console or the APIs. This section describes the prerequisites for running the CLI tools (or command line tools), where to get the tools, how to set up the tools and their environment, and includes a series of common examples of tool usage.

Prerequisites

This document assumes you can work in a Linux/UNIX or Windows environment. The Elastic Beanstalk command line interface also works correctly on Mac OS X (which resembles the Linux and UNIX command environment), but no specific Mac OS X instructions are included in this guide.

As a convention, all command line text is prefixed with a generic **`PROMPT>`** command line prompt. The actual command line prompt on your machine is likely to be different. We also use **`$`** to indicate a Linux/UNIX-specific command and **`C:\>`** for a Windows-specific command. The example output resulting from the command is shown immediately thereafter without any prefix.

The command line tools used in this guide require Ruby (version 1.8.7+ or 1.9.2+) and Python version 2.7 to run. To view and download Ruby clients for a range of platforms, including Linux/UNIX and Windows, go to http://www.ruby-lang.org/en/. Python is available at python.org.

Note

If you are using Linux with a system version of Linux lower than 2.7, install Python 2.7 with your distribution's package manager and then modify the eb script under `eb/linux/python2.7/eb` to refer to the Python 2.7 executable:

```
#!/usr/bin/env python2.7
```

Additionally, you will need to install the boto module with pip:

```
$ sudo /usr/bin/easy_install-2.7 pip
$ sudo pip install boto
```

Getting the Command Line Tools

The command line tools are available as a .zip file on the AWS Sample Code & Libraries website. These tools are written in Ruby, and include shell scripts for Windows 2000, Windows XP, Windows Vista, Windows 7, Linux/UNIX, and Mac OS X. The .zip file is self-contained and no installation is required; simply download the .zip file and unzip it to a directory on your local machine. You can find the tools in the **api** directory.

Providing Credentials for the Command Line Interface

The command line interface requires the access key ID and secret access key. To get your access keys (access key ID and secret access key), see How Do I Get Security Credentials? in the *AWS General Reference*.

You need to create a file containing your access key ID and secret access key. The contents of the file should look like this:

```
AWSAccessKeyId=Write your AWS access ID
AWSSecretKey=Write your AWS secret key
```

Important

On UNIX, limit permissions to the owner of the credential file:

```
$ chmod 600 <the file created above>
```

With the credentials file set up, you'll need to set the AWS_CREDENTIAL_FILE environment variable so that the Elastic Beanstalk CLI tools can find your information.

To set the AWS_CREDENTIAL_FILE environment variable

- Set the environment variable using the following command:

On Linux and UNIX	On Windows
`$ export AWS_CREDENTIAL_FILE=<the file created above>`	`C:\> set AWS_CREDENTIAL_FILE=<the file created above>`

Set the Service Endpoint URL

By default, the AWS Elastic Beanstalk uses the US East (N. Virginia) region (us-east-1) with the elasticbeanstalk.us-east-1.amazonaws.com service endpoint URL. This section describes how to specify a different region by setting the service endpoint URL. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference.

To set the service endpoint URL

- Set the environment variable using the following command:

On Linux and UNIX	On Windows
`$ export ELASTICBEANSTALK_URL=<service_endpoint>`	`C:\> set ELASTICBEANSTALK_URL=<service_endpoint>`

For example, on Linux, type the following to set your endpoint to us-west-2:

```
export ELASTICBEANSTALK_URL="https://elasticbeanstalk.us-west-2.amazonaws.com"
```

For example, on Windows, type the following to set your endpoint to us-west-2:

```
set ELASTICBEANSTALK_URL=https://elasticbeanstalk.us-west-2.amazonaws.com
```

Common Options

Important

This tool, the Elastic Beanstalk API CLI, and its documentation have been replaced with the AWS CLI. See EB Command Line Interface (p. 384) for information on using the AWS CLI with Elastic Beanstalk.

The command line operations accept the set of optional parameters described in the following table.

Option	Description
`--help` `-h`	Displays help text for the command. You can also use `help` *`command-name`*. This option applies to eb and the original command line interface. Default: off
`--show-json` `-j`	Displays the raw JSON response. This option applies only to the original command line interface. Default: off

Operations

Important

This tool, the Elastic Beanstalk API CLI, and its documentation have been replaced with the AWS CLI. See EB Command Line Interface (p. 384) for information on using the AWS CLI with Elastic Beanstalk.

Topics

elastic-beanstalk-check-dns-availability

Description

Checks if the specified CNAME is available.

Syntax

```
elastic-beanstalk-check-dns-availability -c [CNAMEPrefix]
```

Options

Name	Description	Required

`-c`
`--cname-prefix` *`CNAMEPrefix`*
	The name of the CNAME to check.
	Type: String
	Default: None
		Yes

Output

The command returns a table with the following information:

- **Available—**Shows `true` if the CNAME is available; otherwise, shows `false`.
- **FullyQualifiedCNAME—**Shows the fully qualified CNAME if it is available; otherwise shows N/A.

Examples

Checking to Availability of a CNAME

This example shows how to check to see if the CNAME prefix "myapp23" is available.

```
PROMPT> elastic-beanstalk-check-dns-availability -c myapp23
```

elastic-beanstalk-create-application

Description

Creates an application that has one configuration template named `default` and no application versions.

> **Note**
> The `default` configuration template is for a 32-bit version of the Amazon Linux operating system running the Tomcat 6 application container.

Syntax

```
elastic-beanstalk-create-application -a [name] -d [desc]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application. Constraint: This name must be unique within your account. If the specified name already exists, the action returns an `InvalidParameterValue` error. Type: String Default: None	Yes
`-d` `--description` *desc*	The description of the application. Type: String Default: None	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application. If no application is found with this name, and `AutoCreateApplication` is `false`, Elastic Beanstalk returns an `InvalidParameterValue` error.
- **ConfigurationTemplates—**A list of the configuration templates used to create the application.
- **DateCreated—**The date the application was created.
- **DateUpdated—**The date the application was last updated.
- **Description—**The description of the application.
- **Versions—**The versions of the application.

Examples

Creating an Application

This example shows how to create an application.

```
PROMPT> elastic-beanstalk-create-application -a MySampleApp -d "My description"
```

elastic-beanstalk-create-application-version

Description

Creates an application version for the specified application.

> **Note**
> Once you create an application version with a specified Amazon S3 bucket and key location, you cannot change that Amazon S3 location. If you change the Amazon S3 location, you receive an exception when you attempt to launch an environment from the application version.

Syntax

```
elastic-beanstalk-create-application-version -a [name] -l [label] -c -d [desc]
-s [location]
```

Options

Name	Description	Required
-a --application-name *name*	The name of the application. If no application is found with this name, and `AutoCreateApplication` is `false`, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
-c --auto-create	Determines how the system behaves if the specified application for this version does not already exist: • `true`: Automatically creates the specified application for this release if it does not already exist. • `false`: Throws an `InvalidParameterValue` if the specified application for this release does not already exist. Type: Boolean Valid Values: `true` | `false` Default: `false`	No
-d --description *desc*	The description of the version. Type: String Default: None Length Constraints: Minimum value of 0. Maximum value of 200.	No

Name	Description	Required

`-l`
`--version-label` *`label`*
	A label identifying this version.
	Type: String
	Default: None
	Constraint: Must be unique per application. If an application version already exists with this label for the specified application, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Length Constraints: Minimum value of 1. Maximum value of 100.
		Yes

`-s`
`--source-location` *`location`*
	The name of the Amazon S3 bucket and key that identify the location of the source bundle for this version, in the format `bucketname/key`.
	If data found at the Amazon S3 location exceeds the maximum allowed source bundle size, Elastic Beanstalk returns an `InvalidParameterCombination` error.
	Type: String
	Default: If not specified, AWS Elastic Beanstalk uses a sample application. If only partially specified (for example, a bucket is provided but not the key) or if no data is found at the Amazon S3 location, AWS Elastic Beanstalk returns an `InvalidParameterCombination` error.
		No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application.
- **DateCreated—**The date the application was created.
- **DateUpdated—**The date the application was last updated.
- **Description—**The description of the application.
- **SourceBundle—**The location where the source bundle is located for this version.
- **VersionLabel—**A label uniquely identifying the version for the associated application.

Examples

Creating a Version from a Source Location

This example shows create a version from a source location.

```
PROMPT> elastic-beanstalk-create-application-version -a MySampleApp -d "My
version" -l "TestVersion 1" -s amazonaws.com/sample.war
```

elastic-beanstalk-create-configuration-template

Description

Creates a configuration template. Templates are associated with a specific application and are used to deploy different versions of the application with the same configuration settings.

Syntax

```
elastic-beanstalk-create-configuration-template -a [name] -t [name] -E [id] -d
[desc] -s [stack] -f [filename] -A [name] -T [name]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application to associate with this configuration template. If no application is found with this name, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None	Yes
`-t` `--template-name` *name*	The name of the configuration template. If a configuration template already exists with this name, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Constraint: Must be unique for this application. Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-E` `--environment-id` *id*	The environment ID of the configuration template. Type: String Default: None	No
`-d` `--description` *desc*	The description of the configuration. Type: String Default: None	No

Name	Description	Required

`-s`
`--solution-stack` *`stack`*
	The name of the solution stack used by this configuration. The solution stack specifies the operating system, architecture, and application server for a configuration template. It determines the set of configuration options as well as the possible and default values.
	Use `elastic-beanstalk-list-available-solution-stacks` to obtain a list of available solution stacks.
	A solution stack name or a source configuration parameter must be specified; otherwise, Elastic Beanstalk returns an `InvalidParameterValue` error.
	If a solution stack name is not specified and the source configuration parameter is specified, Elastic Beanstalk uses the same solution stack as the source configuration template.
	Type: String
	Length Constraints: Minimum value of 0. Maximum value of 100.
		No

`-f`
`--options-file` *`filename`*
	The name of a JSON file that contains a set of key-value pairs defining configuration options for the configuration template. The new values override the values obtained from the solution stack or the source configuration template.
	Type: String
		No

`-A`
`--source-application-name` *`name`*
	The name of the application to use as the source for this configuration template.
	Type: String
	Default: None
		No

`-T`
`--source-template-name` *`name`*
	The name of the template to use as the source for this configuration template.
	Type: String
	Default: None
		No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with the configuration set.
- **DateCreated—**The date (in UTC time) when this configuration set was created.
- **DateUpdated—**The date (in UTC time) when this configuration set was last modified.
- **DeploymentStatus—**If this configuration set is associated with an environment, the deployment status parameter indicates the deployment status of this configuration set:
  - `null`: This configuration is not associated with a running environment.
  - `pending`: This is a draft configuration that is not deployed to the associated environment but is in the process of deploying.

  - `deployed`: This is the configuration that is currently deployed to the associated running environment.
  - `failed`: This is a draft configuration that failed to successfully deploy.
- **Description—**The description of the configuration set.
- **EnvironmentName—**If not `null`, the name of the environment for this configuration set.
- **OptionSettings—**A list of configuration options and their values in this configuration set.
- **SolutionStackName—**The name of the solution stack this configuration set uses.
- **TemplateNamel—**If not `null`, the name of the configuration template for this configuration set.

Examples

Creating a Basic Configuration Template

This example shows how to create a basic configuration template. For a list of configuration settings, see Configuration Options (p. 103).

```
PROMPT> elastic-beanstalk-create-configuration-template -a MySampleApp -t mycon
figtemplate -E e-eup272zdrw
```

Related Operations

- elastic-beanstalk-describe-configuration-options (p. 496)
- elastic-beanstalk-describe-configuration-settings (p. 498)
- elastic-beanstalk-list-available-solution-stacks (p. 505)

elastic-beanstalk-create-environment

Description

Launches an environment for the specified application using the specified configuration.

Syntax

```
elastic-beanstalk-create-environment -a [name] -l [label] -e [name] [-t [name]
| -s [stack]] -c [prefix] -d [desc] -f[filename] -F [filename]
```

Options

Name	Description	Required

`-a`
`--application-name` *`name`*
	The name of the application that contains the version to be deployed. If no application is found with this name, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		Yes

`-l`
`--version-label` *`label`*
	The name of the application version to deploy.
	If the specified application has no associated application versions, Elastic Beanstalk `UpdateEnvironment` returns an `InvalidParameterValue` error.
	Default: If not specified, Elastic Beanstalk attempts to launch the the sample application in the container.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		No

`-e`
`--environment-name` *`name`*
	A unique name for the deployment environment. Used in the application URL.
	Constraint: Must be from 4 to 23 characters in length. The name can contain only letters, numbers, and hyphens. It cannot start or end with a hyphen. This name must be unique in your account. If the specified name already exists, Elastic Beanstalk returns an `InvalidParameterValue`.
	Type: String
	Default: If the CNAME parameter is not specified, the environment name becomes part of the CNAME, and therefore part of the visible URL for your application.
		Yes

Name	Description	Required

`-t`
`--template-name` *`name`*
	The name of the configuration template to use in the deployment. If no configuration template is found with this name, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Conditional: You must specify either this parameter or a solution stack name, but not both. If you specify both, Elastic Beanstalk returns an `InvalidParameterValue` error. If you do not specify either, Elastic Beanstalk returns a `MissingRequiredParameter`.
	Type: String
	Default: None
	Constraint: Must be unique for this application.
		Conditional

`-s`
`--solution-stack` *`stack`*
	This is the alternative to specifying a configuration name. If specified, Elastic Beanstalk sets the configuration values to the default values associated with the specified solution stack.
	Condition: You must specify either this or a `TemplateName`, but not both. If you specify both, Elastic Beanstalk returns an `InvalidParameterCombination` error. If you do not specify either, Elastic Beanstalk returns `MissingRequiredParameter` error.
	Type: String
	Default: None
		Conditional

`-c`
`--cname-prefix` *`prefix`*
	If specified, the environment attempts to use this value as the prefix for the CNAME. If not specified, the environment uses the environment name.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		No

`-d`
`--description` *`desc`*
	The description of the environment.
	Type: String
	Default: None
		No

`-f`
`--options-file` *`filename`*
	The name of a JSON file that contains a set of key-value pairs defining configuration options for this new environment. These override the values obtained from the solution stack or the configuration template.
	Type: String
		No

`-F`
`--options-to-remove-file` *`value`*
	The name of a JSON file that contains configuration options to remove from the configuration set for this new environment.
	Type: String
	Default: None
		No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this environment.
- **CNAME—**The URL to the CNAME for this environment.
- **DateCreated—**The date the environment was created.
- **DateUpdated—**The date the environment was last updated.
- **Description—**The description of the environment.
- **EndpointURL—**The URL to the LoadBalancer for this environment.
- **EnvironmentID—**The ID of this environment.
- **EnvironmentName—**The name of this environment.
- **Health—**Describes the health status of the environment. Elastic Beanstalk indicates the failure levels for a running environment:
  - `Red`: Indicates the environment is not responsive. Occurs when three or more consecutive failures occur for an environment.
  - `Yellow`: Indicates that something is wrong. Occurs when two consecutive failures occur for an environment.
  - `Green`: Indicates the environment is healthy and fully functional.
  - `Gray`: Default health for a new environment. The environment is not fully launched and health checks have not started or health checks are suspended during an `UpdateEnvironment` or `RestartEnvironment`request.
- **Resources—**A list of AWS resources used in this environment.
- **SolutionStackName—**The name of the solution stack deployed with this environment.
- **Status—**The current operational status of the environment:
  - `Launching`: Environment is in the process of initial deployment.
  - `Updating`: Environment is in the process of updating its configuration settings or application version.
  - `Ready`: Environment is available to have an action performed on it, such as update or terminate.
  - `Terminating`: Environment is in the shut-down process.
  - `Terminated`: Environment is not running.
- **TemplateName—**The name of the configuration template used to originally launch this environment.
- **VersionLabel—**The application version deployed in this environment.

Examples

Creating an Environment Using a Basic Configuration Template

This example shows how to create an environment using a basic configuration template as well as pass in a file to edit configuration settings and a file to remove configuration settings. For a list of configuration settings, see Configuration Options (p. 103).

```
$ elastic-beanstalk-create-environment -a MySampleApp -t myconfigtemplate -e
MySampleAppEnv -f options.txt -F options_remove.txt
```

options.txt

```
[
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MinSize",
        "Value": "2"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MaxSize",
        "Value": "3"
    }
]
```

options_remove.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:sns:topics",
        "OptionName": "PARAM4"
    }
]
```

elastic-beanstalk-create-storage-location

Description

Creates the Amazon S3 storage location for the account. This location is used to store user log files and is used by the AWS Management Console to upload application versions. You do not need to create this bucket in order to work with Elastic Beanstalk.

Syntax

`elastic-beanstalk-create-storage-location`

Examples

Creating the Storage Location

This example shows how to create a storage location.

```
PROMPT> elastic-beanstalk-create-storage-location
```

This command will output the name of the Amazon S3 bucket created.

elastic-beanstalk-delete-application

Description

Deletes the specified application along with all associated versions and configurations.

> **Note**
> You cannot delete an application that has a running environment.

Syntax

```
elastic-beanstalk-delete-application -a [name] -f
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application to delete. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-f` `--force-terminate-env`	Determines if all running environments should be deleted before deleting the application. Type: Boolean Valid Values: `true | false` Default: `false`	No

Output

The command returns the string `Application deleted.`

Examples

Deleting an Application

This example shows how to delete an application.

```
PROMPT> elastic-beanstalk-delete-application -a MySampleApp
```

elastic-beanstalk-delete-application-version

Description

Deletes the specified version from the specified application.

> **Note**
> You cannot delete an application version that is associated with a running environment.

Syntax

```
elastic-beanstalk-delete-application-version -a [name] -l [label] -d
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application to delete releases from. Type: String Default: None	Yes
`-l` `--version-label`	The label of the version to delete. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-d` `--delete-source-bundle`	Indicates whether to delete the associated source bundle from Amazon S3. `true`: An attempt is made to delete the associated Amazon S3 source bundle specified at time of creation. `false`: No action is taken on the Amazon S3 source bundle specified at time of creation. Type: Boolean Valid Values: `true` | `false` Default: `false`	No

Output

The command returns the string `Application version deleted.`

Examples

Deleting an Application Version

This example shows how to delete an application version.

```
PROMPT> elastic-beanstalk-delete-application-version -a MySampleApp -l MyAppVersion
```

Deleting an Application Version and Amazon S3 Source Bundle

This example shows how to delete an application version.

```
PROMPT> elastic-beanstalk-delete-application-version -a MySampleApp -l MyAppVersion -d
```

elastic-beanstalk-delete-configuration-template

Description

Deletes the specified configuration template.

> **Note**
> When you launch an environment using a configuration template, the environment gets a copy of the template. You can delete or modify the environment's copy of the template without affecting the running environment.

Syntax

```
elastic-beanstalk-delete-configuration-template -a [name] -t [name]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application to delete the configuration template from. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-t` `--template-name`	The name of the configuration template to delete. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes

Output

The command returns the string `Configuration template deleted.`

Examples

Deleting a Configuration Template

This example shows how to delete a configuration template.

```
PROMPT> elastic-beanstalk-delete-configuration-template -a MySampleApp -t MyCon
figTemplate
```

elastic-beanstalk-delete-environment-configuration

Description

Deletes the draft configuration associated with the running environment.

> **Note**
> Updating a running environment with any configuration changes creates a draft configuration set. You can get the draft configuration using `elastic-beanstalk-describe-configuration-settings` while the update is in progress or if the update fails. The deployment status for the draft configuration indicates whether the deployment is in process or has failed. The draft configuration remains in existence until it is deleted with this action.

Syntax

`elastic-beanstalk-delete-environment-configuration -a [`*`name`*`] -e [`*`name`*`]`

Options

Name	Description	Required

`-a`
`--application-name` *`name`*
	The name of the application the environment is associated with.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		Yes

`-e`
`--environment-name` *`name`*
	The name of the environment to delete the draft configuration from.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		Yes

Output

The command returns the string `Environment configuration deleted.`

Examples

Deleting a Configuration Template

This example shows how to delete a configuration template.

```
PROMPT> elastic-beanstalk-delete-environment-configuration -a MySampleApp -e
MyEnvConfig
```

elastic-beanstalk-describe-application-versions

Description

Returns information about existing application versions.

Syntax

```
elastic-beanstalk-describe-application-versions -a [name] -l [labels [,label..]]
```

Options

Name	Description	Required
-a --application-name *value*	The name of the application. If specified, Elastic Beanstalk restricts the returned descriptions to only include ones that are associated with the specified application. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
-l --version-label *labels*	Comma-delimited list of version labels. If specified, restricts the returned descriptions to only include ones that have the specified version labels. Type: String[] Default: None	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this release.
- **DateCreated—**The date the application was created.
- **DateUpdated—**The date the application version was last updated.
- **Description—**The description of the application version.
- **SourceBundle—**The location where the source bundle is located for this version.
- **VersionLabel—**A label uniquely identifying the version for the associated application.

Examples

Describing Application Versions

This example shows how to describe all application versions for this account.

```
PROMPT> elastic-beanstalk-describe-application-versions
```

Describing Application Versions for a Specified Application

This example shows how to describe application versions for a specific application.

```
PROMPT> elastic-beanstalk-describe-application-versions -a MyApplication
```

Describing Multiple Application Versions

This example shows how to describe multiple specified application versions.

```
PROMPT> elastic-beanstalk-describe-application-versions -l MyAppVersion1,
MyAppVersion2
```

elastic-beanstalk-describe-applications

Description

Returns descriptions about existing applications.

Syntax

```
elastic-beanstalk-describe-applications -a [names [,name..]]
```

Options

Name	Description	Required
-a --application-names *name*	The name of one or more applications, separated by commas. If specified, Elastic Beanstalk restricts the returned descriptions to only include those with the specified names. Type: String[] Default: None	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application.
- **ConfigurationTemplates—**A list of the configuration templates used to create the application.
- **DateCreated—**The date the application was created.
- **DateUpdated—**The date the application was last updated.
- **Description—**The description of the application.
- **Versions—**The names of the versions for this application.

Examples

Describing the Applications

This example shows how to describe all applications for this account.

```
PROMPT> elastic-beanstalk-describe-applications
```

Describing a Specific Application

This example shows how to describe a specific application.

```
PROMPT> elastic-beanstalk-describe-applications -a MyApplication
```

elastic-beanstalk-describe-configuration-options

Description

Describes the configuration options that are used in a particular configuration template or environment, or that a specified solution stack defines. The description includes the values, the options, their default values, and an indication of the required action on a running environment if an option value is changed.

Syntax

```
elastic-beanstalk-describe-configuration-options -a [name] -t [name] -e [name]
-s [stack] -f [filename]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application associated with the configuration template or environment. Only needed if you want to describe the configuration options associated with either the configuration template or environment. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-t` `--template-name` *name*	The name of the configuration template whose configuration options you want to describe. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-e` `--environment-name` *name*	The name of the environment whose configuration options you want to describe. Type: String Length Constraints: Minimum value of 4. Maximum value of 23.	No
`-s` `--solution-stack` *stack*	The name of the solution stack whose configuration options you want to describe. Type: String Default: None Length Constraints: Minimum value of 0. Maximum value of 100.	No
`-f` `--options-file` *filename*	The name of a JSON file that contains the options you want described. Type: String	No

Output

The command returns a table with the following information:

- **Options—**A list of the configuration options.
- **SolutionStackName—**The name of the `SolutionStack` these configuration options belong to.

Examples

Describing Configuration Options for an Environment

This example shows how to describe configuration options for an environment.

```
PROMPT> elastic-beanstalk-describe-configuration-options -a MySampleApp -t my
configtemplate -e MySampleAppEnv
```

elastic-beanstalk-describe-configuration-settings

Description

Returns a description of the settings for the specified configuration set, that is, either a configuration template or the configuration set associated with a running environment.

When describing the settings for the configuration set associated with a running environment, it is possible to receive two sets of setting descriptions. One is the deployed configuration set, and the other is a draft configuration of an environment that is either in the process of deployment or that failed to deploy.

Syntax

`elastic-beanstalk-describe-configuration-settings -a [`*`name`*`] [-t [`*`name`*`] | -e [`*`name`*`]]`

Options

Name	Description	Required

`-a`
`--application-name` *`name`*
	The application name for the environment or configuration template.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		Yes

`-t`
`--template-name` *`name`*
	The name of the configuration template to describe. If no configuration template is found with this name, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Conditional: You must specify either this parameter or an environment name, but not both. If you specify both, Elastic Beanstalk returns an `InvalidParameterValue` error. If you do not specify either, Elastic Beanstalk returns a `MissingRequiredParameter` error.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		Conditional

`-e`
`--environment-name` *`name`*
	The name of the environment to describe.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		Conditional

Output

The command returns a table with the following information:

- **ConfigurationSettings—**A list of the configuration settings.

Examples

Describing Configuration Settings for an Environment

This example shows how to describe the configuration options for an environment.

```
PROMPT> elastic-beanstalk-describe-configuration-settings -a MySampleApp -e
MySampleAppEnv
```

Related Operations

- elastic-beanstalk-delete-environment-configuration (p. 492)

elastic-beanstalk-describe-environment-resources

Description

Returns AWS resources for this environment.

Syntax

elastic-beanstalk-describe-environment-resources [-e [*name*] | -E [*id*]]

Options

Name	Description	Required
`-e` `--environment-name` *name*	The name of the environment to retrieve AWS resource usage data. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	Conditional
`-E` `--environment-id` *id*	The ID of the environment to retrieve AWS resource usage data. Type: String Default: None	Conditional

Output

The command returns a table with the following information:

- **AutoScalingGroups—**A list of AutoScalingGroups used by this environment.
- **EnvironmentName—**The name of the environment.
- **Instances—**The Amazon EC2 instances used by this environment.
- **LaunchConfigurations—**The Auto Scaling launch configurations in use by this environment.
- **LoadBalancers—**The LoadBalancers in use by this environment.
- **Triggers—**The AutoScaling triggers in use by this environment.

Examples

Describing Environment Resources for an Environment

This example shows how to describe environment resources for an environment.

```
PROMPT> elastic-beanstalk-describe-environment-resources -e MySampleAppEnv
```

elastic-beanstalk-describe-environments

Description

Returns descriptions for existing environments.

Syntax

```
elastic-beanstalk-describe-environments -e [names [,name...]] -E [ids [,id...]]
-a [name] -l [label] -d -D [timestamp]
```

Options

Name	Description	Required

`-e`
`--environment-names` *`names`*
	A list of environment names.
	Type: String[]
	Default: None
		No

`-E`
`--environment-ids` *`ids`*
	A list of environment IDs.
	Type: String[]
	Default: None
		No

`-a`
`--application-name` *`name`*
	A list of descriptions associated with the application.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		No

`-l`
`--version-label` *`label`*
	A list of descriptions associated with the application version.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		No

`-d` `--include-deleted`	Indicates whether to include deleted environments. `true`: Environments that have been deleted after `--include-deleted-back-to` are displayed. `false`: Do not include deleted environments. Type: Boolean Default: `true`	No

`-D`
`--include-deleted-back-to` *`timestamp`*
	If --include-deleted is set to `true`, then a list of environments that were deleted after this date are displayed.
	Type: Date Time
	Default: None
		No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this environment.

- **CNAME—**The URL to the CNAME for this environment.
- **DateCreated—**The date the environment was created.
- **DateUpdated—**The date the environment was last updated.
- **Description—**The description of the environment.
- **EndpointURL—**The URL to the LoadBalancer for this environment.
- **EnvironmentID—**The ID of this environment.
- **EnvironmentName—**The name of this environment.
- **Health—**Describes the health status of the environment. Elastic Beanstalk indicates the failure levels for a running environment:
  - `Red`: Indicates the environment is not responsive. Occurs when three or more consecutive failures occur for an environment.
  - `Yellow`: Indicates that something is wrong. Occurs when two consecutive failures occur for an environment.
  - `Green`: Indicates the environment is healthy and fully functional.
  - `Gray`: Default health for a new environment. The environment is not fully launched and health checks have not started or health checks are suspended during an `UpdateEnvironment` or `RestartEnvironment`request.
- **Resources—**A list of AWS resources used in this environment.
- **SolutionStackName—**The name of the `SolutionStack` deployed with this environment.
- **Status—**The current operational status of the environment:
  - `Launching`: Environment is in the process of initial deployment.
  - `Updating`: Environment is in the process of updating its configuration settings or application version.
  - `Ready`: Environment is available to have an action performed on it, such as update or terminate.
  - `Terminating`: Environment is in the shut-down process.
  - `Terminated`: Environment is not running.
- **TemplateName—**The name of the configuration template used to originally launch this environment.
- **VersionLabel—**The application version deployed in this environment.

Examples

Describing Environments

This example shows how to describe existing environments.

```
PROMPT> elastic-beanstalk-describe-environments
```

elastic-beanstalk-describe-events

Description

Returns a list of event descriptions matching criteria up to the last 6 weeks.

Note
This action returns the most recent 1,000 events from the specified `NextToken`.

Syntax

```
elastic-beanstalk-describe-events -a [name] -e [name] -E [id] -l [label] -L
[timestamp] -m [count] -n [token] -r [id] -s [level] -S [timestamp] -t [name]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-e` `--environment-name` *name*	The name of the environment. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	No
`-E` `--environment-id` *id*	The ID of the environment. Type: String Default: None	No
`-l` `--version-label` *label*	The application version. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-L` `--end-time` *timestamp*	If specified, a list of events that occurred up to but not including the specified time is returned. Type: Date Time Default: None	No
`-m` `--max-records` *count*	Specifies the maximum number of events that can be returned, beginning with the most recent event. Type: Integer Default: None	No
`-n` `--next-token` *token*	Pagination token. Used to return the next batch of results. Type: String Default: None	No

Name	Description	Required

`-r`
`--request-id` *`id`*
	The request ID.
	Type: String
	Default: None
		No

`-s`
`--severity` *`level`*
	If specified, a list of events with the specified severity level or higher is returned.
	Type: String
	Valid Values: `TRACE` | `DEBUG` | `INFO` | `WARN` | `ERROR` | `FATAL`
	Default: None
		No

`-S`
`--start-time` *`timestamp`*
	If specified, a list of events that occurred after the specified time is returned.
	Type: Date Time
	Default: None
		No

`-t`
`--template-name` *`name`*
	The name of the configuration template.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with the event.
- **EnvironmentName—**The name of the environment associated with the event.
- **EventDate—**The date of the event.
- **Message—**The event's message.
- **RequestID—**The web service request ID for the activity of this event.
- **Severity—**The severity level of the event.
- **TemplateName—**The name of the configuration associated with this event.
- **VersionLabel—**The release label for the application version associated with this event.

Examples

Describing Events for an Environment with a Security Level

This example shows how to describe events that have a severity level of WARN or higher for an environment.

```
PROMPT> elastic-beanstalk-describe-events -e MySampleAppEnv -s WARN
```

elastic-beanstalk-list-available-solution-stacks

Description

Returns a list of available solution stack names.

Syntax

```
elastic-beanstalk-list-available-solution-stacks
```

Output

The command returns of available solution stack names.

Examples

Listing the Available Solution Stacks

This example shows how to get the list of available solution stacks.

```
PROMPT> elastic-beanstalk-list-available-solution-stacks
```

elastic-beanstalk-rebuild-environment

Description

Deletes and recreates all of the AWS resources (for example: the Auto Scaling group, LoadBalancer, etc.) for a specified environment and forces a restart.

Syntax

```
elastic-beanstalk-rebuild-environment [-e [name] | -E [id]]
```

Options

Name	Description	Required

`-e`
`--environment-name` *`name`*
	A name of the environment to rebuild.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		Conditional

`-E`
`--environment-id` *`id`*
	The ID of the environment to rebuild.
	Type: String
	Default: None
		Conditional

Output

The command outputs `Rebuilding environment`.

Examples

Rebuilding an Environment

This example shows how to rebuild an environment.

```
PROMPT> elastic-beanstalk-rebuild-environment -e MySampleAppEnv
```

elastic-beanstalk-request-environment-info

Description

Initiates a request to compile the specified type of information of the deployed environment.

Setting the InfoType to `tail` compiles the last lines from the application server log files of every Amazon EC2 instance in your environment. Use RetrieveEnvironmentInfo to access the compiled information.

Syntax

```
elastic-beanstalk-request-environment-info [-e [name] | -E [id]] -i [type]
```

Options

Name	Description	Required

`-e`
`--environment-name` *`name`*
	The name of the environment of the requested data.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		Conditional

`-E`
`--environment-id` *`id`*
	The ID of the environment of the requested data.
	Type: String
	Default: None
		Conditional

`-i`
`--info-type` *`type`*
	The type of information to request.
	Type: String
	Valid Values: `tail`
	Default: None
		Yes

Examples

Requesting Environment Information

This example shows how to request environment information.

```
PROMPT> elastic-beanstalk-request-environment-info -e MySampleAppEnv -i tail
```

Related Operations

- elastic-beanstalk-retrieve-environment-info (p. 509)

elastic-beanstalk-restart-app-server

Description

Causes the environment to restart the application container server running on each Amazon EC2 instance.

Syntax

```
elastic-beanstalk-restart-app-server [-e [name] | -E [id]]
```

Options

Name	Description	Required
-e --environment-name *name*	The name of the environment to restart the server for. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	Conditional
-E --environment-id *id*	The ID of the environment to restart the server for. Type: String Default: None	Conditional

Examples

Restarting the Application Server

This example shows how to restart the application server.

```
PROMPT> elastic-beanstalk-restart-app-server -e MySampleAppEnv
```

elastic-beanstalk-retrieve-environment-info

Description

Retrieves the compiled information from a RequestEnvironmentInfo request.

Syntax

elastic-beanstalk-retrieve-environment-info [-e [*name*] | -E [*id*]] -i [*type*]

Options

Name	Description	Required
-e --environment-name *name*	The name of the data's environment. If no environments are found, Elastic Beanstalk returns an InvalidParameterValue error. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	Conditional
-E --environment-id *id*	The ID of the data's environment. The name of the data's environment. If no environments are found, Elastic Beanstalk returns an InvalidParameterValue error. Type: String Default: None	Conditional
-i --info-type *type*	The type of information to retrieve. Type: String Valid Values: tail Default: None	Yes

Output

The command returns a table with the following information:

- **EC2InstanceId—**The Amazon EC2 instance ID for this information.
- **InfoType—**The type of information retrieved.
- **Message—**The retrieved information.
- **SampleTimestamp—**The time stamp when this information was retrieved.

Examples

Retrieving Environment Information

This example shows how to retrieve environment information.

```
PROMPT> elastic-beanstalk-retrieve-environment-info -e MySampleAppEnv -i tail
```

Related Operations

- elastic-beanstalk-request-environment-info (p. 507)

elastic-beanstalk-swap-environment-cnames

Description

Swaps the CNAMEs of two environments.

Syntax

```
elastic-beanstalk-swap-environment-cnames [-s [name] | -S [desc]] [-d [desc] |
-D [desc]]
```

Options

Name	Description	Required

`-s`
`--source-environment-name` *`name`*
	The name of the source environment.
	Type: String
	Default: None
		Conditional

`-S`
`--source-environment-id` *`id`*
	The ID of the source environment.
	Type: String
	Default: None
		Conditional

`-d`
`--destination-environment-name` *`name`*
	The name of the destination environment.
	Type: String
	Default: None
		Conditional

`-D`
`--destination-environment-id` *`id`*
	The ID of the destination environment.
	Type: String
	Default: None
		Conditional

Examples

Swapping Environment CNAMEs

This example shows how to swap the CNAME for two environments.

```
PROMPT> elastic-beanstalk-swap-environment-cnames -s MySampleAppEnv -d
MySampleAppEnv2
```

elastic-beanstalk-terminate-environment

Description

Terminates the specified environment.

Syntax

```
elastic-beanstalk-terminate-environment [-e [name] | -E [id]] -t
```

Options

Name	Description	Required
`-e` `--environment-name` *name*	The name of the environment to terminate. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	Conditional
`-E` `--environment-id` *id*	The ID of the environment to terminate. Type: String Default: None	Conditional
`-t` `--terminate-resources`	Indicates whether the associated AWS resources should shut down when the environment is terminated: • `true`: The specified environment as well as the associated AWS resources, such as Auto Scaling group and LoadBalancer, are terminated. • `false`: Elastic Beanstalk resource management is removed from the environment, but the AWS resources continue to operate. Type: Boolean Valid Values: `true` | `false` Default: true **Note** You can specify this parameter (`-t`) only for legacy environments because only legacy environments can have resources running when you terminate the environment.	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this environment.
- **CNAME—**The URL to the CNAME for this environment.
- **DateCreated—**The date the environment was created.

- **DateUpdated—**The date the environment was last updated.
- **Description—**The description of the environment.
- **EndpointURL—**The URL to the LoadBalancer for this environment.
- **EnvironmentID—**The ID of this environment.
- **EnvironmentName—**The name of this environment.
- **Health—**Describes the health status of the environment. Elastic Beanstalk indicates the failure levels for a running environment:
  - `Red`: Indicates the environment is not responsive. Occurs when three or more consecutive failures occur for an environment.
  - `Yellow`: Indicates that something is wrong. Occurs when two consecutive failures occur for an environment.
  - `Green`: Indicates the environment is healthy and fully functional.
  - `Gray`: Default health for a new environment. The environment is not fully launched and health checks have not started or health checks are suspended during an `UpdateEnvironment` or `RestartEnvironment`request.
- **Resources—**A list of AWS resources used in this environment.
- **SolutionStackName—**The name of the `SolutionStack` deployed with this environment.
- **Status—**The current operational status of the environment:
  - `Launching`: Environment is in the process of initial deployment.
  - `Updating`: Environment is in the process of updating its configuration settings or application version.
  - `Ready`: Environment is available to have an action performed on it, such as update or terminate.
  - `Terminating`: Environment is in the shut-down process.
  - `Terminated`: Environment is not running.
- **TemplateName—**The name of the configuration template used to originally launch this environment.
- **VersionLabel—**The application version deployed in this environment.

Examples

Terminating an Environment

This example shows how to terminate an environment.

```
PROMPT> elastic-beanstalk-terminate-environment -e MySampleAppEnv
```

elastic-beanstalk-update-application

Description

Updates the specified application to have the specified properties.

> **Note**
> If a property (for example, `description`) is not provided, the value remains unchanged. To clear these properties, specify an empty string.

Syntax

```
elastic-beanstalk-update-application -a [name] -d [desc]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application to update. If no such application is found, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-d` `--description` *desc*	A new description for the application. Type: String Default: If not specified, Elastic Beanstalk does not update the description. Length Constraints: Minimum value of 0. Maximum value of 200.	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application.
- **ConfigurationTemplate—**The names of the configuration templates associated with this application.
- **DateCreated—**The date the environment was created.
- **DateUpdated—**The date the environment was last updated.
- **Description—**The description of the environment.
- **Versions—**The names of the versions for this application.

Examples

Updating an Application

This example shows how to update an application.

```
PROMPT> elastic-beanstalk-update-application -a MySampleApp -d "My new descrip
tion"
```

elastic-beanstalk-update-application-version

Description

Updates the specified application version to have the specified properties.

> **Note**
> If a property (for example, `description`) is not provided, the value remains unchanged. To clear these properties, specify an empty string.

Syntax

`elastic-beanstalk-update-application-version -a [`*`name`*`] -l [`*`label`*`] -d [`*`desc`*`]`

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application associated with this version. If no such application is found, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-l` `--version-label`	The name of the version to update. If no application version is found with this label, Elastic Beanstalk returns an `InvalidParaemter-Value` error. Type: String Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-d` `--description`	A new description for the release. Type: String Default: If not specified, Elastic Beanstalk does not update the description. Length Constraints: Minimum value of 0. Maximum value of 200.	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this release.
- **DateCreated—**The creation date of the application version.
- **DateUpdated—**The last modified date of the application version.
- **Description—**The description of this application version.
- **SourceBundle—**The location where the source bundle is located for this version.
- **VersionLabel—**A label identifying the version for the associated application.

Examples

Updating an Application Version

This example shows how to update an application version.

```
PROMPT> elastic-beanstalk-update-application-version -a MySampleApp -d "My new
 version" -l "TestVersion 1"
```

elastic-beanstalk-update-configuration-template

Description

Updates the specified configuration template to have the specified properties or configuration option values.

Note
If a property (for example, `ApplicationName`) is not provided, its value remains unchanged. To clear such properties, specify an empty string.

Syntax

```
elastic-beanstalk-update-configuration-template -a [name] -t [name] -d [desc]
-f [filename] -F [filename]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application associated with the configuration template to update. If no application is found with this name, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-t` `--template-name` *name*	The name of the configuration template to update. If no configuration template is found with this name, UpdateConfigurationTemplate returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-d` `--description` *desc*	A new description for the configuration. Type: String Default: None Length Constraints: Minimum value of 0. Maximum value of 200.	No
`-f` `--options-file` *filename*	The name of a JSON file that contains option settings to update with the new specified option value. Type: String	No
`-F` `--options-to-remove-file` *value*	The name of a JSON file that contains configuration options to remove. Type: String Default: None	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this configuration set.
- **DateCreated—**The date (in UTC time) when this configuration set was created.
- **DateUpdated—**The date (in UTC time) when this configuration set was last modified.
- **DeploymentStatus—**If this configuration set is associated with an environment, the *DeploymentStatus* parameter indicates the deployment status of this configuration set:
  - `null`: This configuration is not associated with a running environment.
  - `pending`: This is a draft configuration that is not deployed to the associated environment but is in the process of deploying.
  - `deployed`: This is the configuration that is currently deployed to the associated running environment.
  - `failed`: This is a draft configuration that failed to successfully deploy.
- **Description—**The description of the configuration set.
- **EnvironmentName—**If not null, the name of the environment for this configuration set.
- **OptionSettings—**A list of configuration options and their values in this configuration set.
- **SolutionStackName—**The name of the solution stack this configuration set uses.
- **TemplateName—**If not null, the name of the configuration template for this configuration set.

Examples

Updating a Configuration Template

This example shows how to update a configuration template. For a list of configuration settings, see Configuration Options (p. 103).

```
PROMPT> elastic-beanstalk-update-configuration-template -a MySampleApp -t mycon
figtemplate -d "My updated configuration template" -f "options.txt"
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "my_custom_param_1",
        "Value": "firstvalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "my_custom_param_2",
        "Value": "secondvalue"
    }
]
```

Related Operations

- elastic-beanstalk-describe-configuration-options (p. 496)

elastic-beanstalk-update-environment

Description

Updates the environment description, deploys a new application version, updates the configuration settings to an entirely new configuration template, or updates select configuration option values in the running environment.

Attempting to update both the release and configuration is not allowed and Elastic Beanstalk returns an `InvalidParameterCombination` error.

When updating the configuration settings to a new template or individual settings, a draft configuration is created and `DescribeConfigurationSettings` for this environment returns two setting descriptions with different `DeploymentStatus` values.

Syntax

`elastic-beanstalk-update-environment [-e [`*`name`*`] | -E [`*`id`*`]] -l [`*`label`*`] -t [`*`name`*`] -d [`*`desc`*`] -f [`*`filename`*`] -F [`*`filename`*`]`

Options

Name	Description	Required

`-e`
`--environment-name` *`name`*
	The name of the environment to update. If no environment with this name exists, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Type: String
	Default: None
	Length Constraints: Minimum value of 4. Maximum value of 23.
		Conditional

`-E`
`--environment-id` *`id`*
	The ID of the environment to update. If no environment with this ID exists, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Type: String
	Default: None
		Conditional

`-l`
`--version-label` *`label`*
	If this parameter is specified, Elastic Beanstalk deploys the named application version to the environment. If no such application version is found, Elastic Beanstalk returns an `InvalidParameterValue` error.
	Type: String
	Default: None
	Length Constraints: Minimum value of 1. Maximum value of 100.
		No

Name	Description	Required
`-t` `--template-name` *name*	If this parameter is specified, Elastic Beanstalk deploys this configuration template to the environment. If no such configuration template is found, Elastic Beanstalk returns an `InvalidParameterValue` error. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-d` `--description` *desc*	If this parameter is specified, Elastic Beanstalk updates the description of this environment. Type: String Default: None Length Constraints: Minimum value of 0. Maximum value of 200.	No
`-f` `--options-file` *filename*	A file containing option settings to update. If specified, Elastic Beanstalk updates the configuration set associated with the running environment and sets the specified configuration options to the requested values. Type: String Default: None	No
`-F` `--options-to-remove-file` *filename*	A file containing options settings to remove. If specified, Elastic Beanstalk removes the option settings from the configuration set associated with the running environment. Type: String Default: None	No

Output

The command returns a table with the following information:

- **ApplicationName—**The name of the application associated with this environment.
- **CNAME—**The URL to the CNAME for this environment.
- **DateCreated—**The date the environment was created.
- **DateUpdated—**The date the environment was last updated.
- **Description—**The description of the environment.
- **EndpointURL—**The URL to the LoadBalancer for this environment.
- **EnvironmentID—**The ID of this environment.
- **EnvironmentName—**The name of this environment.
- **Health—**Describes the health status of the environment. Elastic Beanstalk indicates the failure levels for a running environment:
  - `Red`: Indicates the environment is not responsive. Occurs when three or more consecutive failures occur for an environment.
  - `Yellow`: Indicates that something is wrong. Occurs when two consecutive failures occur for an environment.

  - `Green`: Indicates the environment is healthy and fully functional.
  - `Gray`: Default health for a new environment. The environment is not fully launched and health checks have not started or health checks are suspended during an `UpdateEnvironment` or `RestartEnvironment`request.
- **Resources—**A list of AWS resources used in this environment.
- **SolutionStackName—**The name of the `SolutionStack` deployed with this environment.
- **Status—**The current operational status of the environment:
  - `Launching`: Environment is in the process of initial deployment.
  - `Updating`: Environment is in the process of updating its configuration settings or application version.
  - `Ready`: Environment is available to have an action performed on it, such as update or terminate.
  - `Terminating`: Environment is in the shut-down process.
  - `Terminated`: Environment is not running.
- **TemplateName—**The name of the configuration template used to originally launch this environment.
- **VersionLabel—**The application version deployed in this environment.

Examples

Updating an Existing Environment

This example shows how to update an existing environment. It passes in a file called options.txt that updates the size of the instance to a t1.micro and sets two environment variables. For a list of possible configuration settings, see Configuration Options (p. 103).

```
PROMPT> elastic-beanstalk-update-environment -e MySampleAppEnv -f "options.txt"
```

options.txt

```
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "InstanceType",
        "Value": "t1.micro"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "my_custom_param_1",
        "Value": "firstvalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "my_custom_param_2",
        "Value": "secondvalue"
    }
]
```

elastic-beanstalk-validate-configuration-settings

Description

Takes a set of configuration settings and either a configuration template or environment, and determines whether those values are valid.

This action returns a list of messages indicating any errors or warnings associated with the selection of option values.

Syntax

```
elastic-beanstalk-validate-configuration-settings -a [name] -t [name] -e [name]
-f [filename]
```

Options

Name	Description	Required
`-a` `--application-name` *name*	The name of the application that the configuration template or environment belongs to. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	Yes
`-t` `--template-name` *name*	The name of the configuration template to validate the settings against. Condition: You cannot specify both this and the environment name. Type: String Default: None Length Constraints: Minimum value of 1. Maximum value of 100.	No
`-e` `--environment-name` *name*	The name of the environment to validate the settings against. Type: String Default: None Length Constraints: Minimum value of 4. Maximum value of 23.	No
`-f` `--options-file` *filename*	The name of a JSON file that contains a list of options and desired values to evaluate. Type: String	Yes

Output

The command returns a table with the following information:

- **Message—**A message describing the error or warning.
- **Namespace**
- **OptionName**
- **Severity—**An indication of the severity of this message:

- `error`: This message indicates that this is not a valid settings for an option.
- `warning`: This message provides information you should take into account.

Examples

Validating Configuration Settings for an Environment

This example shows how to validate the configuration settings for an environment.

```
PROMPT> elastic-beanstalk-validate-configuration-settings -a MySampleApp -e
MySampleAppEnv -f MyOptionSettingsFile.json
```

AWS Command Line Interface

Amazon Web Services (AWS) now offers the AWS Command Line Interface (CLI) as the new, unified API-based command line tool to manage multiple AWS services, including Elastic Beanstalk. The AWS CLI replaces the Elastic Beanstalk API-based command line tool. For information about setting up and configuring the AWS CLI, see What is the AWS Command Line Interface?

For information about the Elastic Beanstalk commands in the AWS CLI, see elasticbeanstalk in the *AWS Command Line Interface Reference*. The Elastic Beanstalk API CLI is still available, but is no longer updated. If you need information about the API CLI, see Elastic Beanstalk API Command Line Interface (deprecated) (p. 472).

Migrating Elastic Beanstalk API CLI Commands to AWS Command Line Interface Commands

The following table lists the Elastic Beanstalk API-based CLI commands and their equivalent commands in the AWS CLI.

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
`elastic-beanstalk-check-dns-availability`	`check-dns-availability`	Get-EBDNSAvailability
`elastic-beanstalk-create-application`	`create-application`	New-EBApplication

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-create-application-version	create-application-version	New-EBApplicationVersion
elastic-beanstalk-create-configuration-template	create-configuration-template	New-EBConfigurationTemplate
elastic-beanstalk-create-environment	create-environment	New-EBEnvironment
elastic-beanstalk-create-storage-location	create-storage-location	New-EBStorageLocation
elastic-beanstalk-delete-application	delete-application	Remove-EBApplication
elastic-beanstalk-delete-application-version	delete-application-version	Remove-EBApplicationVersion

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-delete-configuration-template	delete-configuration-template	Remove-EBConfigurationTemplate
elastic-beanstalk-delete-environment-configuration	delete-environment-configuration	Remove-EBEnvironmentConfiguration
elastic-beanstalk-describe-application-versions	describe-application-versions	Get-EBApplicationVersion
elastic-beanstalk-describe-applications	describe-applications	Get-EBApplication
elastic-beanstalk-describe-configuration-options	describe-configuration-options	Get-EBConfigurationOption

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-describe-configuration-settings	describe-configuration-settings	Get-EBConfigurationSetting
elastic-beanstalk-describe-environment-resources	describe-environment-resources	Get-EBEnvironmentResource
elastic-beanstalk-describe-environments	describe-environments	Get-EBEnvironment
elastic-beanstalk-describe-events	describe-events	Get-EBEvent
elastic-beanstalk-list-available-solution-stacks	list-available-solution-stacks	Get-EBAvailableSolutionStack
elastic-beanstalk-rebuild-environment	rebuild-environment	Start-EBEnvironmentRebuild
elastic-beanstalk-request-environment-info	request-environment-info	Request-EBEnvironmentInfo

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-restart-app-server	restart-app-server	Restart-EBAppServer
elastic-beanstalk-retrieve-environment-info	retrieve-environment-info	Get-EBEnvironmentInfo
elastic-beanstalk-swap-environment-cnames	swap-environment-cnames	Set-EBEnvironmentCNAME
elastic-beanstalk-terminate-environment	terminate-environment	Stop-EBEnvironment
elastic-beanstalk-update-application	update-application	Update-EBApplication
elastic-beanstalk-update-application-version	update-application-version	Update-EBApplicationVersion

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-update-configuration-template	update-configuration-template	Update-EBConfigurationTemplate
elastic-beanstalk-update-environment	update-environment	Update-EBEnvironment
elastic-beanstalk-validate-configuration-settings	validate-configuration-settings	Test-EBConfigurationSetting

Troubleshooting

Topics

This section provides a table of the most common Elastic Beanstalk issues and how to resolve or work around them.

Issue	Workaround
Unable to connect to Amazon RDS from Elastic Beanstalk.	To connect RDS to your Elastic Beanstalk application, do the following: • Make sure RDS is in the same Region as your Elastic Beanstalk application. • Make sure the RDS security group for your instance has an authorization for the Amazon EC2 security group you are using for your Elastic Beanstalk environment. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information about configuring your EC2 security group, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*. • For Java, make sure the MySQL JAR file is in your WEB-INF/lib. See Using Amazon RDS and MySQL Connector/J (p. 592) for more details.
Experiencing a couple of seconds of downtime when updating the version of my application.	Because Elastic Beanstalk uses a drop-in upgrade process, there might be a few seconds of downtime. There is no workaround at this time.
Unable to connect to another Amazon EC2 instance using the Amazon EC2 security group for your Elastic Beanstalk environment.	Create a CNAME record and point this to the public DNS of the Amazon EC2 instance.

Issue	Workaround
How can I change my application URL from myapp.elasticbeanstalk.com to www.myapp.com?	Register in a DNS server a CNAME record such as: www.mydomain.com CNAME mydomain.elasticbeanstalk.com.
Unable to specify a specific Availability Zone for my Elastic Beanstalk application.	You can pick specific AZs using the APIs, CLI, Eclipse plug-in, or Visual Studio plug-in. For instructions about using the AWS Management Console to specify an Availability Zone, see Configuring Auto Scaling with Elastic Beanstalk (p. 177).
Getting charged for my Elastic Beanstalk application.	The default settings for Elastic Beanstalk do not incur any additional charges. However, if you modified the default settings by changing the Amazon EC2 instance type or adding additional Amazon EC2 instance, charges may be accrued. For information about the free tier, see http://aws.amazon.com/free. If you have questions about your account, contact our customer service team directly.
How can I receive notifications by SMS?	If you specify an SMS email address, such as one constructed on http://www.makeuseof.com/tag/email-to-sms, you will receive the notifications by SMS. To subscribe to more than one email address, you can use the Elastic Beanstalk command line to register an SNS topic with an environment.
How do I upgrade my instance type to an m1.large?	You can upgrade your instance type to an m1.large by launching a new environment using a 64-bit container. Once your environment is launched, you can select **Edit Configuration** to select m1.large. See Launching a New AWS Elastic Beanstalk Environment (p. 55) for instructions on launching a new environment. See Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199) for instructions for editing your configuration.
Unable to connect to Elastic Beanstalk when deploying using the AWS Toolkit for Eclipse.	Try one of the following: • Make sure you are running the latest distribution of Eclipse. • Make sure you've signed up your account for Elastic Beanstalk (and have received an email confirmation). • Check the "Error Log" view in Eclipse to see if there's any additional information or stack traces.
How can I get Amazon EBS to work with Elastic Beanstalk?	The default AMIs are EBS-backed; however, the root volume is deleted upon termination of the instance. You can alter the delete on termination behavior by using: `$ ec2-modify-instance-attribute -b '/dev/sdc=<vol-id>:false` as described in the EC2 Command Line Reference.
Servers that were created in the AWS Management Console do not appear in the Toolkit for Eclipse	You can manually import servers by following the instructions at Importing Existing Environments into Eclipse (p. 584).
Environments are launched, but with command timeout errors	You can increase the command timeout period. For more information, see Launch and Update Environment Operations Succeeded but with Command Timeouts (p. 535).

Issue	Workaround
Unable to update an environment	You may have received the following event: **You cannot configure an AWS Elastic Beanstalk environment with values for both the Elastic Load Balancing Target option and Application Healthcheck URL option** This indicates that, Elastic Beanstalk received an option setting that conflicts with the value you provided for the **Application Healthcheck URL** option setting in the `aws:elasticbeanstalk:application` namespace. Remove the **Target** option (in the `aws:elb:healthcheck` namespace) from the `option_settings` key in the `.ebextensions` configuration file and then try updating your environment again. For more information about configuration files, see Elastic Beanstalk Environment Configuration (p. 99).

Understanding Environment Launch Events

Topics

Elastic Beanstalk monitors the environment launch process to ensure your application is healthy. To determine your application's health, Elastic Beanstalk sends periodic requests to the application's health check URL (by default root or '/'). If your application experiences problems and does not respond to the health check, a variety of launch events are published. This section describes the most common launch events, why they happen, and how to address environment launch-related problems. For instructions on how to view your events, see View Events (p. 278). For more information about health check URL, see Health Checks (p. 190).

HTTP HEAD Request to Your Elastic Beanstalk URL Fails

"Failed to Perform HTTP HEAD Request to http://<yourapp>.elasticbeanstalk.com:80"

Elastic Beanstalk sends periodic HTTP HEAD requests to the health check URL. This event fires when the health check URL does not respond successfully (with HTTP code 200).

If you receive this event, try one or both of the following:

- Make sure that your application's health check URL exists. For example, if Elastic Beanstalk makes a health check request to `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.jsp`, ensure that `/myapp/index.jsp` exists and is accessible. Similarly, for PHP, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.php`, make sure that `/myapp/index.php` exists and is accessible. For ASP.NET, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/default.aspx`, make sure that `/myapp/default.aspx` exists and is accessible.
- Inspect previous events on the **Events** page in the AWS Management Console to ensure that your environment is healthy. For example, if instances of your environments are running at close to 100 percent CPU utilization, they may become unresponsive. Elastic Beanstalk will alert you via an event that reads **Instance <instance id> is experiencing CPU Utilization greater than 95.00%. Consider adjusting auto-scaling settings or upgrading to an instance type larger than a <instance type>.** See CPU Utilization Exceeds 95.00% (p. 532) for more information about this event.

CPU Utilization Exceeds 95.00%

"CPU Utilization Greater Than 95.00%"

If you receive an event that reads **Instance <instance id> is experiencing CPU utilization greater than 95.00%. Consider adjusting auto-scaling settings or upgrading to an instance type larger than a <instance type>.**, this means an instance is using the CPU for more than 95 percent of the time.

If your application can be parallelized across multiple instances, you can adjust auto-scaling settings by increasing the maximum number of instances and adjusting your scaling trigger. For instructions, see Configuring Auto Scaling with Elastic Beanstalk (p. 177).

If your application requires higher computing needs, you can upgrade the Amazon EC2 instance type for your environment. For instructions, see Configuring Amazon EC2 Server Instances with Elastic Beanstalk (p. 199).

Elastic Load Balancer Has Zero Healthy Instances

"Elastic Load Balancer awseb-<yourapp> Has Zero Healthy Instances"

When the health status of all of your instances changes from green to red, Elastic Beanstalk alerts you that your application has become unavailable. Make sure that your application's health check URL exists. For example, if Elastic Beanstalk makes a health check request to `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.jsp`, ensure that `/myapp/index.jsp` exists and is accessible. Similarly, for PHP, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.php`, make sure that `/myapp/index.php` exists and is accessible. For ASP.NET, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/default.aspx`, make sure that `/myapp/default.aspx` exists and is accessible.

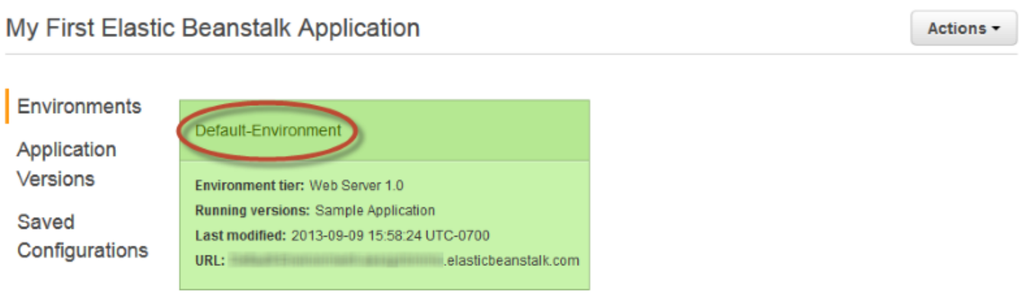
Elastic Load Balancer Cannot Be Found

"Elastic Load Balancer awseb-<yourapp> Cannot Be Found"

If you receive an **Elastic Load Balancer awseb-yourapp cannot be found. If this problem persists, try rebuilding your environment** event, the Elastic Load Balancer for your environment has been removed. This event usually occurs when an account owner or other authorized user manually removes the Elastic Load Balancer. To resolve this issue, you need to rebuild your environment.

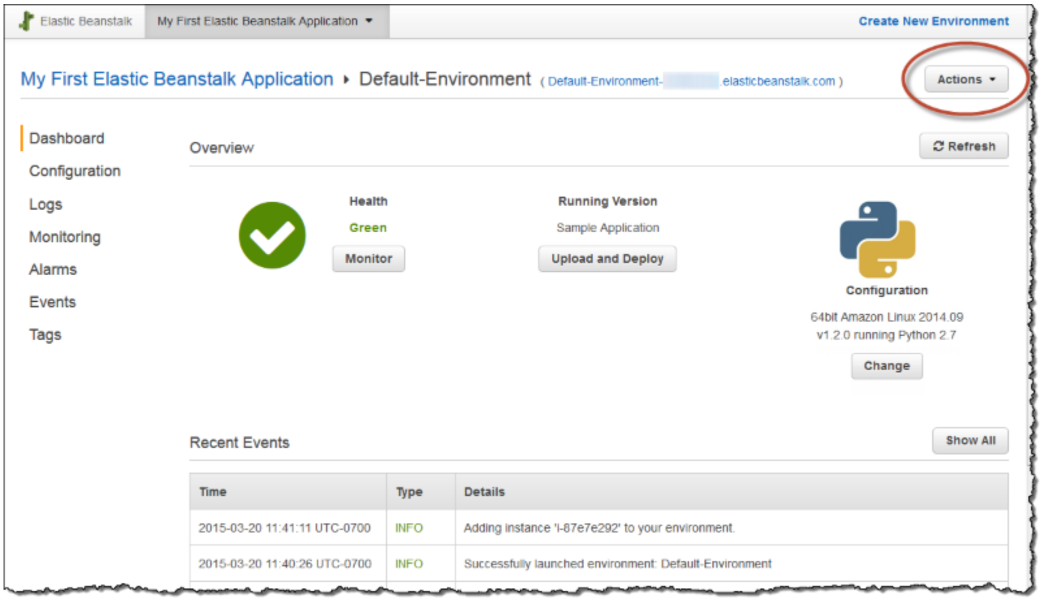
To rebuild your environment

1. From the Elastic Beanstalk console applications page, click the environment name that you want to rebuild.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

2. Click **Actions** and the select **Rebuild Environment**.



Health: Green
Running Version: Sample Application
Configuration: 64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

Time	Type	Details
2015-03-20 11:41:11 UTC-0700	INFO	Adding instance 'i-87e7e292' to your environment.
2015-03-20 11:40:26 UTC-0700	INFO	Successfully launched environment: Default-Environment

Instance Fails to Respond to Status Health Check

"Failed to Retrieve Status of Instance <instance id> 4 Consecutive Time(s)"

This event occurs when Elastic Beanstalk is not getting a response from the health check URL. Elastic Beanstalk will try to reach the health check URL 10 times over 10 minutes. If this event occurs, try one or both of the following:

- Make sure that your application's health check URL exists. For example, if Elastic Beanstalk makes a health check request to `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.jsp`, ensure that `/myapp/index.jsp` exists and is accessible. Similarly, for PHP, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.php`, make sure that `/myapp/index.php` exists and is accessible. For ASP.NET, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/default.aspx`, make sure that `/myapp/default.aspx` exists and is accessible.

- Inspect previous events on the **Events** page on the AWS Management Console to ensure that your environment is healthy. For example, if instances of your environments are running at close to 100 percent CPU utilization, they may become unresponsive. Elastic Beanstalk will alert you via an event that reads **Instance <instance id> is experiencing CPU Utilization greater than 95.00%. Consider adjusting auto-scaling settings or upgrading to an instance type larger than a <instance type>**. For more information, see CPU Utilization Exceeds 95.00% (p. 532).

Environment Launch Fails

"Failed to Launch Environment"

This event occurs when Elastic Beanstalk attempts to launch an environment and encounters failures along the way. Previous events on the **Events** page will alert you to the root cause of this issue.

Amazon EC2 Instance Launch Fails

"EC2 Instance Launch Failure. Waiting for a New EC2 Instance to Launch..."

This event occurs when an Amazon EC2 instance launch fails. If this event occurs, try one or both of the following:

- Check the service health dashboard to ensure that the Elastic Compute Cloud (Amazon EC2) service is green.
- Make sure that you are not over the Amazon EC2 instance limit for your account (the default is 10 instances). To request an increase to the Amazon EC2 instances, complete the form available at https://console.aws.amazon.com/support/home#/case/create?issueType=service-limit-increase&limitType=service-code-ec2-instances.

Application Does Not Enter the Ready State Within the Timeout Period

"Exceeded Maximum Amount of Time to Wait for the Application to Become Available. Setting Environment Ready"

During the launch of a new environment, Elastic Beanstalk monitors the environment to make sure that the application is available. If the application is still unavailable after 6 minutes have passed, the environment enters the ready state; this allows you to take action and make configuration changes. If this event occurs, try one or both of the following:

- Make sure that your application's health check URL exists. For example, if Elastic Beanstalk makes a health check request to `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.jsp`, ensure that `/myapp/index.jsp` exists and is accessible. Similarly, for PHP, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.php`, make sure that `/myapp/index.php` exists and is accessible. For ASP.NET, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/default.aspx`, make sure that `/myapp/default.aspx` exists and is accessible.
- Make sure you have uploaded a valid .war file or a .zip file.

Environment Launches but with Issues

"Launched Environment: <environment id>. However, There Were Issues During Launch. See Event Log for Details"

During the launch of a new environment, Elastic Beanstalk monitors the environment to make sure that the application is available. If the application is still unavailable after 6 minutes have passed, the environment enters the ready state; this allows you to take action and make configuration changes. If this event occurs, try one or both of the following:

- Make sure that your application's health check URL exists. For example, if Elastic Beanstalk makes a health check request to `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.jsp`, ensure that `/myapp/index.jsp` exists and is accessible. Similarly, for PHP, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/index.php`, make sure that `/myapp/index.php` exists and is accessible. For ASP.NET, if you have `http://healthcheckrocks.elasticbeanstalk.com:80/myapp/default.aspx`, make sure that `/myapp/default.aspx` exists and is accessible.
- Make sure you have uploaded a valid .war file or .zip file.
- Check previous events on the **Events** page.

Amazon EC2 Instances Fail to Launch within the Wait Period

"'CREATE_FAILED' Reason: The following resource(s) failed to create: [AWSEBInstanceLaunchWaitCondition]"

If you receive an event that indicates **Stack named '*`awseb-stack-name`*' aborted operation. Current state: 'CREATE_FAILED' Reason: The following resource(s) failed to create: [AWSEBInstanceLaunchWaitCondition]**, this means the Amazon EC2 instances did not communicate to Elastic Beanstalk that they were launched successfully.

If you use Amazon VPC with Elastic Beanstalk, Amazon EC2 instances deployed in a private subnet cannot communicate directly with the Internet. Amazon EC2 instances must have Internet connectivity to communicate to Elastic Beanstalk that they were successfully launched. To provide EC2 instances in a private subnet with Internet connectivity, you must add a load balancer and NAT to the public subnet. You must create the appropriate routing rules for inbound and outbound traffic through the load balancer and NAT. You must also configure the default Amazon VPC security group to allow traffic from the Amazon EC2 instances to the NAT instance. For more information, see Example: Launching a Load-Balancing, Autoscaling Environment with Public and Private Resources in a VPC (p. 302).

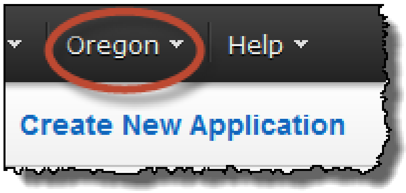
Launch and Update Environment Operations Succeeded but with Command Timeouts

"Create environment operation is complete, but with command timeouts. Try increasing the timeout period." or **"Update environment operation is complete, but with command timeouts. Try increasing the timeout period.'**

This event message appears when an environment is successfully created or updated, but some commands were not completed on the Amazon EC2 instances in the environment within the expected time period. In this situation, check the event stream or logs for details about which on-instance commands were not completed. For more information about viewing events, see View Events (p. 278). For more information about logging, see Instance Logs (p. 282). Then try increasing the command timeout period by doing one of the following.

To configure a command timeout using the AWS Management Console:

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the region list, select a region in which to create the Elastic Beanstalk application.



3. From the region list, select the region that includes the environment that you want to work with.
4. On the Elastic Beanstalk console applications page, click the name of the environment with the settings that you want to edit.
5. In the navigation pane, click **Configuration**.
6. On the **Configuration** page, next to **Updates and Deployments**, click the gear icon (⚙).
7. Under **Command Timeout**, change **Command timeout** to **`900`**, and then click **Save**.

To configure a command timeout using a configuration file

- Add the following to a configuration file with the extension `.config` that you place in an `.ebextensions` top-level directory of your source bundle. For more information, see Customizing the Software on EC2 Instances Running Linux or Customizing the Software on EC2 Instances Running Windows.

```
option_settings:
  - namespace: aws:elasticbeanstalk:command
    option_name: Timeout
    value: 900
```

To configure a command timeout using the API

- Call **CreateEnvironment** or **UpdateEnvironment** with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv2`
  - *`VersionLabel`* = `Version2`
  - *`SolutionStackName`* = `32bit Amazon Linux running Tomcat 6`
  - *`ApplicationName`* = `SampleApp`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:command`
  - *`OptionSettings.member.1.OptionName`* = `Timeout`
  - *`OptionSettings.member.1.Value`* = `900`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&SolutionStackName=32bit%20Amazon%20Linux%20running%20Tomcat%206
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acommand
&OptionSettings.member.1.OptionName=Timeout
&OptionSettings.member.1.Value=900
&AuthParams
```

Application Version Deployment Failed

Error: Failed to create the AWS Elastic Beanstalk application version

Elastic Beanstalk displays this event message when one or more of the following is true:

- The size of the application version (.war or .zip file) exceeds 512 MB. You must reduce the size of your application source bundle if you want to upload it to your application.
- Your environment has reached the maximum number of application versions. You must delete application versions until your environment has fewer than 500 application versions. For more information, see Delete an Application Version (p. 41).

Troubleshooting Docker Containers

This section lists the most common Elastic Beanstalk error messages related to Docker containers, the cause of the errors, and how to resolve or work around them. These errors cause your environment to fail to enter a healthy, Green status. Error messages can appear as events on the Elastic Beanstalk console **Events** page or in the tail logs.

Dockerfile Syntax Errors

Error Messages

The **Events** page displays the following sequence of error messages:

[Instance: *<instance ID>* Module: AWSEBAutoScalingGroup ConfigSet: null] Command failed on instance. Return code: 1 Output: Error occurred during build: Command hooks failed.

Failed to pull Docker image :latest: *<date and timestamp>* Invalid repository name (), only [a-z0-9-_.] are allowed. Tail the logs for more details.

Script /opt/elasticbeanstalk/hooks/appdeploy/enact/01build.sh failed with returncode 1

Additionally, the tail log displays the following series of error messages:

cat: Dockerrun.aws.json: No such file or directory

<date and timestamp> Invalid repository name (), only [a-z0-9-_.] are allowed

<date and timestamp> [ERROR] (1938 MainThread) [directoryHooksExecutor.py-33] [root directoryHooksExecutor error] Script /opt/elasticbeanstalk/hooks/appdeploy/enact/01build.sh failed with returncode 1

Cause

The dockerfile is syntactically incorrect.

Solution

Check the syntax of the dockerfile using a JSON validator. Also verify the dockerfile contents against the requirements described in Single Container Docker Configuration (p. 542).

Dockerrun.aws.json Syntax Errors

Error Messages

The **Events** page displays the following sequence of error messages:

[Instance: *<instance ID>* Module: AWSEBAutoScalingGroup ConfigSet: null] Command failed on instance. Return code: 1 Output: Error occurred during build: Command hooks failed.

Script /opt/elasticbeanstalk/hooks/appdeploy/enact/01build.sh failed with returncode 1

Failed to pull Docker image :latest: *<date and timestamp>* Invalid repository name (), only [a-z0-9-_.] are allowed. Tail the logs for more details.

Additionally, the tail logs display the following sequence of error messages:

parse error: Invalid numeric literal

<date and timestamp> Invalid repository name (), only [a-z0-9-_.] are allowed

[ERROR] (1942 MainThread) [directoryHooksExecutor.py-33] [root directoryHooksExecutor error] Script /opt/elasticbeanstalk/hooks/appdeploy/enact/01build.sh failed with returncode 1

Cause

The `dockerrun.aws.json` file is syntactically incorrect.

Solution

Check the syntax of the dockerfile using a JSON validator. Also verify the dockerfile contents against <topic with dockerfile requirements>.

No EXPOSE Directive Found in Dockerfile

Error Messages

The **Events** page includes the following error message:

No EXPOSE directive found in Dockerfile, abort deployment

Cause

The dockerfile or the `dockerrun.aws.json` file does not declare the container port.

Solution

Do one of the following to declare the port exposed on the image:

- In the dockerfile, include an `EXPOSE` instruction. For example:

  ```
  EXPOSE 80
  ```

- In the dockerrun.aws.json file, include the `Ports` key. For example:

```
{
    "Ports": [
      {
        "ContainerPort": "80"
      }
```

```
    ]
}
```

Note

When the dockerfile exists and includes the `EXPOSE` instruction, Elastic Beanstalk ignores the `Ports` key and value in the `dockerrun.aws.json` file.

Invalid EC2 Key Pair and/or S3 Bucket in Dockerrun.aws.json

Error Messages

The **Events** page displays the following sequence of error messages:

[Instance: *<instance ID>* Module: AWSEBAutoScalingGroup ConfigSet: null] Command failed on instance. Return code: 1 Output: Error occurred during build: Command hooks failed.

Failed to download authentication credentials *<repository>* from *<bucket name>*

Script /opt/elasticbeanstalk/hooks/appdeploy/enact/01build.sh failed with returncode 1

Cause

The `dockerrun.aws.json` provides an invalid EC2 key pair and/or S3 bucket for the `.dockercfg` file. Or, the instance profile does not have GetObject authorization for the S3 bucket.

Solution

Verify that the `.dockercfg` file contains a valid S3 bucket and EC2 key pair. Grant permissions for the action `s3:GetObject` to the IAM role in the instance profile. For an example policy, go to Using IAM Roles with Elastic Beanstalk (p. 332)

Deploying Elastic Beanstalk Applications from Docker Containers

Elastic Beanstalk supports the deployment of web applications from Docker containers. With Docker containers, you can define your own runtime environment. You can choose your own platform, programming language, and any application dependencies (such as package managers or tools), that aren't supported by other platforms. Docker containers are self-contained and include all the configuration information and software your web application requires to run.

By using Docker with Elastic Beanstalk, you have an infrastructure that automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring. You can manage your web application in an environment that supports the range of services that are integrated with Elastic Beanstalk, including but not limited to VPC, RDS, and IAM. For more information about Docker, including how to install it, what software it requires, and how to use Docker images to launch Docker containers, go to Docker: the Linux container engine.

> **Note**
> If a Docker container running in an Elastic Beanstalk environment crashes or is killed for any reason, Elastic Beanstalk restarts it automatically.

Docker Platform Configurations

The Docker platform for Elastic Beanstalk has two generic configurations (single container and multicontainer), and several preconfigured containers.

See the Supported Platforms (p. 24) page for details on the currently supported version of each configuration.

Single Container Docker

The single container configuration can be used to deploy a Docker image (described in a Dockerfile or Dockerrun.aws.json definition) and source code to EC2 instances running in an Elastic Beanstalk environment. Use the single container configuration when you only need to run one container per instance.

For samples and help getting started with a single container Docker environment, see Single Container Docker (p. 542). For detailed information on the container definition formats and their use, see Single Container Docker Configuration (p. 542).

Multicontainer Docker

The other basic configuration, Multicontainer Docker, uses the Amazon EC2 Container Service to coordinate a deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment. The instances in the environment each run the same set of containers, which are defined in a `Dockerrun.aws.json` file. Use the multicontainer configuration when you need to deploy multiple Docker containers to each instance.

For more details on the Multicontainer Docker configuration and its use, see Multicontainer Docker Environments (p. 547). The Multicontainer Docker Configuration (p. 547) topic details version 2 of the Dockerrun.aws.json format, which is similar to but not compatible with the version used with the single container configuration. There is also a tutorial (p. 551) available that guides you through a from scratch deployment of a multicontainer environment running a PHP website with an Nginx proxy running in front of it in a separate container.

Preconfigured Docker Containers

In addition to the two generic Docker configurations, there are several *preconfigured* Docker platform configurations that you can use to run your application in a popular software stack such as *Java with Glassfish* or *Python with uWSGI*. Use a preconfigured container if it matches the software used by your application.

For more information, see Preconfigured Docker Containers (p. 562).

Docker Images

The single container and multicontainer Docker configuration for Elastic Beanstalk support the use of Docker images stored in a public or private online image repository.

Specify images by name in `Dockerrun.aws.json`. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

For single container environments only, you can also build your own image during environment creation with a Dockerfile. See Building Custom Images with a Dockerfile (p. 544) for details.

Using Images in a Private Repository

To configure Elastic Beanstalk to authenticate to a private repository, include the `Authentication` (v1) or `authentication` (v2) parameter in your `Dockerrun.aws.json` file.

For details on the Dockerrun.aws.json format for single container environments, see Single Container Docker Configuration (p. 542). For multicontainer environments, see Multicontainer Docker Configuration (p. 547).

Topics

Single Container Docker Environments

Single Container Docker Configuration

This section describes how to prepare your Docker image and container for uploading to Elastic Beanstalk. Any web application that you deploy to Elastic Beanstalk in single-container Docker container must include a `Dockerfile`, which defines a custom image, a `Dockerrun.aws.json` file, which specifies an existing image to use and environment configuration, or both. You can deploy your web application from a Docker container to Elastic Beanstalk by doing one of the following:

- Create a `Dockerfile` to customize an image and to deploy a Docker container to Elastic Beanstalk.
- Create a `Dockerrun.aws.json` file to deploy a Docker container from an existing Docker image to Elastic Beanstalk.
- Create a `.zip` file containing your application files, any application file dependencies, the `Dockerfile`, and the `Dockerrun.aws.json` file.

> **Note**
> If you use only a `Dockerfile` or only a `Dockerrun.aws.json` file to deploy your application, you do not need to compress the file into a .zip file.

Topics

Dockerrun.aws.json v1

A `Dockerrun.aws.json` file describes how to deploy a Docker container as an Elastic Beanstalk application. This JSON file is specific to Elastic Beanstalk. If your application runs on an image that is available in a hosted repository, you can specify the image in a `Dockerrun.aws.json` file and omit the `Dockerfile`.

Valid keys and values for the `Dockerrun.aws.json` file include the following:

- **AWSEBDockerrunVersion** – (required) Specifies the version number as the value "1" for Single Container Docker environments
- **Authentication** – (required only for private repositories) Specifies the Amazon S3 object storing the `.dockercfg` file.

  See Using Images from a Private Repository (p. 544).
- **Image** – Specifies the Docker base image on an existing Docker repository from which you're building a Docker container. Specify the value of the **Name** key in the format *`<organization>/<image`*

`name>` for images on Docker Hub, or *`<site>/<organization name>/<image name>`* for other sites.

When you specify an image in the `Dockerrun.aws.json` file, each instance in your Elastic Beanstalk environment will run `docker pull` on that image and run it. Optionally include the **Update** key. The default value is "true" and instructs Elastic Beanstalk to check the repository, pull any updates to the image, and overwrite any cached images.

Do not specify the **Image** key in the `Dockerrun.aws.json` file when using a `Dockerfile`. .Elastic Beanstalk will always build and use the image described in the `Dockerfile` when one is present.

- **Ports** – (required when you specify the **Image** key) Lists the ports to expose on the Docker container. Elastic Beanstalk uses **ContainerPort** value to connect the Docker container to the reverse proxy running on the host.

  You can specify multiple container ports, but Elastic Beanstalk uses only the first one to connect your container to the host's reverse proxy and route requests from the public Internet.
- **Volumes** – Maps volumes from an EC2 instance to your Docker container. Specify one or more arrays of volumes to map.
- **Logging** – Maps the log directory inside the container.

  Configure Elastic Beanstalk to publish log files for the Docker container or to view snapshot logs. For more information, see Instance Logs (p. 282).

The following snippet is an example that illustrates the syntax of the `Dockerrun.aws.json` file for a single container.

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "janedoe/image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx"
}
```

You can provide Elastic Beanstalk with only the `Dockerrun.aws.json` file or in addition to the `Dockerfile` in a `.zip` file. When you provide both files, the `Dockerfile` builds the Docker image and the `Dockerrun.aws.json` file provides additional information for deployment as described later in this section. When you provide both the `Dockerfile` and the `Dockerrun.aws.json` file, do not specify an image in the `Dockerrun.aws.json` file. Elastic Beanstalk uses the image specified in the `Dockerfile` and ignores it in the `Dockerrun.aws.json` file.

Using Images from a Private Repository

To use a private repository hosted by a third-party registry, you must provide a JSON file called `.dockercfg` with information required to authenticate with the repository. Your authentication credentials are written to a `.dockercfg` file when you run the command `$ sudo docker login` to create an account on Docker Hub. (For a private repository, run the command `$ sudo docker login` *`server_name`*.)

Declare the `.dockercfg` file in the `Authentication` parameter of the `Dockerrun.aws.json` file. Make sure that the `Authentication` parameter contains a valid Amazon S3 bucket and key. The Amazon S3 bucket must be hosted in the same region as the environment that is using it. Elastic Beanstalk will not download files from Amazon S3 buckets hosted in other regions. Grant permissions for the action `s3:GetObject` to the IAM role in the instance profile. For an example policy, see Using IAM Roles with Elastic Beanstalk (p. 332).

For more information about the `.dockercfg` file, go to Working with Docker Hub on the Docker website.

The following example shows the use of a docker config in a bucket named `my-bucket` to use a private image in a third party registry.

```
{
  "AWSEBDockerrunVersion": "1",
  "Authentication": {
    "Bucket": "my-bucket",
    "Key": "mydockercfg"
  },
  "Image": {
    "Name": "quay.io/johndoe/private-image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx"
}
```

Building Custom Images with a Dockerfile

Docker uses a `Dockerfile` to create a Docker image that contains your source bundle. A Docker image is the template from which you create a Docker container. `Dockerfile` is a plain text file that contains instructions that Elastic Beanstalk uses to build a customized Docker image on each Amazon EC2 instance in your Elastic Beanstalk environment. Create a `Dockerfile` when you do not already have an existing image hosted in a repository.

Include the following instructions in the `Dockerfile`:

- **FROM** – (required as the first instruction in the file) Specifies the base image from which to build the Docker container and against which Elastic Beanstalk runs subsequent `Dockerfile` instructions.

The image can be hosted in a public repository, a private repository hosted by a third-party registry, or a repository that you run on EC2.

- **EXPOSE** – (required) Lists the ports to expose on the Docker container. Elastic Beanstalk uses the port value to connect the Docker container to the reverse proxy running on the host.

  You can specify multiple container ports, but Elastic Beanstalk uses only the first one to connect your container to the host's reverse proxy and route requests from the public Internet.
- **CMD** – Specifies an executable and default parameters, which are combined into the command that the container runs at launch. Use the following format:

  ```
  CMD ["executable","param1","param2"]
  ```

  `CMD` can also be used to provide default parameters for an `ENTRYPOINT` command by omitting the executable argument. An executable must be specified in either a `CMD` or an `ENTRYPOINT`, but not both. For basic scenarios, use a `CMD` and omit the `ENTRYPOINT`.
- **ENTRYPOINT** – Uses the same JSON format as `CMD` and, like `CMD`, specifies a command to run when the container is launched. Also allows a container to be run as an executable with `docker run`.

  If you define an `ENTRYPOINT`, you can use a CMD as well to specify default parameters that can be overridden with `docker run`'s `-d` option. The command defined by an `ENTRYPOINT` (including any parameters) is combined with parameters from `CMD` or `docker run` when the container is run.
- **RUN** – Specifies one or more commands that install packages and configure your web application inside the image.

  If you include RUN instructions in the `Dockerfile`, compress the file and the context used by RUN instructions in the `Dockerfile` into a `.zip` file. Compress files at the top level of the directory.

The following snippet is an example of the `Dockerfile`. When you follow the instructions in Single Container Docker Environments (p. 542), you can upload this `Dockerfile` as written. Elastic Beanstalk runs the game 2048 when you use this `Dockerfile`.

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/gab
rielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -
rf 2048-master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

For more information about instructions you can include in the `Dockerfile`, go to Dockerfile Reference on the Docker website.

Single container Docker environments can be launched from a Dockerfile (which describes an image to build), a Dockerrun.aws.json file (which specifies an image to use and additional Elastic Beanstalk configuration options), or both. These configuration files can be bundled with source code and deployed in a ZIP file.

Get started with one of the following example applications, or see Single Container Docker Configuration (p. 542) for details on authoring Docker configuration files for a single container environment.

To run a sample application using the AWS Management Console

1. Download a source bundle from the linked GitHub page by clicking **Download ZIP** on the right side of the screen.
2. Follow the instructions in Launching an Environment with a Sample Application in the AWS Management Console (p. 50).

To run a sample application using the EB CLI

1. Clone the GitHub repository or save the Dockerfile in a local project folder.
2. Run `eb init` in the project folder to configure the repository for use with the EB CLI.
3. Run `eb create` to create an environment and deploy the sample.

For detailed instructions on configuring and using the EB CLI, see Configure the EB CLI (p. 390) and Using the EB CLI (p. 394).

Sample PHP Application

GitHub link: awslabs/eb-demo-php-simple-app

This sample is a PHP application that runs on a custom Ubuntu image defined in a Dockerfile.

The PHP sample application uses Amazon RDS. You may be charged for using these services. If you are a new customer, you can make use of the AWS Free Usage Tier. For more information about pricing, see the following:

- Amazon Relational Database Service (RDS) Pricing

Sample Python Application

GitHub link: awslabs/eb-py-flask-signup

This sample is a Python application that runs on a custom Ubuntu image defined in a `Dockerfile`. It also includes a `Dockerrun.aws.json` file that maps a storage volume on the container to a matching path on the host instance.

The Python sample application uses Amazon DynamoDB, Amazon SQS, and Amazon SNS. You may be charged for using these services. If you are a new customer, you can make use of the AWS Free Usage Tier. For more information about pricing, see the following:

- Amazon DynamoDB Pricing
- Amazon SQS Pricing
- Amazon SNS Pricing

Sample Dockerfile Application

This sample is a `Dockerfile` configured to download the game 2048 from GitHub and run it on Nginx.

Copy and paste the example into a file named Dockerfile and upload it instead of a source bundle when creating the environment.

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/gab
rielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -
rf 2048-master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

Multicontainer Docker Environments

You can create docker environments that support multiple containers per instance with multicontainer Docker platform for Elastic Beanstalk.

Elastic Beanstalk uses Amazon EC2 Container Service to coordinate container deployments to multicontainer Docker environments. Amazon ECS provides tools to manage a cluster of instances running Docker containers. Elastic Beanstalk takes care of Amazon ECS tasks including cluster creation, task definition and execution.

Multicontainer Docker Configuration

A `Dockerrun.aws.json` file is an Elastic Beanstalk–specific JSON file that describes how to deploy a set of Docker containers as an Elastic Beanstalk application. You can use a `Dockerrun.aws.json` file for a multicontainer Docker environment.

`Dockerrun.aws.json` describes the containers to deploy to each container instance in the environment as well as the data volumes to create on the host instance for the containers to mount.

A `Dockerrun.aws.json` file can be used on its own or zipped up with additional source code in a single archive. Source code that is archived with a `Dockerrun.aws.json` is deployed to container instances and accessible in the `/var/app/current/` directory. Use the `volumes` section of the config to provide mount points for the containers running on the instance, and the `mountPoints` section of the embedded container definitions to mount them from the containers.

Topics

Dockerrun.aws.json v2

`Dockerrun.aws.json` file includes three sections:

- **AWSEBDockerrunVersion** – Specifies the version number as the value "2" for multicontainer Docker environments.
- **containerDefinitions** – An array of container definitions, detailed below.

- **volumes** – Creates mount points in the container instance that a container can use. Configure volumes for folders in your source bundle (deployed to `/var/app/current` on the container instance) for a container's application to read.

  Note
  Elastic Beanstalk configures additional volumes for logs, one for each container. These should be mounted by your containers in order to write logs to the host instance. See Container Definition Format (p. 550) for details.

  Volumes are specified in the following format:

```
"volumes": [
    {
      "name": "volumename",
      "host": {
        "sourcePath": "/path/on/host/instance"
      }
    }
  ],
```

- **authentication** (optional) – the location in Amazon S3 of a dockercfg file that contains authentication data for a private repository. Uses the following format:

```
"authentication": {
    "bucket": "my-bucket",
    "key": "mydockercfg"
  },
```

  See Using Images from a Private Repository (p. 549) for details.

The following snippet is an example that illustrates the syntax of the `Dockerrun.aws.json` file for an instance with two containers.

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "environment": [
        {
          "name": "Container",
          "value": "PHP"
        }
      ],
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

Using Images from a Private Repository

To use a private repository hosted by an online registry, you must provide a JSON file called `.dockercfg` with information required to authenticate with the registry. For Docker Hub, your authentication credentials are written to a `.dockercfg` file when you run the command `$ docker login` to create an account. For a third party registry, run the command `$ docker login` *`registry_server`*.)

Declare the `.dockercfg` file in the `authentication` parameter of the `Dockerrun.aws.json` file. Make sure that the `authentication` parameter contains a valid Amazon S3 bucket and key. The Amazon

S3 bucket must be hosted in the same region as the environment that is using it. Elastic Beanstalk will not download files from Amazon S3 buckets hosted in other regions. Grant permissions for the action `s3:GetObject` to the IAM role in the instance profile. For an example policy, see Using IAM Roles with Elastic Beanstalk (p. 332).

For more information about the `.dockercfg` file, see Working with Docker Hub and `docker login` on the Docker website

Container Definition Format

A `Dockerrun.aws.json` file contains an array of one or more container definition objects with the following fields:

name
The name of the container.

image
The name of a Docker image in an online Docker repository from which you're building a Docker container. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`.
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

environment
An array of environment variables to pass to the container.

For example, the following entry defines an environment variable with the name **`Container`** and the value **`PHP`**:

```
"environment": [
  {
    "name": "Container",
    "value": "PHP"
  }
],
```

essential
True if the task should stop if the container fails. Nonessential containers can finish or crash without affecting the rest of the containers on the instance.

memory
Amount of memory on the container instance to reserve for the container.

mountPoints
Volumes from the container instance to mount and the location on the container file system at which to mount them. Mount volumes containing application content so your container can read the data you upload in your source bundle, as well as log volumes for writing log data to a location where Elastic Beanstalk can gather it.

Elastic Beanstalk creates log volumes on the container instance, one for each container, at `/var/log/containers/`*`containername`*. These volumes are named `awseb-logs-`*`containername`* and should be mounted to the location within the container file structure where logs are written.

For example, the following mount point maps the Nginx log location in the container to the Elastic Beanstalk–generated volume for the `nginx-proxy` container.

```
{
  "sourceVolume": "awseb-logs-nginx-proxy",
  "containerPath": "/var/log/nginx"
}
```

portMappings
Maps network ports on the container to ports on the host.

links
List of containers to link to. Linked containers can discover each other and communicate securely.

Note
The container definition and volumes sections of `Dockerrun.aws.json` use the same formatting as the corresponding sections of an Amazon ECS task definition file.
The above examples show a subset of parameters that are commonly used. More optional parameters are available. For more information on the task definition format and a full list of task definition parameters, see Amazon ECS Task Definitions in the Amazon ECS Developer Guide.

Multicontainer Docker Environments with the AWS Management Console

You can launch a cluster of multicontainer instances in a single-instance or autoscaling Elastic Beanstalk environment using the AWS Management Console. This tutorial details IAM role creation, container configuration, and source code for an environment that uses two containers.

The containers, a PHP application and an Nginx proxy, run side by side on each of the Amazon EC2 instances in an Elastic Beanstalk environment. After creating the environment and verifying that the applications are running, you'll connect to a container instance to see how it all fits together.

Note
This tutorial was developed using Windows 7 and Google Chrome 41.

Topics

Configure a Container Instance IAM Role

Instances in a multicontainer Docker environment must have an instance profile with permission to access the Amazon EC2 Container Service, the service that Elastic Beanstalk uses to coordinate container deployments. To grant Amazon ECS access to the instances in your environment, first create an IAM policy and assign it to a role.

To create a container instance role in IAM

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Click **Policies**, click **Get Started** if the welcome page appears, and then click **Create Policy**.
3. For **Create Your Own Policy**, click **Select**.

4. Type **`BeanstalkECSAccess`** for the policy name and **`Allow container instances in an Elastic Beanstalk environment to run tasks from Amazon ECS and write logs to S3.`** for the description.
5. Copy and paste the following text into the policy document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:RegisterContainerInstance",
        "ecs:DeregisterContainerInstance",
        "ecs:DescribeContainerInstances",
        "ecs:DiscoverPollEndpoint",
        "ecs:Submit*",
        "ecs:Poll"
       ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::elasticbeanstalk-*/resources/environ
ments/logs/*"
    }
  ]
}
```

6. Click **Create Policy**

Next you will need a role to attach the policy to. If you've used Elastic Beanstalk before, you will already have a role named `aws-elasticbeanstalk-ec2-role`. If not, create a new one.

To create a new role

1. While still in the IAM console, click **Roles**.
2. Click **Create New Role**.
3. Type **`aws-elasticbeanstalk-ec2-role`** for the **Role Name** and click **Next Step**.
4. Under **AWS Service Roles** click **Select** next to **Amazon EC2**.
5. Click **Next Step**.
6. Click **Create Role**.

Finally, assign the policy to the role.

To assign the policy to a role

1. While still in the IAM console, click **Roles**.
2. Click **aws-elasticbeanstalk-ec2-role**.
3. Click **Attach Policy** under **Permissions**.
4. Select the **BeanstalkECSAccess** policy and click **Attach Policy**.

5. Click **Create Role**.

Define Docker Containers

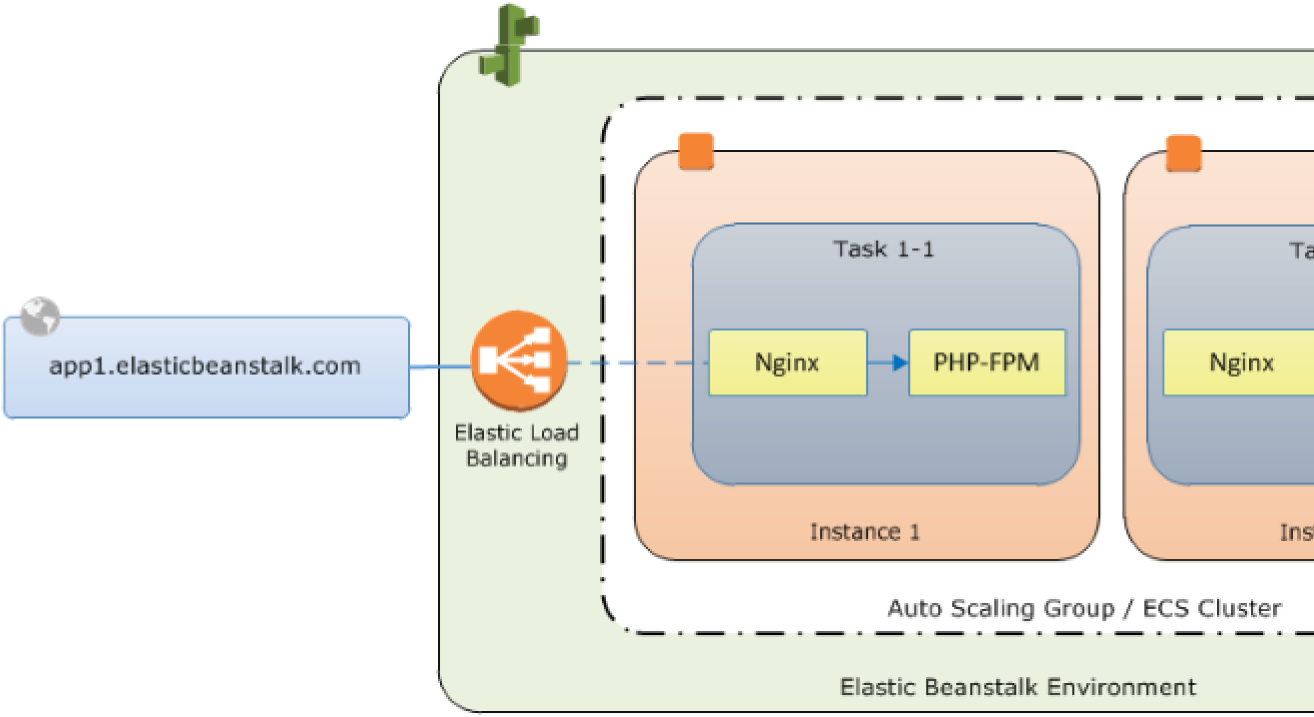
The first step in creating a new Docker environment is to create a directory for your application data. This folder can be located anywhere on your local machine and have any name you choose. In addition to a container configuration file, this folder will contain the content that you will upload to Elastic Beanstalk and deploy to your environment.

> **Note**
> All of the code for this tutorial is available in the awslabs repository on GitHub at https://github.com/awslabs/eb-docker-nginx-proxy

The file that Elastic Beanstalk uses to configure the containers on an EC2 instance is a JSON-formatted text file named `Dockerrun.aws.json`. Create a text file with this name at the root of your application and add the following text:

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

This example configuration defines two containers, a PHP web site with an Nginx proxy in front of it. These two containers will run side by side in Docker containers on each instance in your Elastic Beanstalk environment, accessing shared content (the content of the website) from volumes on the host instance, which are also defined in this file. The containers themselves are created from images hosted in official repositories on Docker Hub. The resulting environment looks like this:


app1.elasticbeanstalk.com
Elastic Load Balancing
Task 1-1
Nginx
PHP-FPM
Instance 1
Ta
Nginx
Ins
Auto Scaling Group / ECS Cluster
Elastic Beanstalk Environment

The volumes defined in the configuration correspond to the content that you will create next and upload as part of your application source bundle. The containers access content on the host by mounting volumes in the `mountPoints` section of the container definitions.

For more information on the format of `Dockerrun.aws.config` and its parameters, see Container Definition Format (p. 550).

Add Content

Next you will add some content for your PHP site to display to visitors, and a configuration file for the Nginx proxy.

php-app\index.php

```
<h1>Hello World!!!</h1>
<h3>PHP Version <pre><?= phpversion()?></pre></h3>
```

php-app\static.html

```
<h1>Hello World!</h1>
<h3>This is a static HTML page.</h3>
```

proxy\conf.d\default.conf

```
server {
  listen 80;
  server_name localhost;
  root /var/www/html;

  index index.php;

  location ~ [^/]\.php(/|$) {
    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
    if (!-f $document_root$fastcgi_script_name) {
      return 404;
    }

    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;

    fastcgi_pass php-app:9000;
    fastcgi_index index.php;
  }
}
```

Deploy to Elastic Beanstalk

Your application folder now contains the following files:

```
php-app\index.php
php-app\static.html
```

```
proxy\conf.d\default.conf
Dockerrun.aws.json
```

This is all you need to create the Elastic Beanstalk environment. Create a `.zip` archive of the above files and folders (not including the top-level project folder). To create the archive in Windows explorer, select the contents of the project folder, right-click, select **Send To**, and then click **Compressed (zipped) Folder**

> **Note**
> For information on the required file structure and instructions for creating archives in other environments, see Create an Application Source Bundle (p. 43)

Next, you'll upload the source bundle to Elastic Beanstalk and create your environment.

To create a multicontainer Elastic Beanstalk environment

1. Sign in to the AWS Management Console and open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Click **Create New Application**.
3. Enter a name and description and click **Next**.
4. Click **Create web server**.
5. Select `aws-elasticbeanstalk-ec2-role` and click **Next**.
6. Set **Predefined configuration** to **Multi-container Docker**.
7. Set **Environment type** to **Single instance** and click **Next**.
8. For **Application Version**, select **Upload your own**.
9. Click **Browse**, select the `.zip` archive and click **Open**. Click **Next**
10. Configure your environment's name and prefix and click **Next**.
11. On the **Configuration Details** page, select a key pair to enable SSH access to the instances in your environment. Keep the default values for everything else and click **Next**.

    > **Note**
    > If you haven't used key pairs before, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*. You can also skip key pair assignment and complete this tutorial without connecting to an instance.

12. Review the information on the final page and click **Launch**.

The AWS Management Console redirects you to the management dashboard for your new environment. This screen shows the health status of the environment and events output by the Elastic Beanstalk service. When the status is Green, click the URL next to the environment name to see your new website.

Connect to a Container Instance

So how does it all work? Next you will connect to an EC2 instance in your Elastic Beanstalk environment to see some of the moving parts in action.

First, identify the instance and note its public IP address, which is available in the Amazon EC2 console at https://console.aws.amazon.com/ec2/. If multiple instances are running and you have trouble identifying the one the belongs to your environment, read through the events on the environment dashboard and find the instance ID. This ID appears in an event listing when Elastic Beanstalk launches an EC2 instance. Search for the instance ID in the Amazon EC2 console and view its details to find the public IP address.

Next, use an SSH client and your private key file to connect to the instance. Use the following settings:

SSH Settings

- **Address** – The public IP address or DNS name of the EC2 instance.
- **Port** – **22**. This port is opened for ingress by Elastic BeanstalkBeanstalk when you select an Amazon EC2 key pair during environment configuration.
- **User Name** – **`ec2-user`**. This is the default user name for EC2 instances running Amazon Linux.
- **Private Key** – Your private key file.

For full instructions on using SSH to connect to an EC2 instance, see Connecting to Your Linux Instance Using SSH in the *Amazon EC2 User Guide for Linux Instances*.

Now that your connected to the EC2 instance hosting your docker containers, you can see how things are set up. Run `ls` on `/var/app/current`:

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/app/current
Dockerrun.aws.json  php-app  proxy
```

This directory contains the files from the source bundle that you uploaded to Elastic Beanstalk during environment creation.

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/log/containers
nginx                          nginx-proxy-ffffd873ada5-stdouterr.log  rotated
nginx-66a4fd37eb63-stdouterr.log  php-app
nginx-proxy                       php-app-b894601a1364-stdouterr.log
```

This is where logs are created on the container instance and collected by Elastic Beanstalk. Elastic Beanstalk creates a volume in this directory for each container, which you mount to the container location where logs are written.

You can also take a look at Docker to see the running containers with `docker ps`.

```
[ec2-user@ip-10-0-0-117 ~]$ sudo docker ps
CONTAINER ID        IMAGE                            COMMAND
CREATED             STATUS              PORTS                            NAMES
ffffd873ada5        nginx:1.7                        "nginx -g 'daemon of
About an hour ago   Up About an hour    443/tcp, 0.0.0.0:80->80/tcp   ecs-eb-
dv-example-env-ycmk5geqrm-2-nginx-proxy-90fce996cc8cbecb2800
b894601a1364        php:5-fpm                        "php-fpm"
About an hour ago   Up About an hour    9000/tcp                         ecs-eb-
dv-example-env-ycmk5geqrm-2-php-app-cec0918ed1a3a49a8001
09fb19828e38        amazon/amazon-ecs-agent:latest   "/agent"
About an hour ago  Up About an hour    127.0.0.1:51678->51678/tcp    ecs-agent
```

This shows the two running containers that you deployed, as well as the Amazon ECS container agent that coordinated the deployment.

Inspect the Amazon ECS Container Agent

EC2 instances in a Multicontainer Docker environment on Elastic Beanstalk run an agent process in a Docker container. This agent connects to the Amazon ECS service in order to coordinate container deployments. These deployments run as tasks in Amazon ECS, which are configured in task definition files. Elastic Beanstalk creates these task definition files based on the `Dockerrun.aws.json` that you upload in a source bundle.

Check the status of the container agent with an HTTP get request to `http://localhost:51678/v1/metadata`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster":"eb-dv-example-env-qpoxiguye24",
  "ContainerInstanceArn":"arn:aws:ecs:us-east-1:123456789012:container-in
stance/6a72af64-2838-400d-be09-3ab2d836ebcd"
}
```

This structure shows the name of the Amazon ECS cluster, and the ARN (Amazon Resource Name) of the cluster instance (the EC2 instance that you are connected to).

For more information, make an HTTP get request to information is available at `http://localhost:51678/v1/tasks`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/tasks
{
  "Tasks":[
    {
      "Arn":"arn:aws:ecs:us-east-1:123456789012:task/3ff2bf0f-790d-4f6d-affb-
5b127b3b6e4a",
      "DesiredStatus":"RUNNING",
      "KnownStatus":"RUNNING",
      "Family":"eb-dv-example-env-qpoxiguye24",
      "Version":"2",
      "Containers":[
        {
          "Dock
erId":"b894601a1364a438156a239813c77cdef17040785bc4d5e49349470dc1556b15",
          "DockerName":"ecs-eb-dv-exampl e-env-qpoxiguye24-2-php-app-
cec0918ed1a3a49a8001",
          "Name":"php-app"
        },
        {
          "Dock
erId":"ffffd873ada5f537c88862cce4e1de7ec3edf962645982fb236961c833a5d0fe",
          "DockerName":"ecs-eb-dv-example-env-qpoxiguye24-2-nginx-proxy-
90fce996cc8cbecb2800",
          "Name":"nginx-proxy"
        }
      ]
    }
  ]
}
```

This structure describes the task that is run to deploy the two docker containers from this turorial's example project. The following information is displayed:

- **KnownStatus** – The `RUNNING` status indicates that the containers are still active.
- **Family** – The name of the task definition that Elastic Beanstalk created from `Dockerrun.aws.json`.
- **Version** – The version of the task definition. This is incremented each time the task definition file is updated.
- **Containers** – Information about the containers running on the instance.

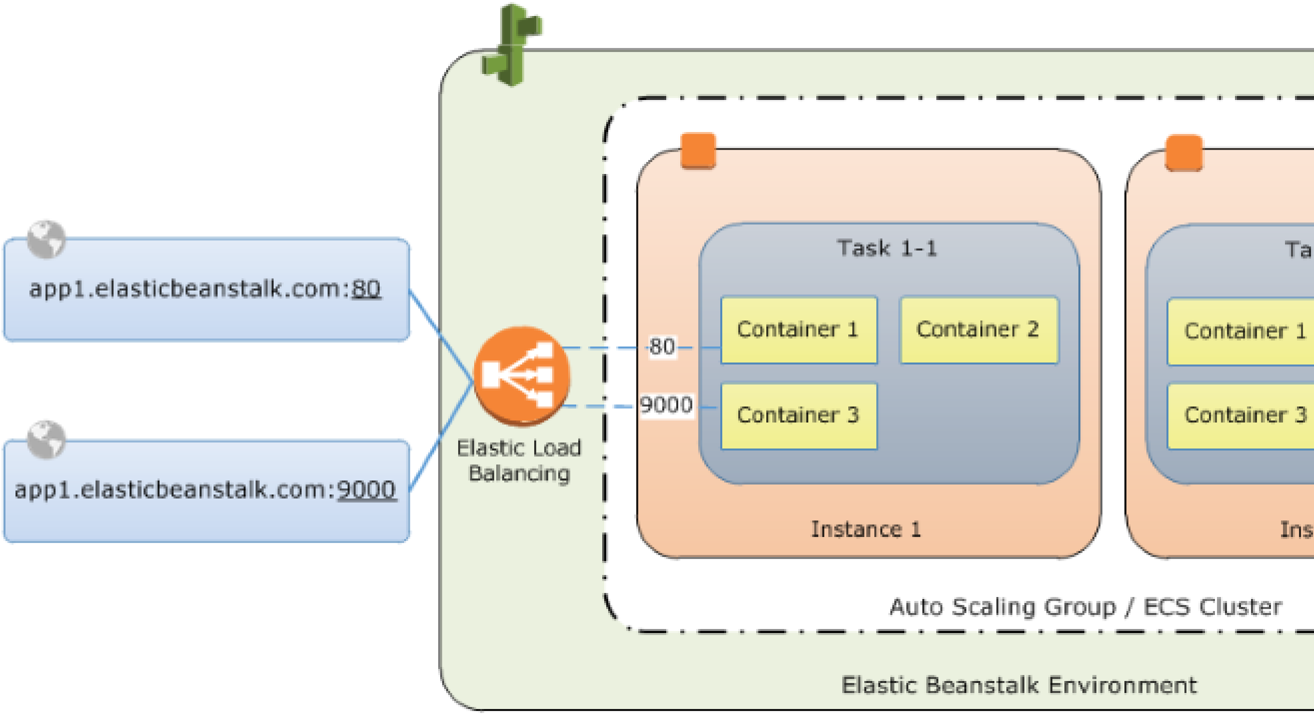
Even more information is available from the Amazon ECS service itself, which you can call using the AWS Command Line Interface. For instructions on using the AWS CLI with Amazon ECS, and information about Amazon ECS in general, see the Amazon ECS User Guide.

Topics

Multicontainer Docker Platform

Standard generic and preconfigured Docker platforms on Elastic Beanstalk support only a single Docker container per Elastic Beanstalk environment. In order to get the most out of Docker, Elastic Beanstalk lets you create an environment where your instances run multiple Docker containers side by side.

The following diagram shows an example Elastic Beanstalk environment configured with three Docker containers running on each EC2 instance in an Auto Scaling group:


app1.elasticbeanstalk.com:80
app1.elasticbeanstalk.com:9000
Elastic Load Balancing
80
9000
Task 1-1
Container 1
Container 2
Container 3
Instance 1
Ta
Container 1
Container 3
Ins
Auto Scaling Group / ECS Cluster
Elastic Beanstalk Environment

Dockerrun.aws.json File

Multicontainer Docker instances on Elastic Beanstalk require a configuration file named `Dockerrun.aws.json`. This file is specific to Elastic Beanstalk and can be used alone or combined with source code and content in a source bundle (p. 43) to create an environment on a Docker platform.

Note
Version 1 of the `Dockerrun.aws.json` format is used to launch a single Docker container to an Elastic Beanstalk environment. Version 2 adds support for multiple containers per instance and can only be used with the multicontainer Docker platform. The format differs significantly from the previous version which is detailed under Single Container Docker Configuration (p. 542)

See Dockerrun.aws.json v2 (p. 547) for details on the updated format and an example file.

Docker Images

The Multicontainer Docker platform for Elastic Beanstalk requires images to be prebuilt and stored in a public or private online image repository.

Note
Building custom images during deployment with a `Dockerfile` is not supported by the multicontainer Docker platform on Elastic Beanstalk. Build your images and deploy them to an online repository before creating an Elastic Beanstalk environment.

Specify images by name in `Dockerrun.aws.json`. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online registries are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

To configure Elastic Beanstalk to authenticate to a private repository, include the `authentication` parameter in your `Dockerrun.aws.json` file.

Container Instance Role

Elastic Beanstalk uses an Amazon ECS-optimized AMI with an Amazon ECS container agent that runs in a Docker container. The agent communicates with Amazon ECS to coordinate container deployments. In order to communicate with Amazon ECS, each instance must have the corresponding permissions in IAM. The following IAM role provides an example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:RegisterContainerInstance",
        "ecs:DeregisterContainerInstance",
        "ecs:DescribeContainerInstances",
        "ecs:DiscoverPollEndpoint",
        "ecs:Submit*",
        "ecs:Poll"
       ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
     "Resource": "arn:aws:s3:::elasticbeanstalk-*/resources/environments/logs/*"

    }
  ]
}
```

For instructions on creating policies and roles in IAM, see Creating IAM Roles in the IAM User Guide.

> **Note**
> If you use a role with permissions configured for worker environment tiers, add the container instance permissions to your existing role policy and use that. Worker environment permissions are detailed under IAM Roles for Elastic Beanstalk Environment Tiers (p. 381).

Amazon ECS Resources Created by Elastic Beanstalk

When you create an environment using the multicontainer Docker platform, Elastic Beanstalk automatically creates and configures several Amazon EC2 Container Service resources while building the environment in order to create the necessary containers on each EC2 instance.

- **Amazon ECS Cluster** – Container instances in Amazon ECS are organized into clusters. When used with Elastic Beanstalk, one cluster is always created for each multicontainer Docker environment.
- **Amazon ECS Task Definition** – Elastic Beanstalk uses the `Dockerrun.aws.json` file in your project to generate the Amazon ECS task definition that is used to configure container instances in the environment.
- **Amazon ECS Task** – Elastic Beanstalk communicates with Amazon ECS to run a task on every instance in the environment to coordinate container deployment. In an autoscaling environment, Elastic Beanstalk initiates a new task whenever an instance is added to the cluster.
- **Amazon ECS Container Agent** – The agent runs in a Docker container on the instances in your environment. The agent polls the Amazon ECS service and waits for a task to run.
- **Amazon ECS Data Volumes** – Elastic Beanstalk inserts volume definitions (in addition to the volumes that you define in `Dockerrun.aws.json`) into the task definition to facilitate log collection.

  Elastic Beanstalk creates log volumes on the container instance, one for each container, at `/var/log/containers/`*`containername`*. These volumes are named `awseb-logs-`*`containername`* and are provided for containers to mount. See Container Definition Format (p. 550) for details on how to mount them.

Using Multiple Elastic Load Balancing Listeners

You can configure multiple Elastic Load Balancing listeners on a multicontainer Docker environment in order to support inbound traffic for proxies or other services that don't run on the default HTTP port.

Create a `.ebextensions` folder in your source bundle and add a file with a `.config` file extension. The following example shows a configuration file that creates an Elastic Load Balancing listener on port 8080 and opens the same port for inbound traffic on the Elastic Beanstalk environment's security group.

.ebextensions/elb-listeners.config

```
option_settings:
  aws:elb:listener:8080:
    ListenerProtocol: HTTP
    InstanceProtocol: HTTP
    InstancePort: 8080

Resources:
  port8080SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupName: {Ref : AWSEBSecurityGroup}
      IpProtocol: tcp
      ToPort: 8080
      FromPort: 8080
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
"SourceSecurityGroup.GroupName"] }
      SourceSecurityGroupOwnerId: { "Fn::GetAtt": ["AWSEBLoadBalancer",
"SourceSecurityGroup.OwnerAlias"] }
```

For more information on the configuration file format, see Customizing Environment Resources (p. 127) and Option_settings (p. 156)

In addition to adding a listener to the Elastic Load Balancing configuration and opening a port in the security group, you need to map the port on the host instance to a port on the Docker container in the `containerDefinitions` section of the `Dockerrun.aws.json` file. The following excerpt shows an example:

```
"portMappings": [
  {
    "hostPort": 8080,
    "containerPort": 8080
  }
]
```

See Dockerrun.aws.json v2 (p. 547) for details on the `Dockerrun.aws.json` file format.

Failed Container Deployments

If an Amazon ECS task fails, one or more containers in your Elastic Beanstalk environment will not start. Elastic Beanstalk does not roll back multicontainer environments due to a failed Amazon ECS task. If a container fails to start in your environment, redeploy the current version or a previous working version from the AWS Management Console.

To deploy an existing version

1. Open the Elastic Beanstalk console in your environment's region.
2. Click **Actions** to the right of your application name and then click **View Application Versions**.
3. Select a version of your application and click **Deploy**.

Preconfigured Docker Containers

Topics

Elastic Beanstalk supports Docker containers that are based on the language stacks provided in the Docker Official Repositories. You can use preconfigured Docker containers to develop and test your application locally and then deploy the application in an Elastic Beanstalk environment that is identical to your local environment.

For an end-to-end walkthrough about deploying an application to Elastic Beanstalk using a preconfigured Docker container, see Getting Started with Preconfigured Docker Containers (p. 563).

For more information about supported platforms for preconfigured Docker containers, see Preconfigured Docker (p. 25).

Getting Started with Preconfigured Docker Containers

This section walks you through how to develop a sample application locally and then deploy your application to Elastic Beanstalk with a preconfigured Docker container.

Set Up Your Local Development Environment

For this walkthrough we will use a Python Flask “Hello World” application.

To set up your develop environment

1. Create a new folder for the sample application.

```
$ mkdir eb-flask-sample
$ cd eb-flask-sample
```

2. In the application's root folder, create an `application.py` file. In the file, include the following:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

3. In the application's root folder, create a `requirements.txt` file. In the file, include the following:

```
flask
```

Develop and Test Locally

To develop a sample Python Flask application

1. Add a `Dockerfile` to your application’s root folder. In the file, specify the AWS Elastic Beanstalk Docker base image to be used to run your local preconfigured Docker container and against which Elastic Beanstalk runs any subsequent `Dockerfile` instructions by including the following:

```
# For Python 3.4
FROM amazon/aws-eb-python:3.4.2-onbuild-3.5.1
```

AWS Elastic Beanstalk also supports Docker images for Glassfish 4.1 Java 8 and Glassfish 4.0 Java 7. For their Docker image names, see Supported Platforms (p. 24). For more information about using a `Dockerfile`, see Single Container Docker Configuration (p. 542).

2. Build the Docker image.

```
$ docker build –t my-app-image .
```

3. Run the Docker container from the image.

   Note
   You must include the `-p` flag to map port 8080 on the container to the localhost port 3000. Elastic Beanstalk Docker containers always expose the application on port 8080 on the container. The `-it` flag runs the image as an interactive process. The `-rm` flag cleans up the container file system when the container exits. You can optionally include the `-d` flag to run the image as a daemon.

```
$ docker run –it --rm -p 3000:8080 my-app-image
```

4. To view the sample application, type the following URL into your web browser.

```
http://localhost:3000
```

Deploy to Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk.

To deploy your application to Elastic Beanstalk

1. In your application's root folder, rename the `Dockerfile` to `Dockerfile.local`. This step is required for Elastic Beanstalk to use the `Dockerfile` that contains the correct instructions for Elastic Beanstalk to build a customized Docker image on each Amazon EC2 instance in your Elastic Beanstalk environment.
2. Create an application source bundle. For more information, see Create an Application Source Bundle (p. 43).
3. To create a new Elastic Beanstalk application to which you can deploy your application, see Create an Application (p. 34). At the appropriate step, on the **Environment Type** page, in the **Predefined configuration** list, under **Preconfigured - Docker**, click **Python**.

Example: Using a Dockerfile to Customize and Configure a Preconfigured Docker Platform

With preconfigured Docker platforms you cannot use a configuration file to customize and configure the software that your application depends on. Instead, if you want to customize the preconfigured Docker platform to install additional software packages that your application needs, you can add a `Dockerfile` to your application's root folder.

You can include the following instructions in the `Dockerfile`:

- **FROM** – (required as the first instruction in the file) Specifies the base image from which to build the Docker container and against which Elastic Beanstalk runs subsequent `Dockerfile` instructions.

  The image can be hosted in a public repository, a private repository hosted by a third-party registry, or a repository that you run on EC2.
- **EXPOSE** – (required) Lists the ports to expose on the Docker container. Elastic Beanstalk uses the port value to connect the Docker container to the reverse proxy running on the host.

  You can specify multiple container ports, but Elastic Beanstalk uses only the first one to connect your container to the host's reverse proxy and route requests from the public Internet.
- **CMD** – Specifies an executable and default parameters, which are combined into the command that the container runs at launch. Use the following format:

  ```
  CMD ["executable","param1","param2"]
  ```

  `CMD` can also be used to provide default parameters for an `ENTRYPOINT` command by omitting the executable argument. An executable must be specified in either a `CMD` or an `ENTRYPOINT`, but not both. For basic scenarios, use a `CMD` and omit the `ENTRYPOINT`.
- **ENTRYPOINT** – Uses the same JSON format as `CMD` and, like `CMD`, specifies a command to run when the container is launched. Also allows a container to be run as an executable with `docker run`.

  If you define an `ENTRYPOINT`, you can use a CMD as well to specify default parameters that can be overridden with `docker run`'s `-d` option. The command defined by an `ENTRYPOINT` (including any parameters) is combined with parameters from `CMD` or `docker run` when the container is run.
- **RUN** – Specifies one or more commands that install packages and configure your web application inside the image.

  If you include RUN instructions in the `Dockerfile`, compress the file and the context used by RUN instructions in the `Dockerfile` into a `.zip` file. Compress files at the top level of the directory.

For more information about instructions you can include in the `Dockerfile`, go to Dockerfile Reference on the Docker website.

The following snippet is an example of a `Dockerfile`. The instructions in the `Dockerfile` customize the Python 3.4 platform by adding PostgreSQL dependencies and expose port 8080.

> **Note**
> Elastic Beanstalk preconfigured Docker platforms for Glassfish and Python require you to expose port 8080. Elastic Beanstalk preconfigured Docker platforms for Go require you to expose port 3000.

```
# Use the AWS Elastic Beanstalk Python 3.4 image
FROM amazon/aws-eb-python:3.4.2-onbuild-3.5.1

# Exposes port 8080
EXPOSE 8080

# Install PostgreSQL dependencies
RUN apt-get update && \
    apt-get install -y postgresql libpq-dev && \
    rm -rf /var/lib/apt/lists/*
```

If you want to use additional AWS resources (such as Amazon DynamoDB or Amazon Simple Notification Service), modify the proxy server or modify the operating system configuration for your Elastic Beanstalk environment. For more information about using configuration files, see Elastic Beanstalk Environment Configuration (p. 99).

Configuring Docker Environments

AWS Management Console

To access the Docker configuration for your Elastic Beanstalk environment

1. Navigate to the management console (p. 51) for your environment.
2. In the **Overview** section of the environment dashboard, click **Edit**.
3. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

Log Options

The Log Options section has two settings:

- **Instance profile**–Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file Amazon EC2 instances** should be copied to the Amazon S3 bucket associated with your application.

Amazon S3 Log Rotation

Elastic Beanstalk can copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application on an hourly basis. To enable this feature, select **Enable log file rotation to Amazon S3**.

Environment Properties

The **Environment Properties** section lets you specify application settings. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.

You can configure the following application settings:

- Specify AWS credentials using the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_KEY** text boxes.

  Note
  Use instance profiles so that your application can use temporary security credentials to access AWS resources. To learn more, see Service Roles, Instance Profiles, and User Policies (p. 19)

- Specify up to five additional key-value pairs by entering them in the **PARAM** boxes.

  Note
  These settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Command Line Interface CLI)

To edit your Elastic Beanstalk application's environment settings container/Docker options

- Update an application's environment settings.

  ```
  $ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
  ```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_ACCESS_KEY_ID",
        "Value": "AKIAIOSFODNN7EXAMPLE"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_SECRET_KEY",
        "Value": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "myvar",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To access the Container/Docker Options panel for your Elastic Beanstalk application

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.1.OptionName`* = `AWS_ACCESS_KEY_ID`
  - *`OptionSettings.member.1.Value`* = `AKIAIOSFODNN7EXAMPLE`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.2.OptionName`* = `AWS_SECRET_KEY`
  - *`OptionSettings.member.2.Value`* = `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
  - *`OptionSettings.member.3.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.3.OptionName`* = `myvar`
  - *`OptionSettings.member.3.Value`* = `somevalue`
  - *`OptionSettings.member.4.Namespace`* = `aws:elasticbeanstalk:hostmanager`

- *OptionSettings.member.4.OptionName* = `LogPublicationControl`
- *OptionSettings.member.4.Value* = `false`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=MySampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.1.OptionName=AWS_ACCESS_KEY_ID
&OptionSettings.member.1.Value=AKIAIOSFODNN7EXAMPLE
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.2.OptionName=AWS_SECRET_KEY
&OptionSettings.member.2.Value=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.3.OptionName=myvar
&OptionSettings.member.3.Value=somevalue
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.4.OptionName=LogPublicationControl
&OptionSettings.member.4.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Running a Docker Environment Locally with the EB CLI

You can use the EB Command Line Interface (EB CLI) to run the Docker container(s) configured in your AWS Elastic Beanstalk application locally. The EB CLI uses the Docker configuration file (Dockerfile or Dockerrun.aws.json) and source code in your project directory to run your application locally in Docker.

The EB CLI supports running single container, multicontainer, and preconfigured container applications locally.

Topics

Prerequisites for Running Docker Applications Locally

- Linux OS or Mac OS X
- EB CLI version 3.3 or greater (p. 385)

  Run `eb init` in your project directory to initialize an EB CLI repository. If you haven't used the EB CLI before, see Using the EB CLI (p. 394).

- Docker version 1.6 or greater

  Add yourself to the docker group, log out, and then log back in to ensure that you can run Docker commands without sudo:

  ```
  $ sudo usermod -a -G docker $USER
  ```

  Run docker ps to verify that the Docker daemon is up and running:

  ```
  $ docker ps
  CONTAINER ID        IMAGE               COMMAND             CREATED
      STATUS              PORTS               NAMES
  ```

- A Docker application

  If you don't have a Docker application in a project folder on your local machine, see Deploying Elastic Beanstalk Applications from Docker Containers (p. 540) for an introduction to using Docker with AWS Elastic Beanstalk.
- Docker profile (optional)

  If your application uses Docker images that are in a private repository, run docker login and follow the prompts to create an authentication profile.
- w3m (optional)

  W3m is a web browser that you can use to view your running web application within a command line terminal with eb local run. If you are using the command line in a desktop environment, you don't need w3m.

Docker containers run locally without emulating AWS resources that are provisioned when you deploy an application to Elastic Beanstalk, including security groups and data or worker tiers.

You can configure your local containers to connect to a database by passing the necessary connection string or other variables with the envvars option, but you must ensure that any resources in AWS are accessible from your local machine by opening the appropriate ports in their assigned security groups or attaching a default gateway or elastic IP address.

Preparing a Docker Application for Use with the EB CLI

Prepare your Docker configuration file and source data as though you were deploying them to Elastic Beanstalk. This topic uses the PHP and Nginx proxy example from the Multicontainer Docker tutorial (p. 551) earlier in this guide as an example, but you can use the same commands with any single container, multicontainer, or preconfigured Docker application.

Running a Docker Application Locally

Run your Docker application locally with the eb local run command from within the project directory:

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
```

```
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

The EB CLI reads the Docker configuration and executes the Docker commands necessary to run your application. The first time you run a project locally, Docker downloads images from a remote repository and stores them on your local machine. This process can take several minutes.

> **Note**
>
> The `eb local run` command takes two optional parameters, `port` and `envvars`.
> To override the default port for a single container application, use the `port` option:
>
> ```
> $ eb local run --port 8080
> ```
>
> This command tells the EB CLI to use port 8080 on the host and map it to the exposed port on the container. If you don't specify a port, the EB CLI uses the container's port for the host. This option only works with single container applications
> To pass environment variables to the application containers, use the `envvars` option:
>
> ```
> $ eb local run --envvars
> RDS_HOST=$RDS_HOST,RDS_DB=$RDS_DB,RDS_USER=$RDS_USER,RDS_PASS=$RDS_PASS
> ```
>
> Use environment variables to configure a database connection, set debug options, or pass secrets securely to your application. For more information on the options supported by the `eb local` subcommands, see local (p. 425).

After the containers are up and running in Docker, they are ready to take requests from clients. The `eb local` process stays open as long as the containers are running. If you need to stop the process and containers, press **Ctrl**+**C** .

Open a second terminal to run additional commands while the `eb local` process is running. Use `eb local status` to view your application's status:

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3
 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): None
Full local URL(s): None
```

You can use `docker ps` to see the status of the containers from Docker's point of view:

```
~/project$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                              NAMES
6a8e71274fed        nginx:latest        "nginx -g 'daemon of     9 minutes ago
```

```
     Up 9 minutes        0.0.0.0:80->80/tcp, 443/tcp   elasticbean
stalk_nginxproxy_1
82cbf620bdc1        php:fpm             "php-fpm"              9 minutes ago
     Up 9 minutes       9000/tcp                     elasticbeanstalk_phpapp_1
```

Next, view your application in action with `eb local open`:

```
~/project$ eb local open
```

This command opens your application in the default web browser. If you are running a terminal in a desktop environment, this may be Firefox, Safari, or Google Chrome. If you are running a terminal in a headless environment or over an SSH connection, a command line browser, such as w3m, will be used if one is available.

Switch back to the terminal running the application process for a moment and note the additional output:

```
phpapp_1     | 172.17.0.36 -  21/Apr/2015:23:46:17 +0000 "GET /index.php" 200
```

This shows that the web application in the Docker container received an HTTP GET request for index.php that was returned successfully with a 200 (non error) status.

Run `eb local logs` to see where the EB CLI writes the logs.

```
~/project$ eb local logs
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbean
stalk/logs/local.
Logs were most recently created 3 minutes ago and written to /home/user/pro
ject/.elasticbeanstalk/logs/local/150420_234011665784.
```

Cleaning Up After Running a Docker Application Locally

When you are done testing your application locally, you can stop the applications and remove the images downloaded by Docker when you use `eb local run`. Removing the images is optional. You may want to keep them for future use.

Return to the terminal running the `eb local` process and press **Ctrl**+**C** to stop the application:

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping elasticbeanstalk_nginxproxy_1...
Stopping elasticbeanstalk_phpapp_1...

Aborting.
[1]+  Exit 5                  eb local run
```

The EB CLI attempts to stop each running container gracefully with Docker commands. If you need to stop a process immediately, press **Ctrl**+**C** again.

After you stop the applications, the Docker containers should also stop running. Verify this with `docker ps`:

```
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND                CREATED
     STATUS                     PORTS               NAMES
73d515d99d2a        nginx:latest        "nginx -g 'daemon of   21 minutes ago
     Exited (0) 11 minutes ago                      elasticbean
stalk_nginxproxy_1
7061c76220de        php:fpm             "php-fpm"              21 minutes ago
     Exited (0) 11 minutes ago                      elasticbeanstalk_phpapp_1
```

The `all` option shows stopped containers (if you omitted this option, the output will be blank). In the above example, Docker shows that both containers exited with a 0 (non-error) status.

If you are done using Docker and EB CLI local commands, you can remove the Docker images from your local machine to save space.

To remove Docker images from your local machine

1. View the images that you downloaded using `docker images`:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
      VIRTUAL SIZE
php                 fpm                 68bc5150cffc        1 hour ago
      414.1 MB
nginx               latest              637d3b2f5fb5        1 hour ago
      93.44 MB
```

2. Remove the two Docker containers with `docker rm`:

```
$ docker rm 73d515d99d2a 7061c76220de
73d515d99d2a
7061c76220de
```

3. Remove the images with `docker rmi`:

```
$ docker rmi 68bc5150cffc 637d3b2f5fb5
Untagged: php:fpm
Deleted: 68bc5150cffc0526c66b92265c3ed8f2ea50f3c71d266aa655b7a4d20c3587b0
Untagged: nginx:latest
Deleted: 637d3b2f5fb5c4f70895b77a9e76751a6e7670f4ef27a159dad49235f4fe61e0
```

Deploying Go Applications to Elastic Beanstalk Applications

Elastic Beanstalk supports applications that are developed using the Go programming language (sometimes called Golang). Elastic Beanstalk supports Docker containers that are based on the language stacks provided in the Docker Official Repositories. You can use Elastic Beanstalk preconfigured Docker containers to develop and test your Go application locally and then deploy the application in an Elastic Beanstalk environment that is identical to your local environment.

For an end-to-end walkthrough about deploying a Go application to Elastic Beanstalk using a preconfigured Docker container, see Getting Started with Go Preconfigured Docker Containers (p. 573).

For more information about supported platforms for preconfigured Docker containers, including the containers for Go applications, see Preconfigured Docker (p. 25).

Getting Started with Go Preconfigured Docker Containers

Follow the steps here to walk you through the process of deploying a Go application to Elastic Beanstalk with a preconfigured Docker container for Go.

Set Up Your Local Development Environment

This tutorial uses a Go “Hello World” application.

To set up your develop environment

1. Create a new folder for the sample application.

```
$ mkdir eb-go-sample
$ cd eb-go-sample
```

2. In the application's root folder, create a file with the name `server.go`. In the file, include the following:

```
package main

import "github.com/go-martini/martini"

func main() {
    m := martini.Classic()
    m.Get("/", func() string {
        return "Hello world!"
    })
    m.Run()
}
```

Note

- The application source bundle must include a package called `main`. Within that package, you must have a `main` function for the container to execute.
- Dependencies that need to be imported (for example, the Martini package, `go-martini`) will be downloaded to the container and installed during deployment. Therefore, you do not need to include the dependencies in the application source bundle that you upload to Elastic Beanstalk.
- Elastic Beanstalk sets the container's GOPATH environment variable to `/go`.

Develop and Test Locally Using Docker

With your environment set up, you're ready to create and test your Go application.

To develop a sample Go application

1. Add a `Dockerfile` to your application's root folder. In the file, specify the Elastic Beanstalk Docker base image to use to run your local preconfigured Docker container. Elastic Beanstalk uses this image to run any subsequent `Dockerfile` instructions.

   Note
   Include only the instruction with the Docker image name for your platform version. For preconfigured Docker image names, see Supported Platforms (p. 24). For more information about using a `Dockerfile`, see Single Container Docker Configuration (p. 542). For an example `Dockerfile` for preconfigured Docker platforms, see Example: Using a Dockerfile to Customize and Configure a Preconfigured Docker Platform (p. 564).

   You can use the following example:

```
# For Go 1.3
FROM golang:1.3.3-onbuild

# For Go 1.4
FROM golang:1.4.1-onbuild
```

2. Build the Docker image.

```
$ docker build –t my-app-image .
```

3. Run the Docker container from the image.

Note

You must include the `-p` flag to map port 3000 on the container to the localhost port 8080. Elastic Beanstalk preconfigured Docker containers for Go applications always expose the application on port 3000 on the container. The `-it` flag runs the image as an interactive process. The `-rm` flag cleans up the container file system when the container exits. You can optionally include the `-d` flag to run the image as a daemon.

```
$ docker run –it --rm -p 8080:3000 my-app-image
```

4. To view the sample application, type the following URL into your web browser.

```
http://localhost:8080
```

Deploy to Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk.

To deploy your application to Elastic Beanstalk

1. In your application's root folder, rename the `Dockerfile` to `Dockerfile.local`. This step is required for Elastic Beanstalk to use the `Dockerfile` that contains the correct instructions for Elastic Beanstalk to build a customized Docker image on each Amazon EC2 instance in your Elastic Beanstalk environment.

   Note

   You do not need to perform this step if your `Dockerfile` includes instructions that modify the base Go Docker image. You do not need to use a `Dockerfile` if your `Dockerfile` includes only a `FROM` line to specify the base image from which to build the container. In that situation, the `Dockerfile` is redundant.

2. Create an application source bundle. For more information, see Create an Application Source Bundle (p. 43).
3. Create an Elastic Beanstalk environment to which you can deploy your application. For step-by-step instructions, see Create an Application (p. 34). At the appropriate step, on the **Environment Type** page, in the **Predefined configuration** list, under **Preconfigured - Docker**, click **Go**.

Creating and Deploying Elastic Beanstalk Applications in Java Using AWS Toolkit for Eclipse

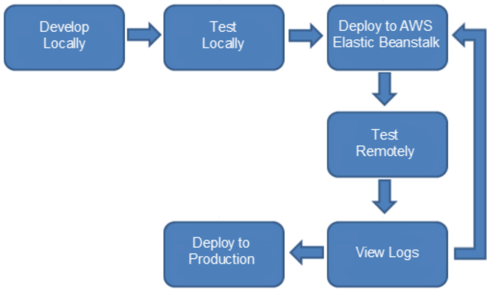
Topics

The first part of this topic provides step-by-step instructions for creating, testing, deploying, and redeploying your Java application to Elastic Beanstalk using the AWS Toolkit for Eclipse. The second part of this topic provides information on how you can manage and configure your applications and environments using the AWS Toolkit for Eclipse. For more information about prerequisites and installing the AWS Toolkit for Eclipse, go to http://aws.amazon.com/eclipse. You can also check out the Using AWS Elastic Beanstalk with the AWS Toolkit for Eclipse video. This topic also provides useful information covering tools, how-to topics, and additional resources for Java developers.

Develop, Test, and Deploy

Topics

The following diagram illustrates a typical software development life cycle that includes deploying your application to Elastic Beanstalk.



After developing and testing your application locally, you typically deploy your application to AWS Elastic Beanstalk. After deployment, your application will be live at a URL such as http://myexampleapp-wpams3yrvj.elasticbeanstalk.com.

If you want to replace the URL with a custom domain name, you can use Amazon Route 53, a highly available and scalable Domain Name System (DNS) web service. You can point your domain name to the Amazon Route 53 CNAME <yourappname>.elasticbeanstalk.com. Contact your DNS provider to set this up. For information about how to map your root domain to your Elastic Load Balancer, see Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295).

Because your application is live, you should consider setting up multiple environments, such as a testing environment and a production environment. You can use the AWS Toolkit for Eclipse to set up different AWS accounts for testing, staging, and production. For information about managing multiple accounts, see Managing Multiple AWS Accounts (p. 598).

After you remotely test and debug your Elastic Beanstalk application, you can make any updates and then redeploy it to Elastic Beanstalk. After you are satisfied with all of your changes, you can upload the latest version to your production environment. The following sections explain each stage of the software development life cycle.

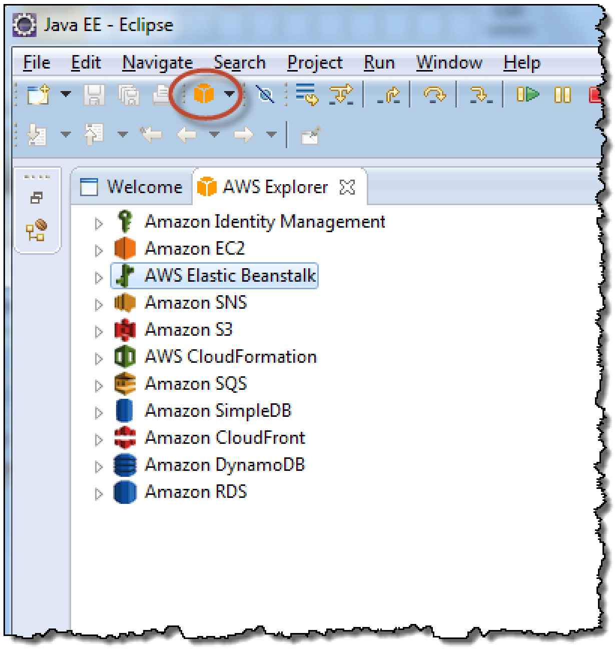
Create Project

The AWS Toolkit provides an AWS Java web project template for use in Eclipse. The template creates a web tools platform (WTP) dynamic web project that includes the AWS SDK for Java in the project's classpath. Your AWS account credentials and a simple `index.jsp` file are provided to help you get started. The following instructions assume you have installed both the Eclipse IDE for Java EE Developers and the AWS Toolkit plug-in. For more information, see Setting Up the AWS Toolkit for Eclipse.

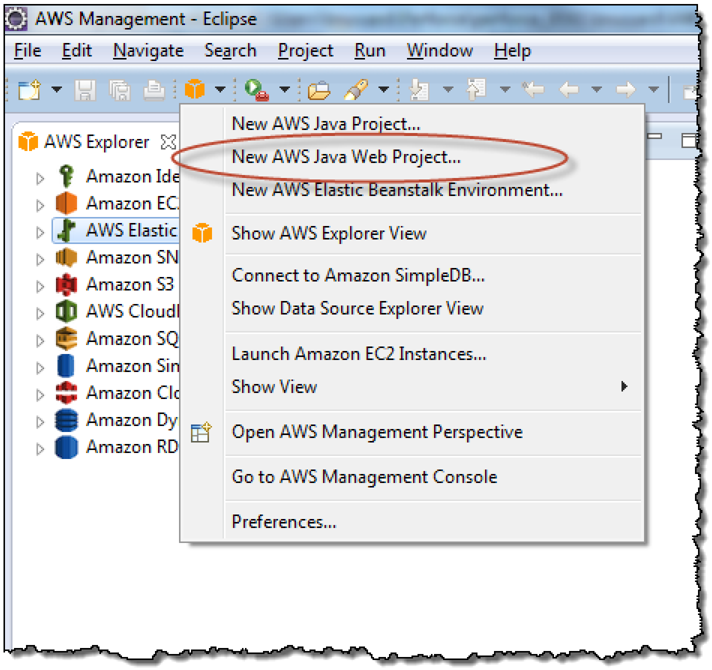
> **Note**
> For information about what versions of Java are supported, see Supported Platforms (p. 24).

To create a new AWS Java web project in Eclipse

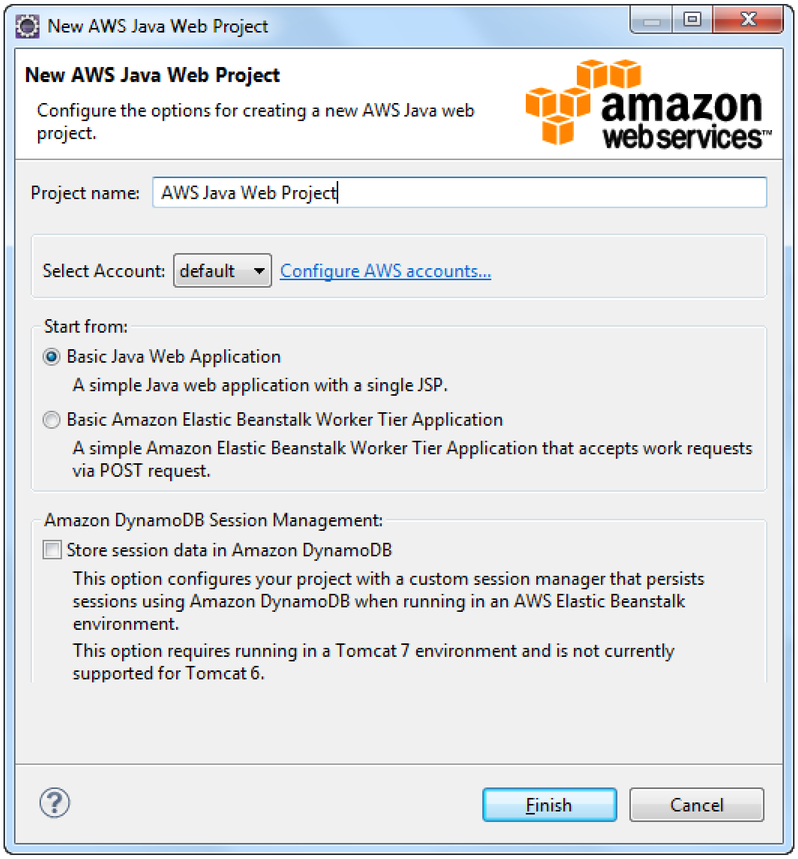
1. In Eclipse, make sure the toolbar and the AWS icon are visible.



2. On the toolbar, click the arrow next to the AWS icon and choose New AWS Java Web Project.


AWS Management - Eclipse
File Edit Navigate Search Project Run Window Help
AWS Explorer
Amazon Ide
Amazon EC
AWS Elastic
Amazon SN
Amazon S3
AWS Cloudl
Amazon SQ
Amazon Sin
Amazon Clo
Amazon Dy
Amazon RD
New AWS Java Project...
New AWS Java Web Project...
New AWS Elastic Beanstalk Environment...
Show AWS Explorer View
Connect to Amazon SimpleDB...
Show Data Source Explorer View
Launch Amazon EC2 Instances...
Show View
Open AWS Management Perspective
Go to AWS Management Console
Preferences...

3. In the **New AWS Java Web Project** wizard, click **Project name**, and then enter the name of your AWS Java Web Project.



New AWS Java Web Project

New AWS Java Web Project

Configure the options for creating a new AWS Java web project.

Project name: AWS Java Web Project

Select Account: default ▾ Configure AWS accounts...

Start from:

● Basic Java Web Application

A simple Java web application with a single JSP.

○ Basic Amazon Elastic Beanstalk Worker Tier Application

A simple Amazon Elastic Beanstalk Worker Tier Application that accepts work requests via POST request.

Amazon DynamoDB Session Management:

□ Store session data in Amazon DynamoDB

This option configures your project with a custom session manager that persists sessions using Amazon DynamoDB when running in an AWS Elastic Beanstalk environment.

This option requires running in a Tomcat 7 environment and is not currently supported for Tomcat 6.

4. Select the AWS account you want to use to deploy your application. If you haven't already configured an account, click **Configure AWS accounts** to add a new account. For instructions on how to add a new account, go to Managing Multiple AWS Accounts (p. 598).
5. Select the application you want to start from.
6. Click **Finish.**

A new AWS Java web project is created inside your Eclipse workspace. To help you get started developing your project, a basic `index.jsp` file is included inside your WebContent folder. Your AWS security credentials are included in the `AwsCredentials.properties` file inside your `src` folder.

Test Locally

You can test and debug your application before you deploy it to Elastic Beanstalk using the built-in WTP tools.

To test and debug your application locally

1. If Eclipse isn't displaying **Project Explorer** view, do one of the following:

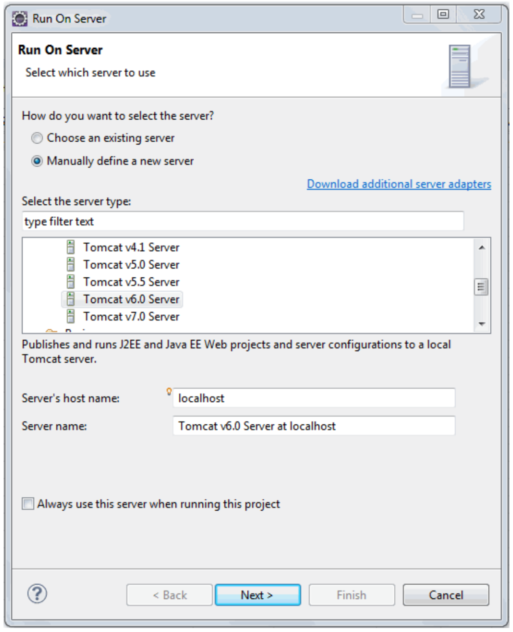
   - In the Java EE perspective, click the **Window** menu, click **Show View**, and then click **Project Explorer**.
   - In the AWS Management perspective, click the **Window** menu, click **Show View**, expand **General**, and then click **Project Explorer**.

2. In the **Project Explorer** tab, right-click your Java web project, choose **Run As**, **Run On Server**.

Note

If the **Run on Server** option does not appear, ensure that you have the Web Tools Platform Eclipse extensions installed. The WTP is included with Eclipse if you install the Eclipse IDE for Java EE Developers. The tools must be installed when your project is created for the option to appear.

3. In the **Run On Server** wizard, click **Manually define a new server**.



Run On Server

Run On Server

Select which server to use

How do you want to select the server?

○ Choose an existing server

● Manually define a new server

Download additional server adapters

Select the server type:

type filter text

- Tomcat v4.1 Server
- Tomcat v5.0 Server
- Tomcat v5.5 Server
- Tomcat v6.0 Server
- Tomcat v7.0 Server

Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.

Server's host name: localhost

Server name: Tomcat v6.0 Server at localhost

□ Always use this server when running this project

4. Under **Select the server type**, expand **Apache**, and then click **Tomcat v6.0 Server** or **Tomcat v7.0 Server**.
5. Click **Next**.
6. You might be prompted to install Apache. If you are using Microsoft Windows, you can click **Download and Install** to install the software. After you install the software, click **Finish**.

   ### Note

   You might need to manually create a Tomcat server. This includes going directly to http://apache.org to download Apache Tomcat, extracting the compressed file containing Apache Tomcat onto your computer, and then adding the Apache Tomcat runtimes. For more information, go to the "Creating an Apache Tomcat Server" topic of the Eclipse online documentation at help.eclipse.org.

   After you create a Tomcat server and add the Apache Tomcat runtimes, go to Step 2 in these procedures.

   Your application should be visible on the localhost.

Note

Environment properties that are typically provided by Elastic Beanstalk, such as RDS_HOSTNAME (when using RDS), are not available to applications running locally.

After you test your application, you can deploy to Elastic Beanstalk.

Deploy to AWS Elastic Beanstalk

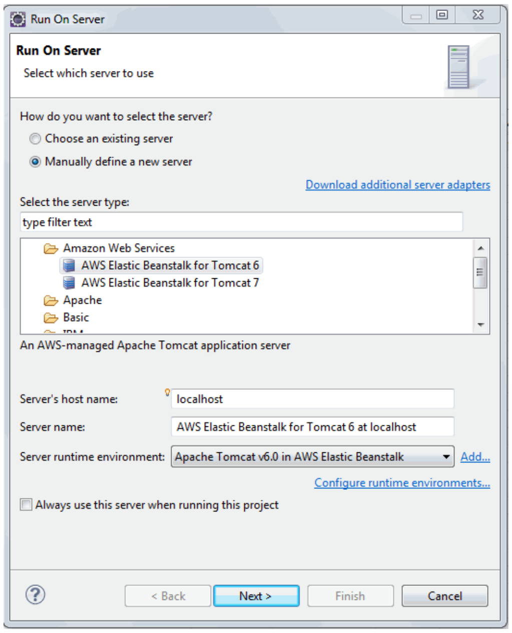
The Elastic Beanstalk server is an Eclipse WTP server with added functionality for restarting, publishing, and terminating your Java web application. The Elastic Beanstalk server in Eclipse represents an Elastic Beanstalk environment. An *environment* is a running instance of an application on the AWS platform.

Define Your Server

Before deploying your application, you must define your server and configure your application.

To define your Java web application server

1. In Eclipse, right-click your Java web project in the **Project Explorer** view. Choose **Amazon Web Services**, **Deploy to AWS Elastic Beanstalk**.
2. In the **Run On Server** wizard, select **Manually define a new server**.



Run On Server

Select which server to use

How do you want to select the server?

○ Choose an existing server

● Manually define a new server

Select the server type:

type filter text

- Amazon Web Services
  - AWS Elastic Beanstalk for Tomcat 6
  - AWS Elastic Beanstalk for Tomcat 7
- Apache
- Basic

An AWS-managed Apache Tomcat application server

Server's host name: localhost

Server name: AWS Elastic Beanstalk for Tomcat 6 at localhost

Server runtime environment: Apache Tomcat v6.0 in AWS Elastic Beanstalk

□ Always use this server when running this project

Download additional server adapters
Add...
Configure runtime environments...
< Back
Next >
Finish
Cancel

3. Under **Select the server type**, expand **Amazon Web Services**, and then click **Elastic Beanstalk for Tomcat 6** or **Elastic Beanstalk for Tomcat 7**.
4. If necessary, enter the server's host name and server name in the provided fields. Click **Next**.

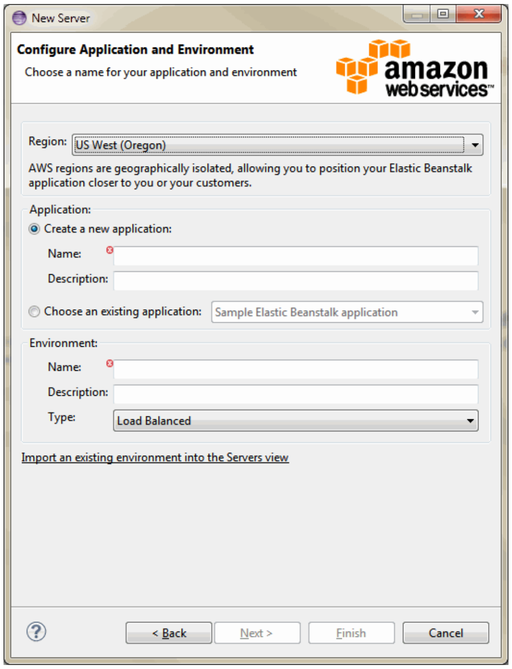
You have defined your AWS Java web application server. You must use the **Configure Application and Environment** options to configure your AWS Java web application before deployment.

Configure

Each application has a configuration and a version. A specific application instance is called an *environment*. An environment can have only one version at a time, but you can have multiple simultaneous environments running the same or different versions. AWS resources created for an environment can include one Elastic Load Balancing load balancer, an Auto Scaling group, and one or more Amazon EC2 instances.

To configure your Amazon web application

1. In the **Configure Application and Environment** view of the **Run On Server** wizard, click **Region**, and then select your Amazon Web Services region.



Configure Application and Environment

Choose a name for your application and environment

Region: US West (Oregon)

AWS regions are geographically isolated, allowing you to position your Elastic Beanstalk application closer to you or your customers.

Application:

● Create a new application:

Name:

Description:

○ Choose an existing application: Sample Elastic Beanstalk application

Environment:

Name:

Description:

Type: Load Balanced

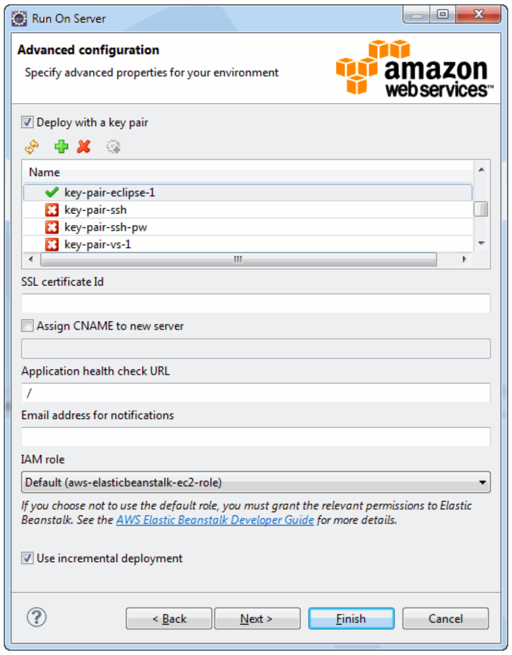
Import an existing environment into the Servers view

2. Enter an application name and, if desired, provide an application description.
3. Enter an environment name and, if desired, provide an environment description. The environment name must be unique within your AWS account.
4. Select the **Type** of environment that you want.

   You can select either **Load Balanced** or a **Single Instance** environment. For more information, see Environment Types (p. 68).

   > **Note**
   > For single-instance environments, load balancing, autoscaling, and the health check URL settings don't apply.

5. Click **Next**.
6. Click **Deploy with a key pair**.



Advanced configuration

Specify advanced properties for your environment

☑ Deploy with a key pair

Name
✓ key-pair-eclipse-1
☒ key-pair-ssh
☒ key-pair-ssh-pw
☒ key-pair-vs-1

SSL certificate Id

☐ Assign CNAME to new server

Application health check URL: /

Email address for notifications

IAM role: Default (aws-elasticbeanstalk-ec2-role)

If you choose not to use the default role, you must grant the relevant permissions to Elastic Beanstalk. See the AWS Elastic Beanstalk Developer Guide for more details.

☑ Use incremental deployment

7. Right-click anywhere inside the key pair list menu and select **New Key Pair**.
8. For **Key Pair Name**, enter a key pair name. For **Private Key Directory**, enter a private key directory or click **Browse** and select a directory. Click **OK** and select the newly created key pair in the key pair list menu.
9. Select **Use incremental deployment** to deploy only the changed files. An incremental deployment is faster because you are updating only the files that changed instead of all the files.
10. Select an instance profile.

    If you do not have an instance profile, select **Create a default instance profile**. For information about using instance profiles with Elastic Beanstalk, see Using IAM Roles with Elastic Beanstalk (p. 332).
11. Click **Finish**.

After you finish the wizard, you are prompted to enter a new version label for your application version, and a new server appears in the Server view. Your Java web project will be exported as a `.war` file, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature monitors your environment until it becomes available with the newly deployed code and opens your application in a web browser when it's ready.

Debug/View Logs

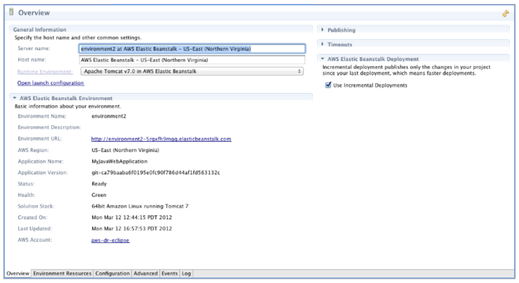
To investigate any issues, you can view logs. For information about viewing logs, see Instance Logs (p. 282). If you need to test remotely, you can connect to your EC2 instances. For instructions on how to connect to your instance, see Listing and Connecting to Server Instances (p. 280).

Edit the Application and Redeploy

Now that you have tested your application, it is easy to edit and redeploy it and to see the results in moments. Typically, when you redeploy your application, all of the files get updated. However, if you choose to do an incremental deployment, the toolkit updates the modified files, making your deployment faster.

To choose to use incremental deployments

1. If Eclipse isn't displaying **AWS Explorer** view, in the menu choose **Window**, **Show View**, **AWS Explorer**. In the **AWS Explorer** pane, expand the **Elastic Beanstalk** node and your application node.
2. Double-click your Elastic Beanstalk environment.
3. In the **Overview** tab, expand the **AWS Elastic Beanstalk Deployment** tab, and then select **Use Incremental Deployments**.



General Information
Specify the host name and other common settings.
Server name: environment2 at AWS Elastic Beanstalk - US-East (Northern Virginia)
Host name: AWS Elastic Beanstalk - US-East (Northern Virginia)
Runtime Environment: Apache Tomcat v7.0 in AWS Elastic Beanstalk
Open launch configuration

AWS Elastic Beanstalk Environment
Basic information about your environment.
Environment Name: environment2
Environment Description:
Environment URL: http://environment2-5rgxfh9mgg.elasticbeanstalk.com
AWS Region: US-East (Northern Virginia)
Application Name: MyJavaWebApplication
Application Version: git-ca79baaba6f0195e0fc90f786d44af1fd563132c
Status: Ready
Health: Green
Solution Stack: 64bit Amazon Linux running Tomcat 7
Created On: Mon Mar 12 12:44:15 PDT 2012
Last Updated: Mon Mar 12 16:57:53 PDT 2012
AWS Account: aws-dr-eclipse

AWS Elastic Beanstalk Deployment
Incremental deployment publishes only the changes in your project since your last deployment, which means faster deployments.
☑ Use Incremental Deployments

To edit and redeploy your Java web application

1. In Eclipse, locate and double-click the `index.jsp` file in the **Project Explorer** or **Package Explorer** view. Edit the source contained in the file.
2. Right-click your Java web project in the **Project Explorer** or **Package Explorer** view and choose **Amazon Web Services**, **Deploy to AWS Elastic Beanstalk**.
3. In the **Run On Server** wizard, select **Choose an existing server**, and click **Finish**.

   Note
   If you launched a new environment using the AWS Management Console, you will need to import your existing environments into Eclipse. For more information, see Importing Existing Environments into Eclipse (p. 584).

When the application has been deployed successfully, `index.jsp` is displayed in Eclipse and shows any edits you made.

Deploy to Production

When you are satisfied with all of the changes that you want to make to your application, you can deploy your application to your production environment. To deploy the existing version of the application to production with zero downtime, follow the steps at Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83). You can also create a new environment inside Eclipse, but you will need to specify a new version for your application. For instructions on deploying to Elastic Beanstalk inside Eclipse, see Deploy to AWS Elastic Beanstalk (p. 581).

Importing Existing Environments into Eclipse

You can import existing environments that you created in the AWS Management Console into Eclipse.

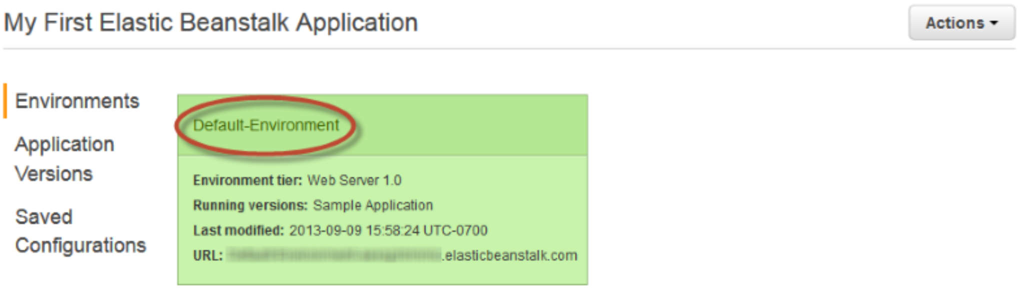
To import existing environments, expand the **AWS Elastic Beanstalk** node and double-click on an environment in the **AWS Explorer** inside Eclipse. You can now deploy your Elastic Beanstalk applications to this environment.

Configuring Java Containers with Elastic Beanstalk

AWS Management Console

To access the JVM container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

JVM Container Options

The heap size in the Java Virtual Machine affects how many objects can be created in memory before *garbage collection*—a process of managing your application's memory—occurs. You can specify an initial heap size and a maximum heap size. A larger initial heap size allows more objects to be created before garbage collection occurs, but it also means that the garbage collector will take longer to compact the heap. The maximum heap size specifies the maximum amount of memory the JVM can allocate when expanding the heap during heavy activity.

You can set the initial and the maximum JVM heap sizes using the **Initial JVM Heap Size (-Xms argument)** and **Maximum JVM Heap Size (-Xmx argument)** boxes. The available memory is dependent on the Amazon EC2 instance type. For more information about the Amazon EC2 instance types available for your Elastic Beanstalk environment, go to Instance Types in the *Amazon EC2 User Guide*.

The *permanent generation* is a section of the JVM heap that is used to store class definitions and associated metadata. To modify the size of the permanent generation, type the new size in the **Maximum JVM PermGen Size (-XX:MaxPermSize argument)** box.

Full documentation of JVM is beyond the scope of this guide; for more information on JVM garbage collection, go to Java Garbage Collection Basics.

Amazon S3 Log Rotation

Elastic Beanstalk can copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application on an hourly basis. To enable this feature, select **Enable log file rotation to Amazon S3**.

Environment Properties

The environment properties lets you specify environment properties on the Amazon EC2 instances that are running your application. Environment properties are specific to your application environment and are not actual (shell) environment variables. More specifically, **PARAM1**, **PARAM2**, etc. are system properties passed into the JVM at startup using the `-D` flag. You can use them to pass database connection strings, security credentials, or other information that you don't want to hard-code into your application. Storing this information in environment properties can help increase the portability and scalability of your application. You do not need to recompile your source code when you move between environments. You can acquire them with `System.getProperty(name)`. For more information on using and accessing custom environment properties, see Using Custom Environment Properties with Elastic Beanstalk (p. 589).

> **Note**
> The combined size of all environment variables defined for an environment is limited to 4096 bytes. The format of environment variables is **`KEY1=VALUE1, KEY2=VALUE2`**, which means that both the value and key of each variable are included in the total. When a platform has one or more predefined environment variables, such as **JDBC_CONNECTION_STRING** those variables are also included in the total.

You can configure the following environment properties:

- Specify AWS credentials using the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_KEY** boxes.

  > **Note**
  > Use instance profiles so that your application can use temporary security credentials to access AWS resources. To learn more, see Service Roles, Instance Profiles, and User Policies (p. 19).

- Specify a connection string to an external database (such as Amazon RDS) by entering it in the **JDBC_CONNECTION_STRING** box. For more information on how to set your JDBC_CONNECTION_STRING, see Using Custom Environment Properties with Elastic Beanstalk (p. 589).
- Specify up to five additional environment properties by entering them in the **PARAM** boxes.

  > **Note**
  > Environment properties can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Command Line Interface (CLI)

To edit an application's environment settings for your container/JVM options

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
        "OptionName": "Xms",
        "Value": "256m"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
        "OptionName": "Xmx",
        "Value": "256m"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
        "OptionName": "XX:MaxPermSize",
        "Value": "64m"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
        "OptionName": "JVM Options",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_ACCESS_KEY_ID",
        "Value": "AKIAIOSFODNN7EXAMPLE"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_SECRET_KEY",
        "Value": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "JDBC_CONNECTION_STRING",
        "Value": "jdbc:mysql://localhost:3306/mydatabase?user=me&password=my
password"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "myvar",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To edit an application's environment settings for your container/JVM options

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:container:tomcat:jvmoptions`
  - *`OptionSettings.member.1.OptionName`* = `Xms`
  - *`OptionSettings.member.1.Value`* = `256m`

- *OptionSettings.member.2.Namespace* = aws:elasticbeanstalk:container:tomcat:jvmoptions
- *OptionSettings.member.2.OptionName* = Xmx
- *OptionSettings.member.2.Value* = 256m
- *OptionSettings.member.3.Namespace* = aws:elasticbeanstalk:container:tomcat:jvmoptions
- *OptionSettings.member.3.OptionName* = XX:MaxPermSize
- *OptionSettings.member.3.Value* = 64m
- *OptionSettings.member.4.Namespace* = aws:elasticbeanstalk:container:tomcat:jvmoptions
- *OptionSettings.member.4.OptionName* = JVM Options
- *OptionSettings.member.4.Value* = somevalue
- *OptionSettings.member.5.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.5.OptionName* = AWS_ACCESS_KEY_ID
- *OptionSettings.member.5.Value* = AKIAIOSFODNN7EXAMPLE
- *OptionSettings.member.6.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.6.OptionName* = AWS_SECRET_KEY
- *OptionSettings.member.6.Value* = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
- *OptionSettings.member.7.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.7.OptionName* = JDBC_CONNECTION_STRING
- *OptionSettings.member.7.Value* = jdbc:mysql://localhost:3306/mydatabase?user=me&password=mypassword
- *OptionSettings.member.8.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.8.OptionName* = myvar
- *OptionSettings.member.8.Value* = somevalue
- *OptionSettings.member.9.Namespace* = aws:elasticbeanstalk:hostmanager
- *OptionSettings.member.9.OptionName* = LogPublicationControl
- *OptionSettings.member.9.Value* = false

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Atom
cat%3Ajvmoptions
&OptionSettings.member.1.OptionName=Xms
&OptionSettings.member.1.Value=256m
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Atom
cat%3Ajvmoptions
&OptionSettings.member.2.OptionName=Xmx
&OptionSettings.member.2.Value=256m
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Atom
cat%3Ajvmoptions
&OptionSettings.member.3.OptionName=XX:MaxPermSize
&OptionSettings.member.3.Value=64m
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Atom
cat%3Ajvmoptions
&OptionSettings.member.4.OptionName=JVM&20Options
&OptionSettings.member.4.Value=somevalue
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.5.OptionName=AWS_ACCESS_KEY_ID
&OptionSettings.member.5.Value=AKIAIOSFODNN7EXAMPLE
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.6.OptionName=AWS_SECRET_KEY
&OptionSettings.member.6.Value=wJalrXUtnFEMI%2FK7MDENG%2FbPxRfiCYEXAMPLEKEY
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.7.OptionName=JDBC_CONNECTION_STRING
&OptionSettings.member.7.Value=jdbc%3Amysql%3A%2F%2Flocalhost%3A3306%2Fmydata
base%3Fuser%3Dme%26password%3Dmypassword
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.8.OptionName=myvar
&OptionSettings.member.8.Value=somevalue
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.9.OptionName=LogPublicationControl
&OptionSettings.member.9.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Using Custom Environment Properties with Elastic Beanstalk

You can use the AWS Management Console or the AWS Toolkit for Eclipse to set environment properties that Elastic Beanstalk passes to your server instances. Environment properties are specific to your application environment and are not actual (shell) environment variables. More specifically, **PARAM1**, **PARAM2**, etc. are system properties passed into the JVM at startup using the `-D` flag. You can use them to pass database connection strings, security credentials, or other information that you don't want to hard-code into your application. Storing this information in environment properties can help increase the portability and scalability of your application. You do not need to recompile your source code when you move between environments. You can acquire them with `System.getProperty(name)`.

Setting Custom Environment Properties with AWS Toolkit for Eclipse

The following example sets the `JDBC_CONNECTION_STRING` and `PARAM1` environment properties in the AWS Toolkit for Eclipse. After you set these properties, they become available to your Elastic Beanstalk application as system properties called `JDBC_CONNECTION_STRING` and `PARAM1`, respectively.

The procedure for setting other environment properties is the same. You can use `PARAM1` through `PARAM5` for any purpose you choose, but you cannot rename the properties.

> **Note**
>
> The AWS Toolkit for Eclipse does not yet support modifying environment configuration, including environment variables, for environments in a VPC. Unless you have an older account using EC2 Classic, you must use the AWS Management Console (described in the next section) or the EB CLI (version 3 or newer) (p. 384)

> **Note**
>
> Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

To set environment properties for your Elastic Beanstalk application

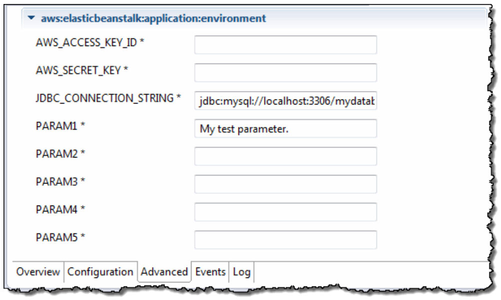
1. If Eclipse isn't displaying the **AWS Explorer** view, choose **Window**, **Show View**, **Other**. Expand **AWS Toolkit** and then click **AWS Explorer**.
2. In the **AWS Explorer** pane, expand **Elastic Beanstalk**, expand the node for your application, and then double-click your Elastic Beanstalk environment.
3. At the bottom of the pane for your environment, click the **Advanced** tab.
4. Under **aws:elasticbeanstalk:application:environment**, click **JDBC_CONNECTION_STRING** and then type a connection string. For example, the following JDBC connection string would connect to a MySQL database instance on port 3306 of localhost, with a user name of `me` and a password of `mypassword`:

   ```
   jdbc:mysql://localhost:3306/mydatabase?user=me&password=mypassword
   ```

   This will be accessible to your Elastic Beanstalk application as a system property called `JDBC_CONNECTION_STRING`.
5. Click **PARAM1** and then type a string. For example:

   ```
   My test parameter.
   ```

   This will be accessible to your Elastic Beanstalk application as a system property called `PARAM1`.



aws:elasticbeanstalk:application:environment

AWS_ACCESS_KEY_ID *	
AWS_SECRET_KEY *	
JDBC_CONNECTION_STRING *	jdbc:mysql://localhost:3306/mydatak
PARAM1 *	My test parameter.
PARAM2 *	
PARAM3 *	
PARAM4 *	
PARAM5 *	

6. Press **Ctrl+S** on the keyboard or choose **File**, **Save** to save your changes to the environment configuration. Changes are reflected in about one minute.

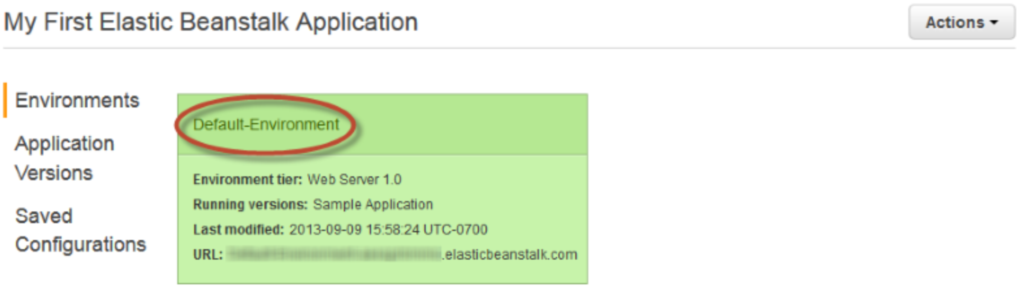
Setting Custom Environment Properties with AWS Management Console

The following example sets the `JDBC_CONNECTION_STRING` and `PARAM1` environment properties in the AWS Management Console. After you set these properties, they become available to your Elastic Beanstalk application as system properties called `JDBC_CONNECTION_STRING` and `PARAM1`, respectively.

The procedure for setting other environment properties is the same. You can use `PARAM1` through `PARAM5` for any purpose you choose, but you cannot rename the properties.

To set environment properties for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment name to view its dashboard.



My First Elastic Beanstalk Application

Environments
Application Versions
Saved Configurations

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the navigation pane, click **Configuration**.
4. On the **Configuration** page, in the **Software Configuration** section, click ⚙.
5. Under **Environment Properties**, next to **JDBC_CONNECTION_STRING**, in the **Property Value** column, type a connection string.

   For example, the following JDBC connection string would connect to a MySQL database instance on port 3306 of localhost, with a username of `me` and a password of `mypassword`:

   ```
   jdbc:mysql://localhost:3306/mydatabase?user=me&password=mypassword
   ```

   This will be accessible to your Elastic Beanstalk application as a system property called `JDBC_CONNECTION_STRING`.
6. Next to **PARAM1**, in the **Property Value** column, type a string, for example: **`My test parameter`**.

   This parameter will be accessible to your Elastic Beanstalk application as a system property called `PARAM1`.
7. Click **Save**.

   Elastic Beanstalk updates your environment. This takes about one minute.

Accessing Custom Environment Properties

After you set your environment properties for your Elastic Beanstalk application, you can access the environment properties from your code. For example, the following code snippet shows how to access the Elastic Beanstalk environment properties using JavaScript in a JavaServer Page (JSP):

```
<p>
    The JDBC_CONNECTION_STRING environment property is:
    <%= System.getProperty("JDBC_CONNECTION_STRING") %>
</p>

<p>
    The PARAM1 environment property is:
    <%= System.getProperty("PARAM1") %>
</p>
```

Using Amazon RDS and MySQL Connector/J

With Amazon Relational Database Service, you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. For more information about Amazon RDS, go to http://aws.amazon.com/rds/.

This topic explains how you can use Amazon RDS and MySQL Connector/J with your Elastic Beanstalk Java application.

> **Note**
> The instructions in this topic are for nonlegacy container types. If you have deployed an Elastic Beanstalk application using a legacy container type, we recommend that you migrate to a nonlegacy container type to gain access to new features. For instructions on how to check the container type and migrate your application, see Migrating Your Application from a Legacy Container Type (p. 91). For instructions on using MySQL Connector/J with applications running on legacy container types, see Using Amazon RDS and MySQL Connector/J (Legacy Container Types) (p. 863).

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. Download and install MySQL Connector/J.
3. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
4. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with Java

This topic walks you through creating a new Amazon RDS DB Instance and using it with your Java application.

1. Sign in to the AWS Management Console and open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Click your environment name and select **Configuration**.
3. Scroll down to the Data Tier section and click **Create a new RDS database**.

4. Enter a user name and password and click **Save**. Click **Save** again on the message about instance profiles, if shown. Elastic Beanstalk updates your environment to include the new RDS instance and provides connection information to the web container as environment properties.
5. Download and install MySQL Connector/J for your development environment. For download and installation instructions, go to http://dev.mysql.com/downloads/connector/j.
6. Copy the MySQL Connector/J `.jar` file into the project's `WebContent/WEB-INF/lib` directory.
7. Right click on `WebContent/WEB-INF/lib/`*`mysql_connector_java_5.1.34_bin.jar`* and click **Add to Build Path** to add the library to the classpath.
8. Test the database connection by adding the following code to the main view of your Java Web Application (`index.jsp`).

```
<%@ page import="java.sql.*" %>
<%
  // Read RDS Connection Information from the Environment
  String dbName = System.getProperty("RDS_DB_NAME");
  String userName = System.getProperty("RDS_USERNAME");
  String password = System.getProperty("RDS_PASSWORD");
  String hostname = System.getProperty("RDS_HOSTNAME");
  String port = System.getProperty("RDS_PORT");
  String jdbcUrl = "jdbc:mysql://" + hostname + ":" +
    port + "/" + dbName + "?user=" + userName + "&password=" + password;

  // Load the JDBC Driver
  try {
    System.out.println("Loading driver...");
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded!");
  } catch (ClassNotFoundException e) {
    throw new RuntimeException("Cannot find the driver in the classpath!",
 e);
  }

  Connection conn = null;
  Statement setupStatement = null;
  Statement readStatement = null;
  ResultSet resultSet = null;
  String results = "";
  int numresults = 0;
  String statement = null;

  try {
    // Create connection to RDS instance
    conn = DriverManager.getConnection(jdbcUrl);

    // Create a table and write two rows
    setupStatement = conn.createStatement();
    String createTable = "CREATE TABLE Beanstalk (Resource char(50));";
    String insertRow1 = "INSERT INTO Beanstalk (Resource) VALUES ('EC2 In
stance');";
    String insertRow2 = "INSERT INTO Beanstalk (Resource) VALUES ('RDS In
stance');";

    setupStatement.addBatch(createTable);
    setupStatement.addBatch(insertRow1);
    setupStatement.addBatch(insertRow2);
    setupStatement.executeBatch();
    setupStatement.close();

  } catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
    System.out.println("Closing the connection.");
    if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
  }

  try {
    conn = DriverManager.getConnection(jdbcUrl);

    readStatement = conn.createStatement();
   resultSet = readStatement.executeQuery("SELECT Resource FROM Beanstalk;");


    resultSet.first();
    results = resultSet.getString("Resource");
    resultSet.next();
    results += ", " + resultSet.getString("Resource");

    resultSet.close();
    readStatement.close();
    conn.close();

  } catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
       System.out.println("Closing the connection.");
      if (conn != null) try { conn.close(); } catch (SQLException ignore)
{}
  }
%>
```

9. Place the following code in the body of the html portion of `index.jsp` to display the results.

```
<p>Established connection to RDS. Read first two rows: <%= results %></p>
```

10. Right-click your project in the Package Explorer and choose **Run As**, **Run on Server**.
11. If your project is not configured to always deploy to the same environment, select the environment that your application is currently running on and click **Finish**.

For more information on getting started using the MySQL Connector/J to access your MySQL database, go to http://dev.mysql.com/doc/connector-j/en/index.html.

Using an Existing Amazon RDS DB Instance with Java

With Amazon Relational Database Service you can quickly and easily provision and maintain a MySQL Server instance in the cloud. This topic discusses how you can use Amazon RDS and the MySQL Connector/J with your Elastic Beanstalk application.

To use an existing Amazon RDS DB Instance and Java from your Elastic Beanstalk application

1. Create an Elastic Beanstalk environment in one of the following ways:

   - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For instructions using Eclipse, see Develop, Test, and Deploy (p. 576). You do not need to create an Amazon RDS DB Instance with this environment because you already have an existing Amazon RDS DB Instance.
   - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.

2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Download and install MySQL Connector/J for your development environment. For download and installation instructions, go to http://dev.mysql.com/downloads/connector/j.
4. Create a JDBC connection string that includesyour Amazon RDS DB instance's public DNS name, port number, and (optionally) database name and login credentials. The following example shows a JDBC connection string that would connect to the employees database on an Amazon RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

   ```
   jdbc:mysql://mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com:3306/em
   ployees?user=sa&password=mypassword
   ```

5. Configure your Elastic Beanstalk environment to pass the string to your Elastic Beanstalk application as an environment property. For instructions on how to do this, go to Using Custom Environment Properties with Elastic Beanstalk (p. 589).
6. Retrieve the JDBC connection string from the environment property passed to your server instance from Elastic Beanstalk and use MySQL Connector/J to access your Amazon RDS database. The following code example shows how to retrieve the JDBC_CONNECTION_STRING custom environment property from a Java Server Page (JSP).

   ```
   <p>
       The JDBC_CONNECTION_STRING environment variable is:
       <%= System.getProperty("JDBC_CONNECTION_STRING") %>
   </p>
   ```

For more information on getting started using the MySQL Connector/J to access your MySQL database, go to http://dev.mysql.com/doc/connector-j/en/index.html.

7. Copy the MySQL Connector/J `.jar` file into the Tomcat WEB-INF/lib directory.
8. Deploy your updated application to your existing Elastic Beanstalk environment. For information on how to deploy a new application version to an existing environment using the Elastic Beanstalk console, see Step 4: Deploy New Version (p. 6). For information on how to deploy your application using Eclipse, see Develop, Test, and Deploy (p. 576).

Note
To connect to Tomcat RDS environments, you must load the driver explicitly using `Class.forName(<driverClassName>)` prior to the call to `DriverManager.getConnection()` in the Java code.

Troubleshooting Database Connections

If and when you run into issues connecting to a database from within your application, the web container log and database itself are two good places to look for information.

Logs

You can view all of the logs from your Elastic Beanstalk environment from within Eclipse. If you don't have the AWS Explorer view open, click the arrow next to the orange AWS icon in the toolbar and choose **Show AWS Explorer View**. Expand **AWS Elastic Beanstalk** and your environment name, and then right-click the server and choose **Open in WTP Server Editor**.

Click the log tab of the server view to see the aggregate logs from your environment. Use the refresh button at the upper right corner of the view whenever you need to open the latest logs.

Scroll down and locate the TomCat logs from `/var/log/tomcat7/catalina.out`. If you loaded the web page from our earlier example several times, you might see the following in the output.

```
-------------------------------------
/var/log/tomcat7/catalina.out
-------------------------------------
INFO: Server startup in 9285 ms
Loading driver...
Driver loaded!
SQLException: Table 'Beanstalk' already exists
SQLState: 42S01
VendorError: 1050
Closing the connection.
Closing the connection.
```

Everything sent to standard output by the web application will show up in the web container log. This example tries to create the table on every page load and ends up catching a SQL exception on every page load after the first one.

Note
This is okay for an example, but in real world applications you'll keep your database definitions in schema objects, perform transactions from within model classes, and coordinate requests with controller servlets.

RDS

You can connect directly to the Amazon RDS instance in your Elastic Beanstalk environment using the My SQL client application.

First you'll need to open the security group to your Amazon RDS instance to allow traffic from your computer.

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Click the name of your environment and click **Configuration**.
3. Under **Network Tier**, in the **RDS** section, click ⚙the gear icon.
4. Select **View in RDS Console** next to the DB endpoint.
5. On the **RDS Dashboard** instance details page, under **Security and Network**, click the security group starting with *rds-* next to **Security Groups**.

   Note
   The database may have multiple entries labelled **Security Groups**. Use the first (starts with *awseb*) only if you have an older account that does not have a default VPC.
6. In the security group details, select the **Inbound** tab and click **Edit**
7. Add a rule for My SQL (port 3306) that allows traffic from your IP address, specified in CIDR format.
8. Click **Save**. The changes take effect immediately.

Return to the Elastic Beanstalk configuration details for your environment and note the endpoint. You will use the domain name to connect to the RDS instance.

Install My SQL client and initiate a connection to the database on port 3306. In Windows, install the My SQL Workbench application from the My SQL home page and follow the prompts.

In Linux, install the My SQL client using the package manager for your distribution. The following example works on Ubuntu and other Debian derivatives.

```
// Install My SQL client
$ sudo apt-get install mysql-client-5.5
...
// Connect to database
$ mysql -h aas839jo2vwhwb.cnubrrfwfka8.us-west-2.rds.amazonaws.com -u username
 -ppassword ebdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 117
Server version: 5.5.40-log Source distribution
...
```

Once connected, you can run SQL commands to see the status of the database, whether your tables and rows have been created, and other information.

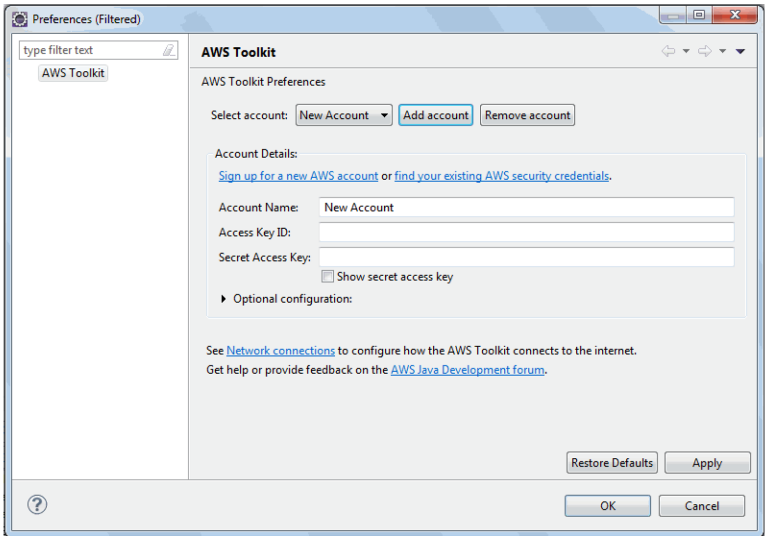
```
mysql> SELECT Resource from Beanstalk;
+--------------+
| Resource     |
+--------------+
| EC2 Instance |
| RDS Instance |
+--------------+
2 rows in set (0.01 sec)
```

Managing Multiple AWS Accounts

You might want to set up different AWS accounts to perform different tasks, such as testing, staging, and production. You can use the AWS Toolkit for Eclipse to add, edit, and delete accounts easily.

To add an AWS account with the AWS Toolkit for Eclipse

1. In Eclipse, make sure the toolbar is visible. On the toolbar, click the arrow next to the AWS icon and select **Preferences**.
2. Click **Add account**.



Sign up for a new AWS account or find your existing AWS security credentials.

Account Name: New Account

Access Key ID:

Secret Access Key:

□ Show secret access key

See Network connections to configure how the AWS Toolkit connects to the internet.

Get help or provide feedback on the AWS Java Development forum.

3. In the **Account Name** text box, type the display name for the account.
4. In the **Access Key ID** text box, type your AWS access key ID.
5. In the **Secret Access Key** text box, type your AWS secret key.

   For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see How Do I Get Security Credentials? in the *AWS General Reference*.
6. Click **OK**.

To use a different account to deploy an application to Elastic Beanstalk

1. In the Eclipse toolbar, click the arrow next to the AWS icon and select **Preferences**.
2. For **Default Account**, select the account you want to use to deploy applications to Elastic Beanstalk.
3. Click **OK**.
4. In the **Project Explorer** pane, right-click the application you want to deploy, and then select **Amazon Web Services** > **Deploy to Elastic Beanstalk**.

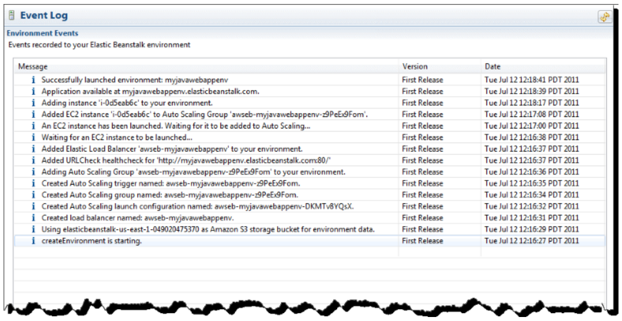
Viewing Events

You can use the AWS Toolkit for Eclipse to access events and notifications associated with your application. For more details on the most common events, see Understanding Environment Launch Events (p. 531).

To view application events

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu click **Window** > **Show View** > **AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In the AWS Explorer, double-click your Elastic Beanstalk environment.
3. At the bottom of the pane, click the **Events** tab.

   A list of the events for all environments for your application is displayed.



Environment Events

Events recorded to your Elastic Beanstalk environment

Message	Version	Date
i Successfully launched environment: myjavawebappenv	First Release	Tue Jul 12 12:18:41 PDT 2011
i Application available at myjavawebappenv.elasticbeanstalk.com.	First Release	Tue Jul 12 12:18:39 PDT 2011
i Adding instance 'i-0d5eab6c' to your environment.	First Release	Tue Jul 12 12:18:17 PDT 2011
i Added EC2 instance 'i-0d5eab6c' to Auto Scaling Group 'awseb-myjavawebappenv-z9PeEx9Fom'.	First Release	Tue Jul 12 12:17:08 PDT 2011
i An EC2 instance has been launched. Waiting for it to be added to Auto Scaling...	First Release	Tue Jul 12 12:17:00 PDT 2011
i Waiting for an EC2 instance to be launched...	First Release	Tue Jul 12 12:16:38 PDT 2011
i Added Elastic Load Balancer 'awseb-myjavawebappenv' to your environment.	First Release	Tue Jul 12 12:16:37 PDT 2011
i Added URLCheck healthcheck for 'http://myjavawebappenv.elasticbeanstalk.com:80/'	First Release	Tue Jul 12 12:16:37 PDT 2011
i Adding Auto Scaling Group 'awseb-myjavawebappenv-z9PeEx9Fom' to your environment.	First Release	Tue Jul 12 12:16:36 PDT 2011
i Created Auto Scaling trigger named: awseb-myjavawebappenv-z9PeEx9Fom.	First Release	Tue Jul 12 12:16:35 PDT 2011
i Created Auto Scaling group named: awseb-myjavawebappenv-z9PeEx9Fom.	First Release	Tue Jul 12 12:16:34 PDT 2011
i Created Auto Scaling launch configuration named: awseb-myjavawebappenv-DKMTv8YQsX.	First Release	Tue Jul 12 12:16:32 PDT 2011
i Created load balancer named: awseb-myjavawebappenv.	First Release	Tue Jul 12 12:16:31 PDT 2011
i Using elasticbeanstalk-us-east-1-049020475370 as Amazon S3 storage bucket for environment data.	First Release	Tue Jul 12 12:16:29 PDT 2011
i createEnvironment is starting.	First Release	Tue Jul 12 12:16:27 PDT 2011

Managing Elastic Beanstalk Application Environments

Topics

With the AWS Toolkit for Eclipse and the AWS Management Console, you can change the provisioning and configuration of the AWS resources that are used by your application environments. For information on how to manage your application environments using the AWS Management Console, see Managing Environments (p. 50). This section discusses the specific service settings you can edit in the AWS Toolkit for Eclipse as part of your application environment configuration. For more about AWS Toolkit for Eclipse, see AWS Toolkit for Eclipse Getting Started Guide.

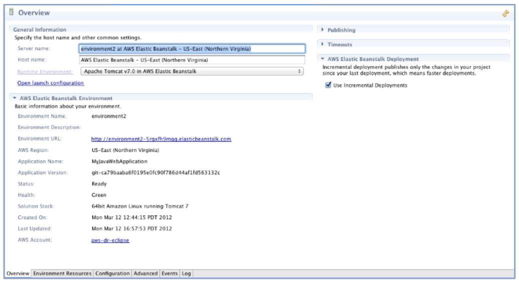
Changing Environment Configuration Settings

When you deploy your application, Elastic Beanstalk configures a number of AWS cloud computing services. You can control how these individual services are configured using the AWS Toolkit for Eclipse.

To edit an application's environment settings

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu click **Window** > **Show View** > **AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In **AWS Explorer**, double-click your Elastic Beanstalk environment.

3. At the bottom of the pane, click the **Configuration** tab.



Overview
General Information
Publishing
Timeouts
AWS Elastic Beanstalk Deployment
AWS Elastic Beanstalk Environment
Overview | Environment Resources | Configuration | Advanced | Events | Log

Specify the host name and other common settings.

Server name: environment2 at AWS Elastic Beanstalk - US-East (Northern Virginia)

Host name: AWS Elastic Beanstalk - US-East (Northern Virginia)

Runtime Environment: Apache Tomcat v7.0 in AWS Elastic Beanstalk

Open launch configuration

Incremental deployment publishes only the changes in your project since your last deployment, which means faster deployments.

☑ Use Incremental Deployments

Basic information about your environment.

Environment Name: environment2

Environment Description:

Environment URL: http://environment2-5rgxfh9mgq.elasticbeanstalk.com

AWS Region: US-East (Northern Virginia)

Application Name: MyJavaWebApplication

Application Version: git-ca79baaba6f0195e0fc90f786d44af1fd563132c

Status: Ready

Health: Green

Solution Stack: 64bit Amazon Linux running Tomcat 7

Created On: Mon Mar 12 12:44:15 PDT 2012

Last Updated: Mon Mar 12 16:57:53 PDT 2012

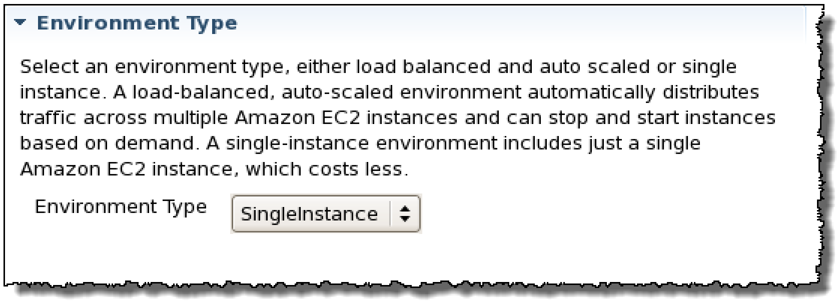
AWS Account: aws-dr-eclipse

You can now configure settings for the following:

- EC2 server instances
- Load balancer
- Autoscaling
- Notifications
- Environment types
- Environment properties

Changing Environment Type

In AWS Toolkit for Eclipse, the **Environment Type** section of your environment's **Configuration** tab lets you select either **Load balanced, auto scaled** or a **Single instance** environment, depending on the requirements of the application that you deploy. For an application that requires scalability, select **Load balanced, auto scaled**. For a simple, low traffic application, select **Single instance**. For more information, see Environment Types (p. 68).



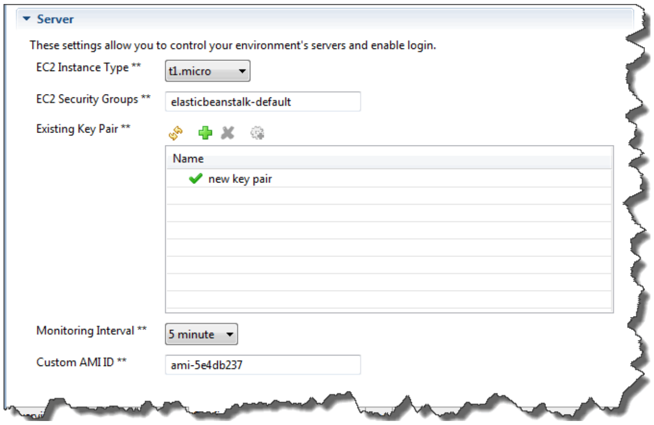
Select an environment type, either load balanced and auto scaled or single instance. A load-balanced, auto-scaled environment automatically distributes traffic across multiple Amazon EC2 instances and can stop and start instances based on demand. A single-instance environment includes just a single Amazon EC2 instance, which costs less.

Environment Type: SingleInstance

Configuring EC2 Server Instances Using AWS Toolkit for Eclipse

Amazon Elastic Compute Cloud (EC2) is a web service for launching and managing server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to the Amazon EC2 product page.

Under **Server**, on your environment's **Configuration** tab inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration.



Server

These settings allow you to control your environment's servers and enable login.

EC2 Instance Type ** t1.micro

EC2 Security Groups ** elasticbeanstalk-default

Existing Key Pair **

Name
✓ new key pair

Monitoring Interval ** 5 minute

Custom AMI ID ** ami-5e4db237

Amazon EC2 Instance Types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations may require more CPU or memory. Elastic Beanstalk regularly checks your running instances to ensure they are healthy. If your application consumes 95 percent or greater of the CPU, Elastic Beanstalk will trigger an event. For more information about this event, see CPU Utilization Exceeds 95.00% (p. 532).

> **Note**
> You cannot change between 32-bit and 64-bit instance types. For example, if your application is built on a 32-bit platform, only 32-bit instance types appear in the list.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see Instance Types in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Security Groups

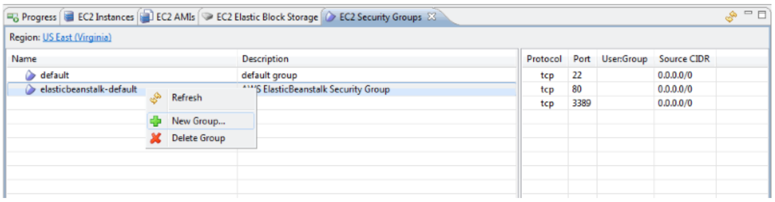
You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the AWS Management Console or by using the AWS Toolkit for Eclipse. You can specify which Amazon EC2 security groups control access to your Elastic Beanstalk application by entering the names of one or more Amazon EC2 security group names (delimited by commas) into the **EC2 Security Groups** box.

> **Note**
> If you are running your application using a legacy container type, make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see Health Checks (p. 604). To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).

To create a security group using the AWS Toolkit for Eclipse

1. In the AWS Toolkit for Eclipse, click **AWS Explorer** tab. Expand the **Amazon EC2** node, and then double-click **Security Groups**.
2. Right-click anywhere in the left table, and then click **New Group**.



Region: US East (Virginia)

Name	Description
default	default group
elasticbeanstalk-default	AWS ElasticBeanstalk Security Group

Protocol	Port	User:Group	Source CIDR
tcp	22		0.0.0.0/0
tcp	80		0.0.0.0/0
tcp	3389		0.0.0.0/0

3. In the **Security Group** dialog box, type the security group name and description and then click **OK**.

For more information on Amazon EC2 Security Groups, see Using Security Groups in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Key Pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

> **Important**
> You must create an Amazon EC2 key pair and configure your Elastic Beanstalk-provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk-provisioned Amazon EC2 instances. You can create your key pair using the **Publish to Beanstalk Wizard** inside AWS Toolkit for Eclipse when you deploy your application to Elastic Beanstalk. Alternatively, you can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the Amazon Elastic Compute Cloud Getting Started Guide.

For more information on Amazon EC2 key pairs, go to Using Amazon EC2 Credentials in the *Amazon Elastic Compute Cloud User Guide*. For more information on connecting to Amazon EC2 instances, go to Connecting to Instances and Connecting to a Linux/UNIX Instance from Windows using PuTTY in the *Amazon Elastic Compute Cloud User Guide*.

CloudWatch Metrics

By default, only basic Amazon CloudWatch metrics are enabled; they return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

> **Note**
> Amazon CloudWatch service charges can apply for one-minute interval metrics. See Amazon CloudWatch for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

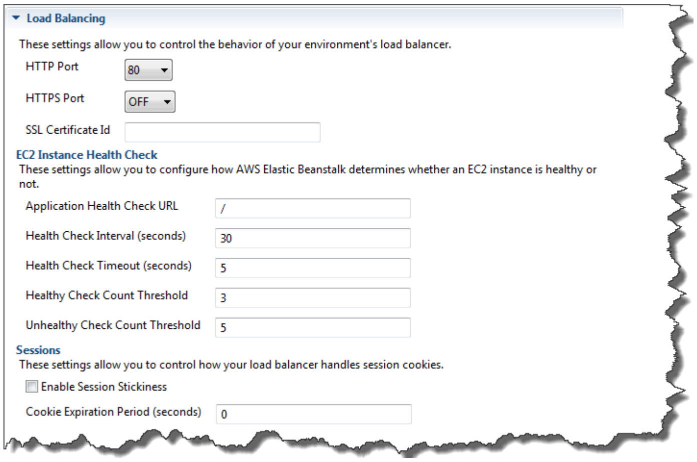
> **Important**
> Using your own AMI is an advanced task and should be done with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing Using AWS Toolkit for Eclipse

Elastic Load Balancing is an Amazon web service that improves the availability and scalability of your application. With Elastic Load Balancing, you can distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing improves availability through redundancy, and it supports traffic growth for your application.

Elastic Load Balancing automatically distributes and balances incoming application traffic among all the EC2 server instances you are running. The service also makes it easy to add new instances when you need to increase the capacity of your application.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. Under **Load Balancing**, on the **Configuration** tab for your environment inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's load balancing configuration.



Load Balancing

These settings allow you to control the behavior of your environment's load balancer.

HTTP Port: 80

HTTPS Port: OFF

SSL Certificate Id:

EC2 Instance Health Check

These settings allow you to configure how AWS Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check URL: /

Health Check Interval (seconds): 30

Health Check Timeout (seconds): 5

Healthy Check Count Threshold: 3

Unhealthy Check Count Threshold: 5

Sessions

These settings allow you to control how your load balancer handles session cookies.

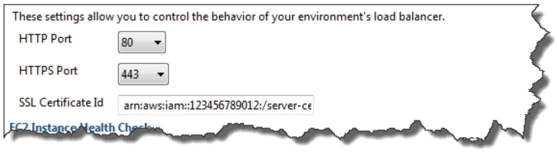
☐ Enable Session Stickiness

Cookie Expiration Period (seconds): 0

The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. At least one of the ports (either HTTP or HTTPS) must be turned on.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Port: 80

HTTPS Port: 443

SSL Certificate Id: arn:aws:iam::123456789012:/server-ce

EC2 Instance Health Check

Important
Make sure that the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application.

Controlling the HTTP port

To turn off the HTTP port, you select OFF for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**).

> **Note**
>
> If you want to access your environment using a different port other than the default port 80 (e.g., port 8080), you can add a listener to the existing load balancer and configure the new listener to listen on that port. For example, using the Elastic Load Balancing API Tools, type the following command replacing `<yourloadbalancername>` with the name of your load balancer for Elastic Beanstalk.
>
> ```
> elb-create-lb-listeners --lb <yourloadbalancername> --listener "pro
> tocol=http, lb-port=8080, instance-port=80"
> ```
>
> If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances are done using plain text. By default, the HTTPS port is turned off.

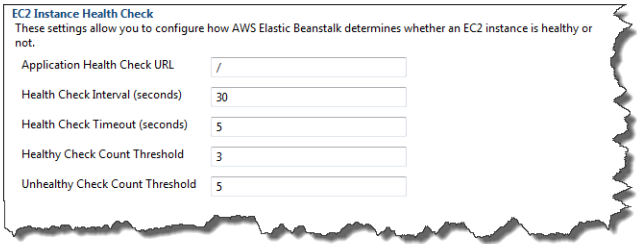
To turn on the HTTPS port

1. Create and upload a certificate and key to the AWS Identity and Access Management (IAM) service. The IAM service will store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate you've uploaded. For more information on creating and uploading certificates, see the Managing Server Certificates section of *Using AWS Identity and Access Management*.
2. Specify the HTTPS port by selecting a port from the **HTTPS Listener Port** drop-down list.
3. In the **SSL Certificate ID** text box, enter the Amazon Resources Name (ARN) of your SSL certificate (e.g., `arn:aws:iam::123456789012:server-certificate/abc/certs/build`). Use the SSL certificate that you created and uploaded in step 1. For information on viewing the certificate's ARN, see Verify the Certificate Object topic in the *Creating and Uploading Server Certificates* section of the *Using IAM Guide*.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health Checks

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.



EC2 Instance Health Check

These settings allow you to configure how AWS Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check URL	/
Health Check Interval (seconds)	30
Health Check Timeout (seconds)	5
Healthy Check Count Threshold	3
Unhealthy Check Count Threshold	5

The following list describes the health check parameters you can set for your application.

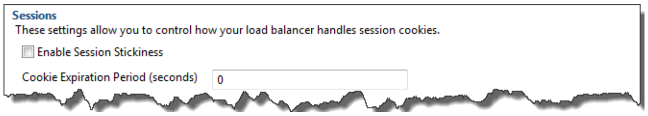
- To determine instance health, Elastic Beanstalk looks for a 200 response code on a URL it queries. By default, Elastic Beanstalk checks TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override to match an existing resource in your application (e.g., **`/myapp/index.jsp`**) by entering it in the **Application Health Check URL** box. If you override the default URL, Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).
- For **Health Check Interval (seconds)**, enter the number of seconds between your application's Amazon EC2 instances health checks.
- For **Health Check Timeout**, specify the number of seconds for Elastic Load Balancing to wait for a response before it considers an instance unresponsive.
- Use the **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold** boxes, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying 5 in the **Unhealthy Check Count Threshold** text box means that the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check "failed."

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer–generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer–generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If it finds no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

Under **Load Balancer** in the **Sessions** section, specify whether or not the load balancer for your application allows session stickiness and the duration for each cookie.



Sessions

These settings allow you to control how your load balancer handles session cookies.

□ Enable Session Stickiness

Cookie Expiration Period (seconds) 0

For more information on Elastic Load Balancing, go to the Elastic Load Balancing Developer Guide.

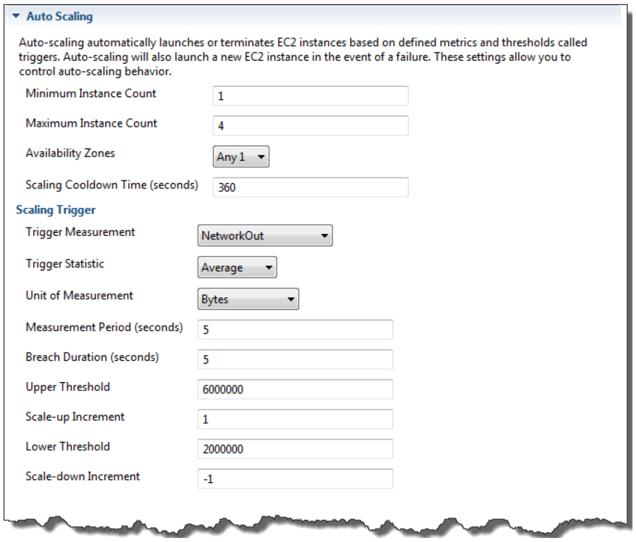
Configuring Auto Scaling Using AWS Toolkit for Eclipse

Auto Scaling is an Amazon web service designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. Users can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Auto Scaling for your application. Under **Auto Scaling**, on your environment's **Configuration** tab inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's Auto Scaling configuration.



Auto Scaling

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count	1
Maximum Instance Count	4
Availability Zones	Any 1
Scaling Cooldown Time (seconds)	360

Scaling Trigger

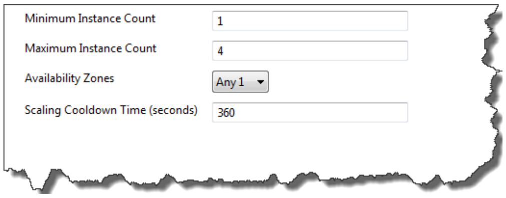
Trigger Measurement	NetworkOut
Trigger Statistic	Average
Unit of Measurement	Bytes
Measurement Period (seconds)	5
Breach Duration (seconds)	5
Upper Threshold	6000000
Scale-up Increment	1
Lower Threshold	2000000
Scale-down Increment	-1

The following sections discuss how to configure Auto Scaling parameters for your application.

Launch Configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Auto Scaling resources.

Use the **Minimum Instance Count** and **Maximum Instance Count** settings to specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.



Minimum Instance Count	1
Maximum Instance Count	4
Availability Zones	Any 1
Scaling Cooldown Time (seconds)	360

Note

To maintain a fixed number of Amazon EC2 instances, set the **Minimum Instance Count** and **Maximum Instance Count** text boxes to the same value.

For **Availability Zones**, specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications: If one Availability Zone goes down, your instances will still be running in your other Availability Zones.

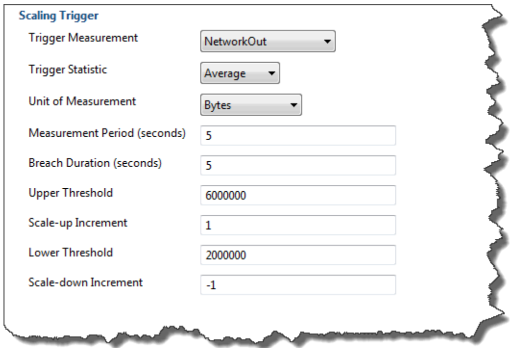
Note

Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Auto Scaling mechanism that you set to tell the system when to increase (*scale out*) and decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch, such as CPU utilization, and determine whether the specified conditions have been met. When your upper or lower thresholds for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *scaling activity*.

You can define a scaling trigger for your Elastic Beanstalk application using the AWS Toolkit for Eclipse.



Scaling Trigger

Trigger Measurement	NetworkOut
Trigger Statistic	Average
Unit of Measurement	Bytes
Measurement Period (seconds)	5
Breach Duration (seconds)	5
Upper Threshold	6000000
Scale-up Increment	1
Lower Threshold	2000000
Scale-down Increment	-1

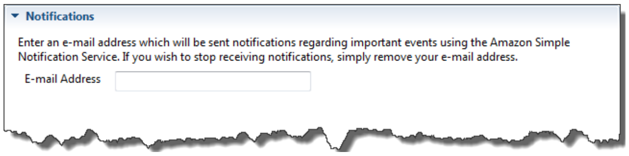
You can configure the following list of trigger parameters in the **Scaling Trigger** section of the **Configuration** tab for your environment inside the Toolkit for Eclipse.

- For **Trigger Measurement**, specify the metric for your trigger.
- For **Trigger Statistic**, specify which statistic the trigger will use—**`Minimum`**, **`Maximum`**, **`Sum`**, or **`Average`**.
- For **Unit of Measurement**, specify the units for the trigger measurement.
- For **Measurement Period**, specify how frequently Amazon CloudWatch measures the metrics for your trigger. For **Breach Duration**, specify the amount of time a metric can be beyond its defined limit (as specified for **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Scale-up Increment** and **Scale-down Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Auto Scaling, go to the Auto Scaling documentation.

Configuring Notifications Using AWS Toolkit for Eclipse

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, simply enter your email address in the **Email Address** text box under **Notifications** on the **Configuration** tab for your environment inside the Toolkit for Eclipse. To disable Amazon SNS notifications, remove your email address from the text box.



Notifications

Enter an e-mail address which will be sent notifications regarding important events using the Amazon Simple Notification Service. If you wish to stop receiving notifications, simply remove your e-mail address.

E-mail Address

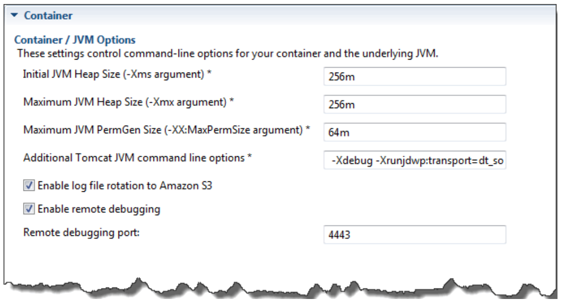
Configuring Java Containers Using AWS Toolkit for Eclipse

The **Container/JVM Options** panel lets you fine-tune the behavior of the Java Virtual Machine on your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can use the AWS Toolkit for Eclipse to configure your container information.

> **Note**
> You can modify your configuration settings with zero downtime by swapping the CNAME for your environments. For more information, see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

To access the Container/JVM Options panel for your Elastic Beanstalk application

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu click **Window** > **Show View** > **AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In the **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the pane, click the **Configuration** tab.
4. Under **Container**, you can configure container options.



Container

Container / JVM Options

These settings control command-line options for your container and the underlying JVM.

Initial JVM Heap Size (-Xms argument) * 256m

Maximum JVM Heap Size (-Xmx argument) * 256m

Maximum JVM PermGen Size (-XX:MaxPermSize argument) * 64m

Additional Tomcat JVM command line options * -Xdebug -Xrunjdwp:transport=dt_so

☑ Enable log file rotation to Amazon S3

☑ Enable remote debugging

Remote debugging port: 4443

JVM Options

The heap size in the Java Virtual Machine affects how many objects can be created in memory before *garbage collection*—a process of managing your application's memory—occurs. You can specify an initial heap size and a maximum heap size. A larger initial heap size allows more objects to be created before garbage collection occurs, but it also means that the garbage collector will take longer to compact the heap. The maximum heap size specifies the maximum amount of memory the JVM can allocate when expanding the heap during heavy activity.

You can set the initial and the maximum JVM heap sizes using the **Initial JVM Heap Size (-Xms argument)** and **Maximum JVM Heap Size (-Xmx argument)** boxes. The available memory is dependent on the Amazon EC2 instance type. For more information about the Amazon EC2 instance types available for your Elastic Beanstalk environment, go to Instance Types in the *Amazon EC2 User Guide*.

The *permanent generation* is a section of the JVM heap that is used to store class definitions and associated metadata. To modify the size of the permanent generation, type the new size in the **Maximum JVM PermGen Size (-XX:MaxPermSize argument)** box.

Full documentation of JVM is beyond the scope of this guide; for more information on JVM garbage collection, go to Java Garbage Collection Basics.

Amazon S3 Log Rotation

Elastic Beanstalk can copy the log files on an hourly basis for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application. To enable this feature, select **Enable log file rotation to Amazon S3**.

Remote Debugging

To test your application remotely, you can run your application in debug mode.

To enable remote debugging

1. Select **Enable remote debugging**.
2. For **Remote debugging port**, specify the port number to use for remote debugging.

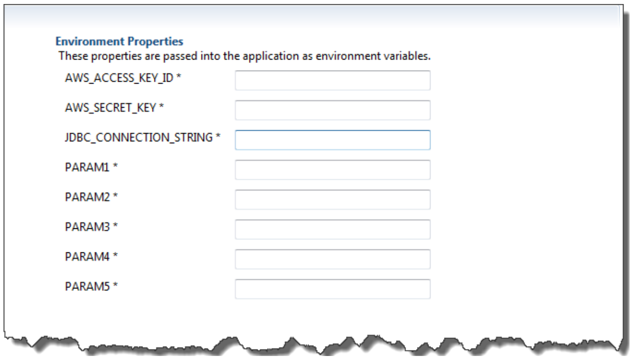
   The **Additional Tomcat JVM command line options** setting is filled automatically.

To start remote debugging

1. In the AWS Toolkit for Eclipse menu, click **Window** > **Show View** > **Other**.
2. Expand the **Server** folder, and then click **Servers**. Click **OK**.
3. In the **Servers** pane, right-click the server your application is running on, and then click **Restart in Debug**.

Environment Properties

This section of the **Container** tab lets you specify environment properties on the Amazon EC2 instances that are running your application. Environment properties are specific to your application environment and are not actual (shell) environment variables. More specifically, **PARAM1**, **PARAM2**, etc. are system properties passed into the JVM at startup using the `-D` flag. You can use them to pass database connection strings, security credentials, or other information that you don't want to hard-code into your application. Storing this information in environment properties can help increase the portability and scalability of your application. You do not need to recompile your source code when you move between environments. You can acquire them with `System.getProperty(name)`. For more information on using and accessing custom environment properties, see Using Custom Environment Properties with Elastic Beanstalk (p. 589)



Environment Properties
These properties are passed into the application as environment variables.
AWS_ACCESS_KEY_ID *
AWS_SECRET_KEY *
JDBC_CONNECTION_STRING *
PARAM1 *
PARAM2 *
PARAM3 *
PARAM4 *
PARAM5 *

You can configure the following environment properties:

- Specify AWS credentials using the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_KEY** boxes.

> **Note**
> Use instance profiles so that your application can use temporary security credentials to access AWS resources. To learn more, see Service Roles, Instance Profiles, and User Policies (p. 19).

- Specify a connection string to an external database (such as Amazon RDS) by entering it in the **JDBC_CONNECTION_STRING** box. For more information on how to set your JDBC_CONNECTION_STRING, see Using Custom Environment Properties with Elastic Beanstalk (p. 589).
- Specify up to five additional environment properties by entering them in the **PARAM** boxes.

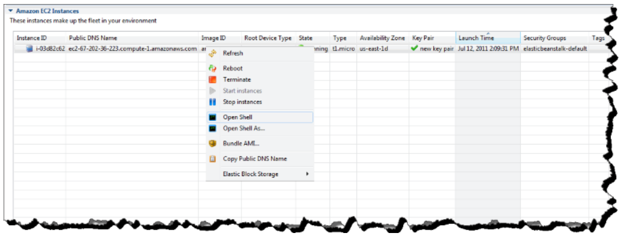
> **Note**
> Environment properties can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Listing and Connecting to Server Instances

You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Toolkit for Eclipse or from the AWS Management Console. You can connect to these instances using Secure Shell (SSH). For information about listing and connecting to your server instances using the AWS Management Console, see Listing and Connecting to Server Instances (p. 280). The following section steps you through viewing and connecting you to your server instances using the AWS Toolkit for Eclipse.

To view and connect to Amazon EC2 instances for an environment

1. In the AWS Toolkit for Eclipse, click **AWS Explorer**. Expand the **Amazon EC2** node, and then double-click **Instances**.
2. In the Amazon EC2 Instances window, in the **Instance ID** column, right-click the **Instance ID** for the Amazon EC2 instance running in your application's load balancer. Then click **Open Shell**.



These instances make up the fleet in your environment

Instance ID	Public DNS Name	Image ID	Root Device Type	State	Type	Availability Zone	Key Pair	Launch Time	Security Groups	Tags
i-03d82c62	ec2-67-202-36-223.compute-1.amazonaws.com	ar		nning	t1.micro	us-east-1d	✓ new key pair	Jul 12, 2011 2:09:31 PM	elasticbeanstalk-default	

Eclipse automatically opens the SSH client and makes the connection to the EC2 instance.

For more information on connecting to an Amazon EC2 instance, see the Amazon Elastic Compute Cloud Getting Started Guide.

Terminating an Environment

To avoid incurring charges for unused AWS resources, you can use the AWS Toolkit for Eclipse to terminate a running environment.

> **Note**
> You can always launch a new environment using the same version later.

To terminate an environment

1. In the AWS Toolkit for Eclipse, click the **AWS Explorer** pane. Expand the **Elastic Beanstalk** node.

2. Expand the Elastic Beanstalk application and right-click on the Elastic Beanstalk environment.
3. Click **Terminate Environment**. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

Tools

AWS SDK for Java

With the AWS SDK for Java, you can get started in minutes with a single, downloadable package that includes the AWS Java library, code samples, and documentation. You can build Java applications on top of APIs to take the complexity out of coding directly against a web service interface. The library provides APIs that hide much of the lower level plumbing, including authentication, request retries, and error handling. Practical examples are provided for how to use the library to build applications.

In addition, with the AWS SDK for Java, you can use DynamoDB to share session states of Apache Tomcat applications across multiple web servers. For more information, see Manage Tomcat Session State with Amazon DynamoDB in the AWS SDK for Java documentation. For more information about the AWS SDK for Java, go to http://aws.amazon.com/sdkforjava/.

AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse is an open source plug-in for the Eclipse Java IDE that makes it easier to develop and debug Java applications using Amazon Web Services, and now includes the Elastic Beanstalk deployment feature. With the Elastic Beanstalk deployment feature, developers can use Eclipse to deploy Java web applications to AWS Elastic Beanstalk and within minutes access their applications running on AWS infrastructure services. The toolkit provides a Travel Log sample web application template that demonstrates the use of Amazon S3 and Amazon SNS. For more information about the AWS Toolkit for Eclipse, go to http://aws.amazon.com/eclipse.

Resources

There are several places you can go to get additional help when developing your Java applications.

Resource	Description
The AWS Java Development Forum	Post your questions and get feedback.
Java Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio

Topics

Elastic Beanstalk for .NET makes it easier to deploy, manage, and scale your ASP.NET web applications that use Amazon Web Services. Elastic Beanstalk for .NET is available to anyone who is developing or hosting a web application that uses IIS.

Get started now: To get started with a tutorial, you can go directly to Tutorial: How to Deploy a .NET Sample Application Using AWS Elastic Beanstalk (p. 613). In this tutorial, you will deploy a sample ASP.NET Web Application to an AWS Elastic Beanstalk application container.

The rest of this section presents instructions for creating, testing, deploying, and redeploying your ASP.NET web application to Elastic Beanstalk using the AWS Toolkit for Visual Studio. The second part explains how to manage and configure your applications and environments using the AWS Toolkit for Visual Studio. For more information about prerequisites, installation instructions, and running code samples, go to the

AWS Toolkit for Microsoft Visual Studio. This site also provides useful information about tools, how-to topics, and additional resources for ASP.NET developers.

If you are not a developer, but want to easily manage and configure your Elastic Beanstalk applications and environments, you can use the AWS Management Console. For information, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

Tutorial: How to Deploy a .NET Sample Application Using AWS Elastic Beanstalk

In this tutorial, you will learn how to deploy a .NET sample application to AWS Elastic Beanstalk (Elastic Beanstalk) using the AWS Toolkit for Visual Studio.

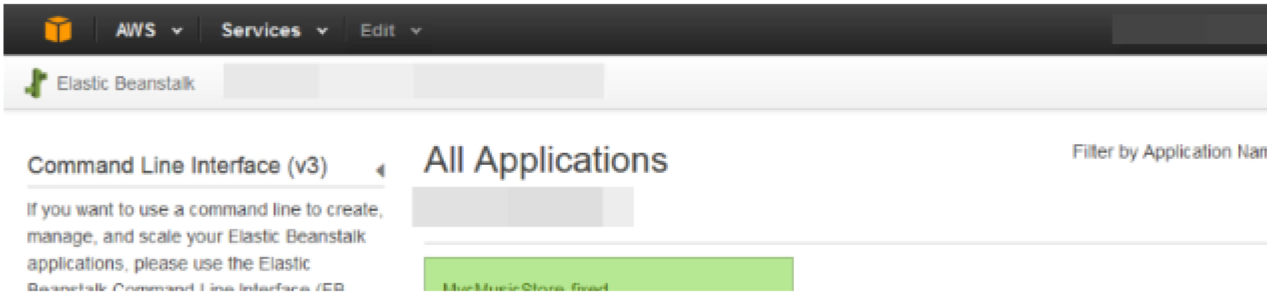
Note
This tutorial uses a sample ASP.NET Web application that you can download here. It also uses the Toolkit for Visual Studio and was tested using Visual Studio Professional 2012.

Step 1. Create the Environment

First, use the Create New Application wizard in the Elastic Beanstalk console to create the application environment.

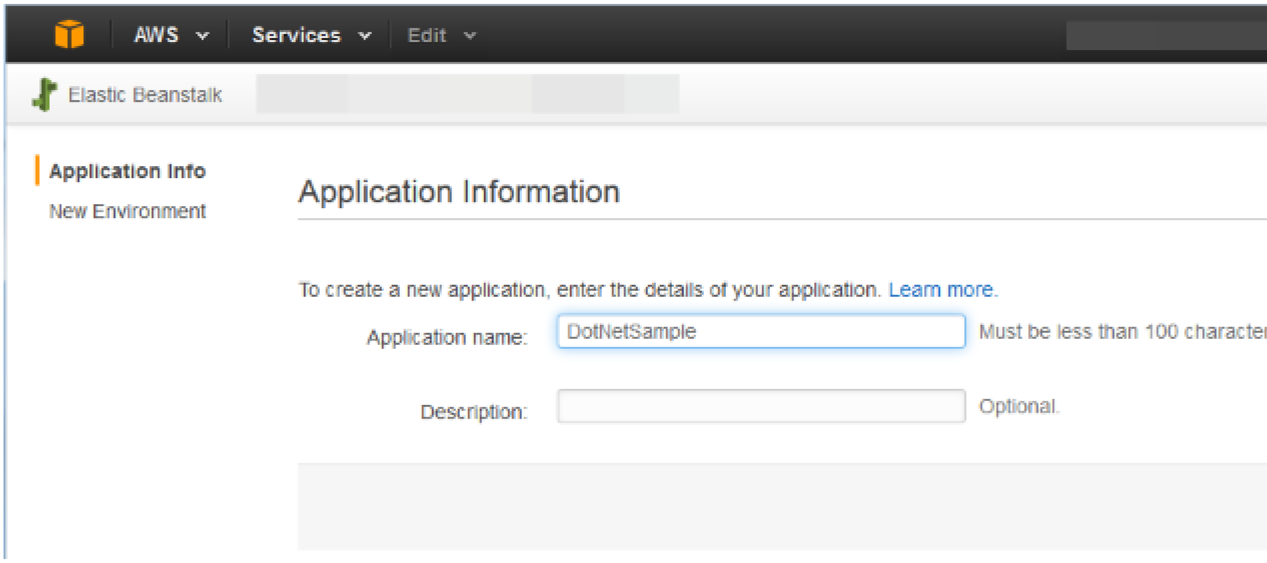
To create the environment

1. Go to console.aws.amazon.com/elasticbeanstalk to log into the Elastic Beanstalk console.
2. Choose **Create New Application**.



If you want to use a command line to create, manage, and scale your Elastic Beanstalk applications, please use the Elastic Beanstalk Command Line Interface (EB

3. On the **Application Information** page, enter an **Application name**, and then choose **Next**.



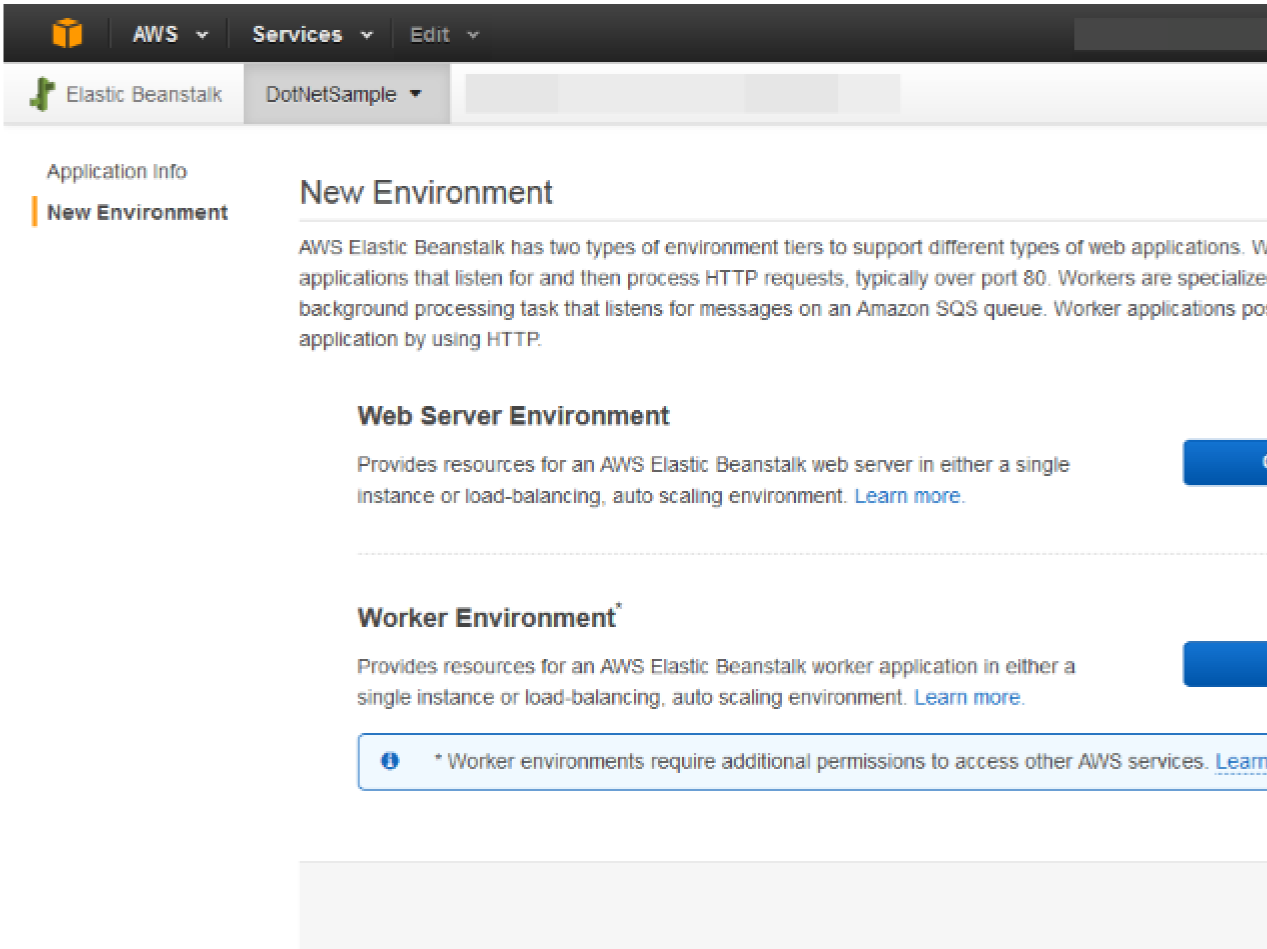
Application Information

To create a new application, enter the details of your application. Learn more.

Application name: DotNetSample Must be less than 100 characte

Description: Optional.

4. On the **New Environment** page, choose **Create web server**.



New Environment

AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications. W applications that listen for and then process HTTP requests, typically over port 80. Workers are specialize background processing task that listens for messages on an Amazon SQS queue. Worker applications po application by using HTTP.

Web Server Environment

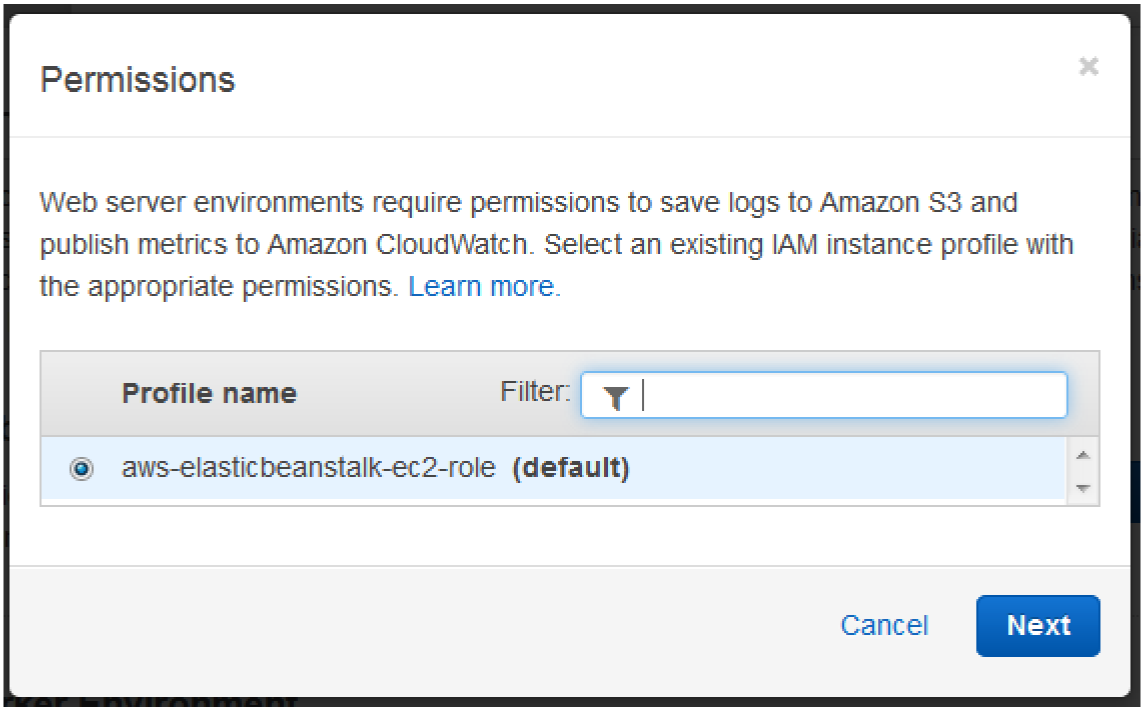
Provides resources for an AWS Elastic Beanstalk web server in either a single instance or load-balancing, auto scaling environment. Learn more.

Worker Environment*

Provides resources for an AWS Elastic Beanstalk worker application in either a single instance or load-balancing, auto scaling environment. Learn more.

* Worker environments require additional permissions to access other AWS services. Learn

5. In the **Permissions** dialog box, for **Profile name**, choose the default, and then choose **Next**.

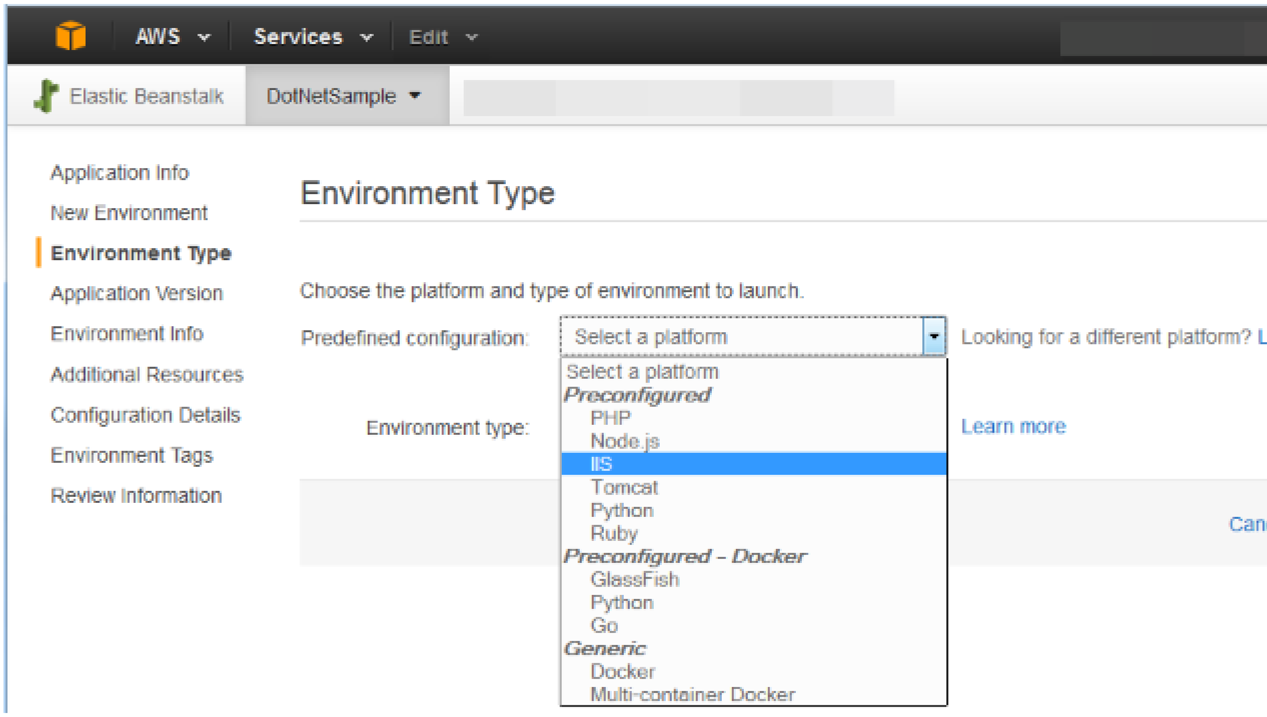


Permissions

Web server environments require permissions to save logs to Amazon S3 and publish metrics to Amazon CloudWatch. Select an existing IAM instance profile with the appropriate permissions. Learn more.

	Profile name
●	aws-elasticbeanstalk-ec2-role **(default)**

6. On the **Environment Type** page, for **Predefined configuration**, choose **IIS**.

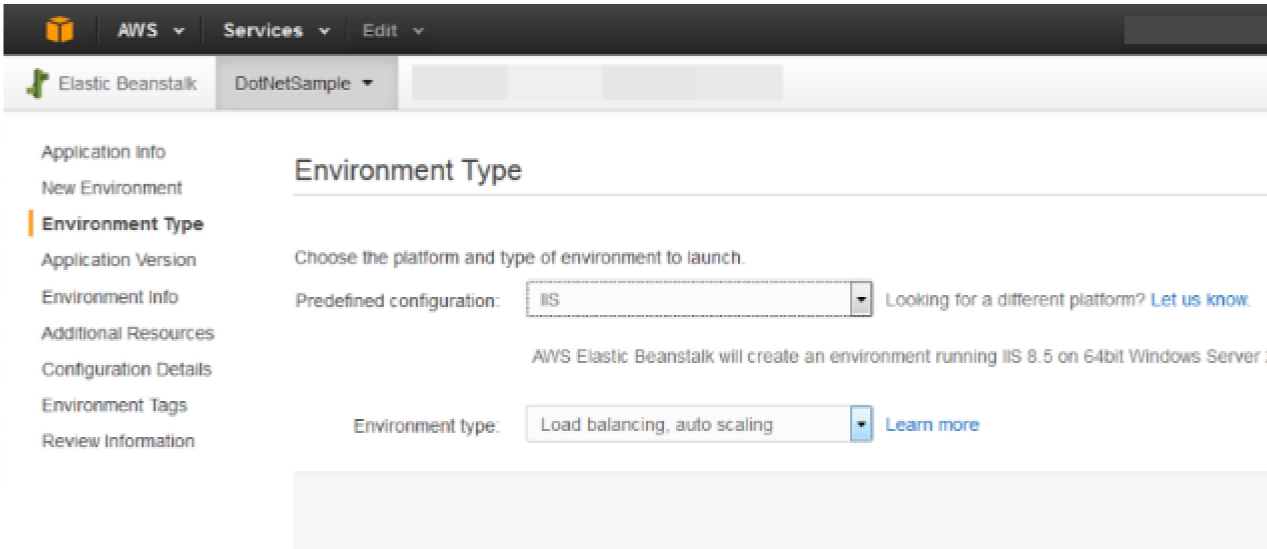


Environment Type

Choose the platform and type of environment to launch.

Predefined configuration: IIS

7. For **Environment type**, accept the default, **Load balancing, auto scaling**, and then choose **Next**.



Environment Type

Choose the platform and type of environment to launch.

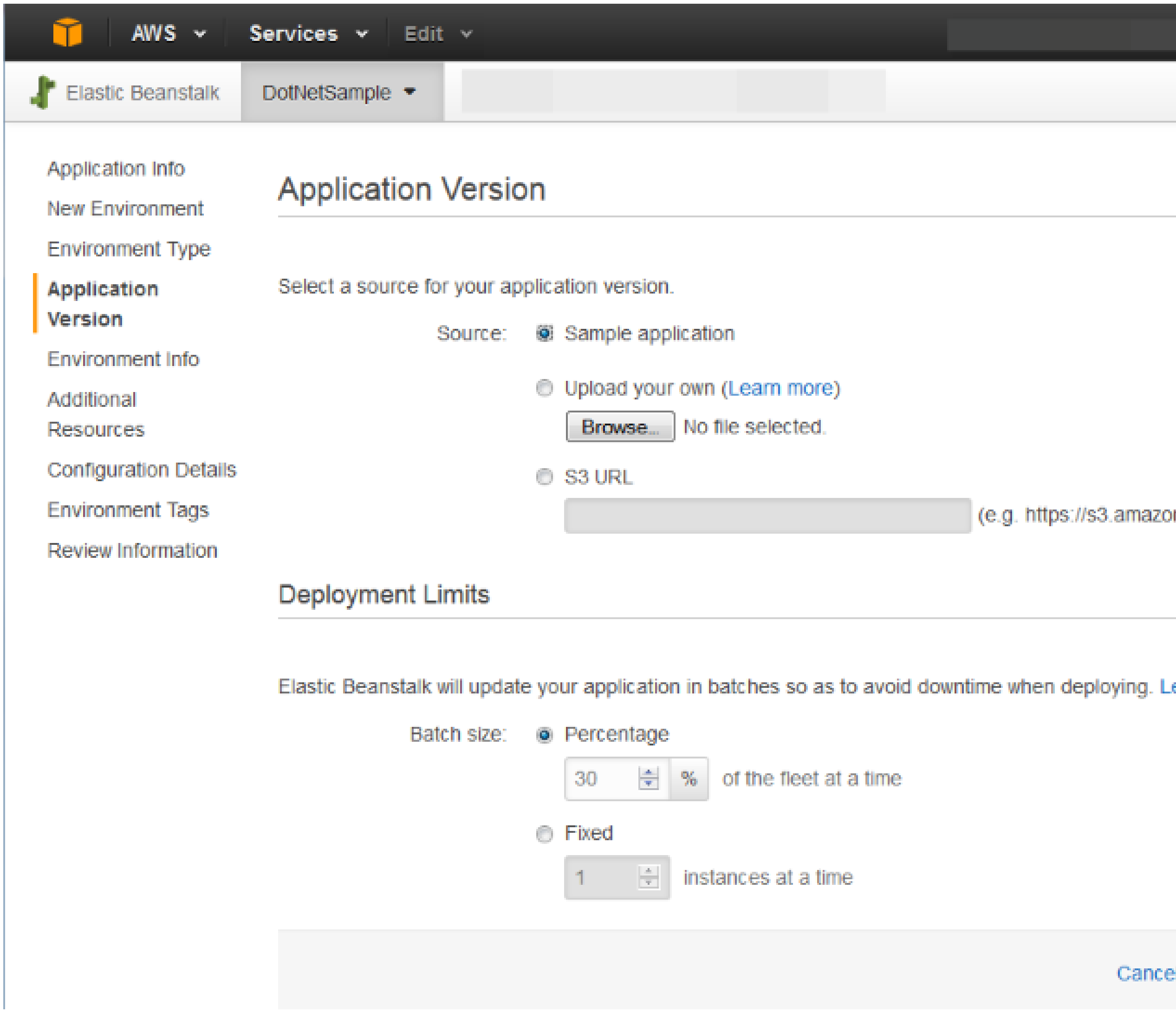
Predefined configuration: IIS

Looking for a different platform? Let us know.

AWS Elastic Beanstalk will create an environment running IIS 8.5 on 64bit Windows Server

Environment type: Load balancing, auto scaling

8. On the **Application Version** page, for **Source**, choose **Sample application**, and then choose **Next**.



Application Version

Select a source for your application version.

Source: ● Sample application

○ Upload your own (Learn more)

No file selected.

○ S3 URL

(e.g. https://s3.amazo

Deployment Limits

Elastic Beanstalk will update your application in batches so as to avoid downtime when deploying. Le

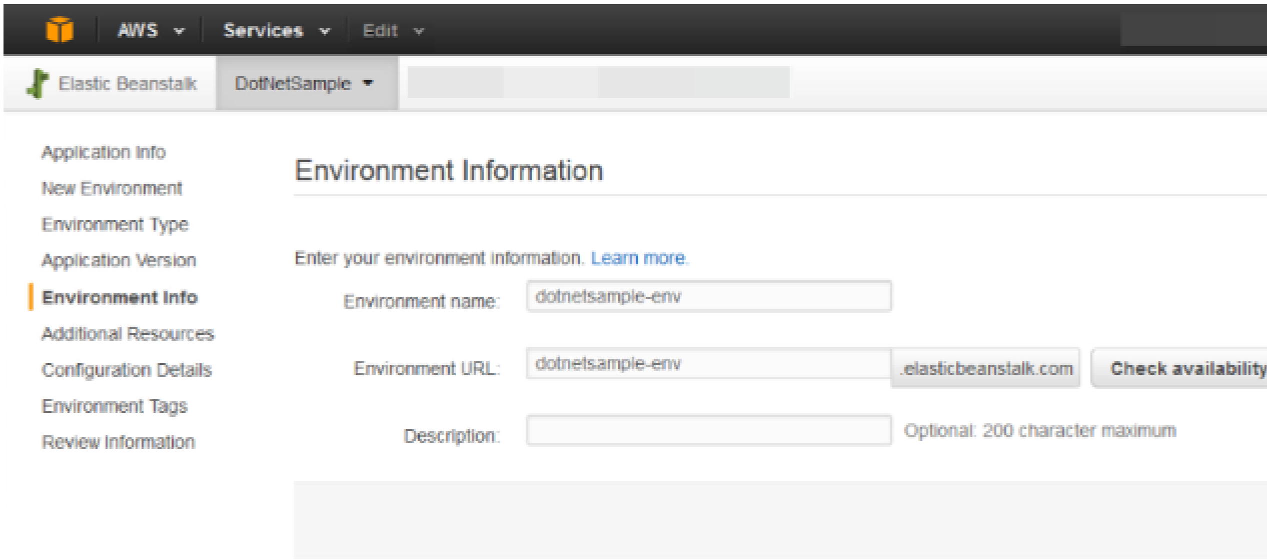
Batch size: ● Percentage

30 % of the fleet at a time

○ Fixed

1 instances at a time

9. On the **Environment Information** page, accept all defaults, and then choose **Next**.



Environment Information

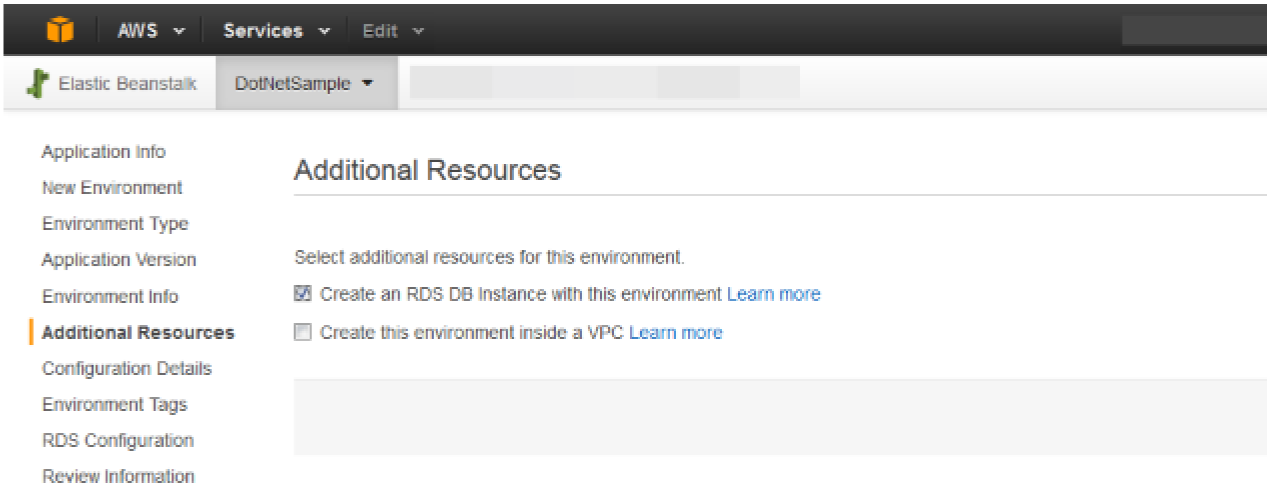
Enter your environment information. Learn more.

Environment name: dotnetsample-env

Environment URL: dotnetsample-env .elasticbeanstalk.com

Description: Optional: 200 character maximum

10. On the **Additional Resources** page, choose **Create an RDS DB instance with this environment**, and then choose **Next**.



Additional Resources

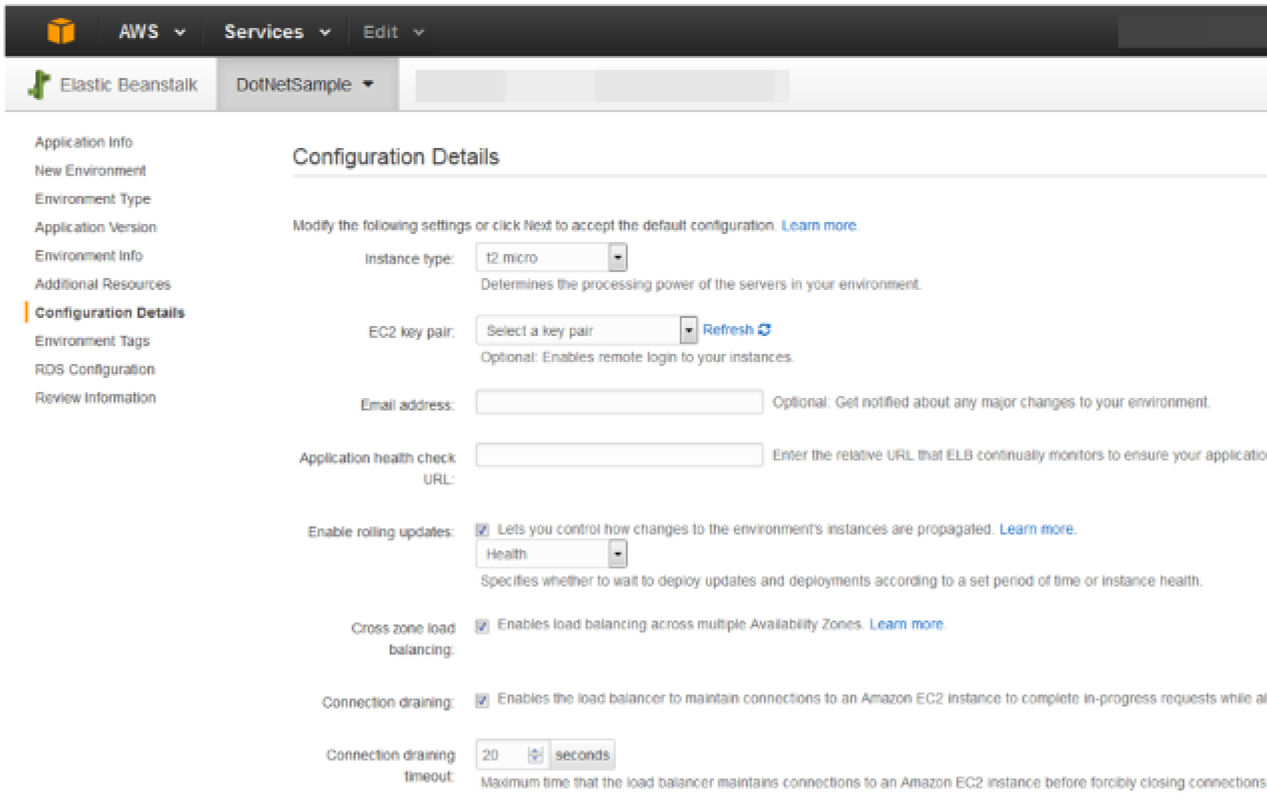
Select additional resources for this environment.

☑ Create an RDS DB Instance with this environment Learn more

□ Create this environment inside a VPC Learn more

11. On the **Configuration Details** page, for **Instance type**, choose `t2.micro`.

    Accept thedefault values for the other fields, and then choose **Next**.



Configuration Details

Modify the following settings or click Next to accept the default configuration. Learn more.

Instance type: t2.micro
Determines the processing power of the servers in your environment.

EC2 key pair: Select a key pair — Refresh
Optional: Enables remote login to your instances.

Email address: Optional: Get notified about any major changes to your environment.

Application health check URL: Enter the relative URL that ELB continually monitors to ensure your applicatio

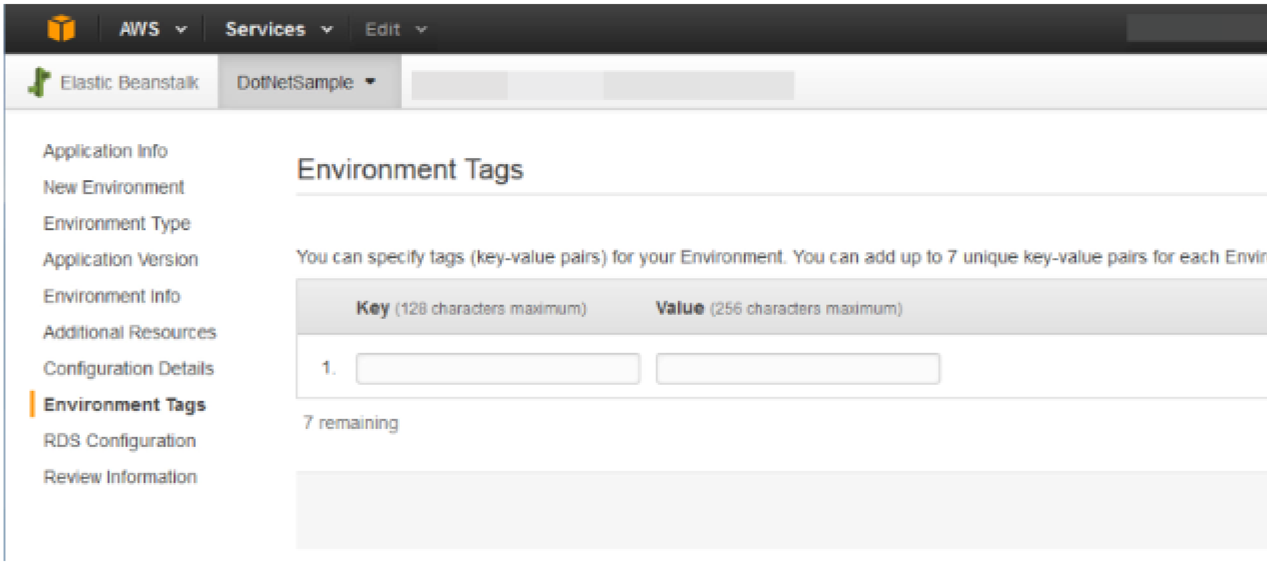
Enable rolling updates: ☑ Lets you control how changes to the environment's instances are propagated. Learn more.
Health
Specifies whether to wait to deploy updates and deployments according to a set period of time or instance health.

Cross zone load balancing: ☑ Enables load balancing across multiple Availability Zones. Learn more.

Connection draining: ☑ Enables the load balancer to maintain connections to an Amazon EC2 instance to complete in-progress requests while al

Connection draining timeout: 20 seconds
Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections

12. On the **Environment Tags** page, leave both the **Key** and the **Value** fields blank, and then choose **Next**.



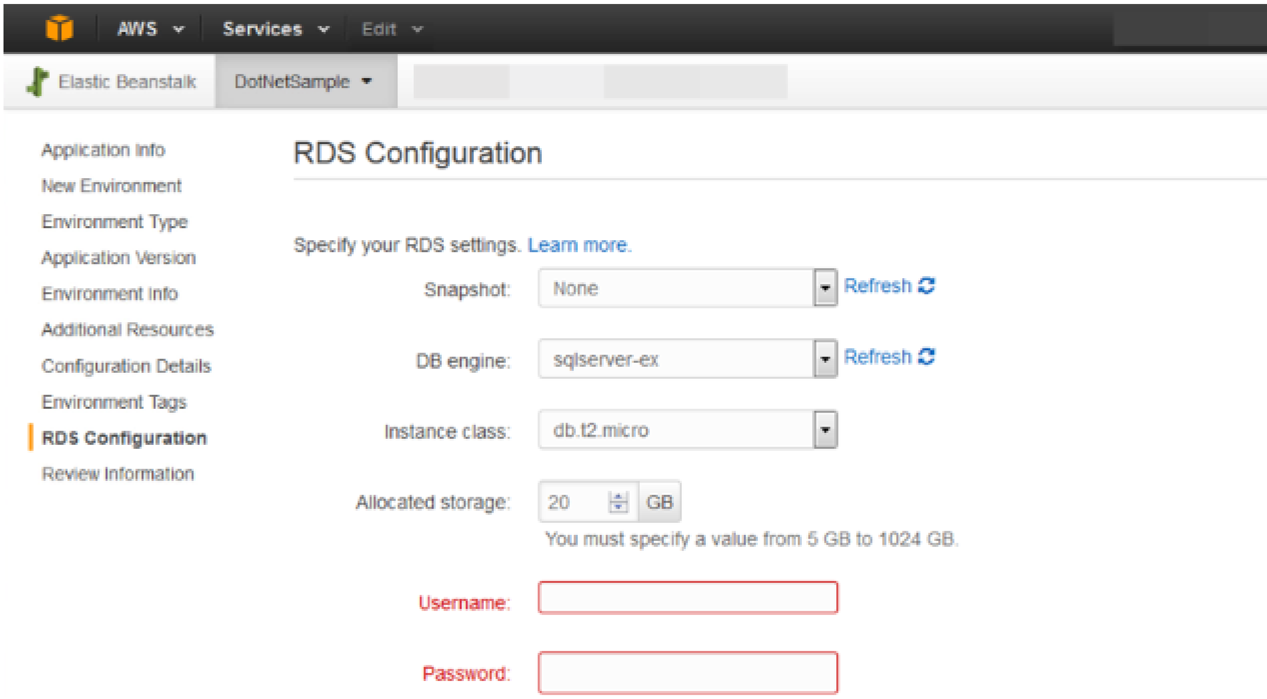
Environment Tags

You can specify tags (key-value pairs) for your Environment. You can add up to 7 unique key-value pairs for each Envir

	Key (128 characters maximum)	Value (256 characters maximum)
1.		

7 remaining

13. On the **RDS Configuration** page, for **DB engine**, choose **sqlserver-ex**. For **Instance class**, choose **db.t2.micro**, and then increase the **Allocated storage** to **`20 GB`**.



RDS Configuration

Specify your RDS settings. Learn more.

Snapshot: None

DB engine: sqlserver-ex

Instance class: db.t2.micro

Allocated storage: 20 GB

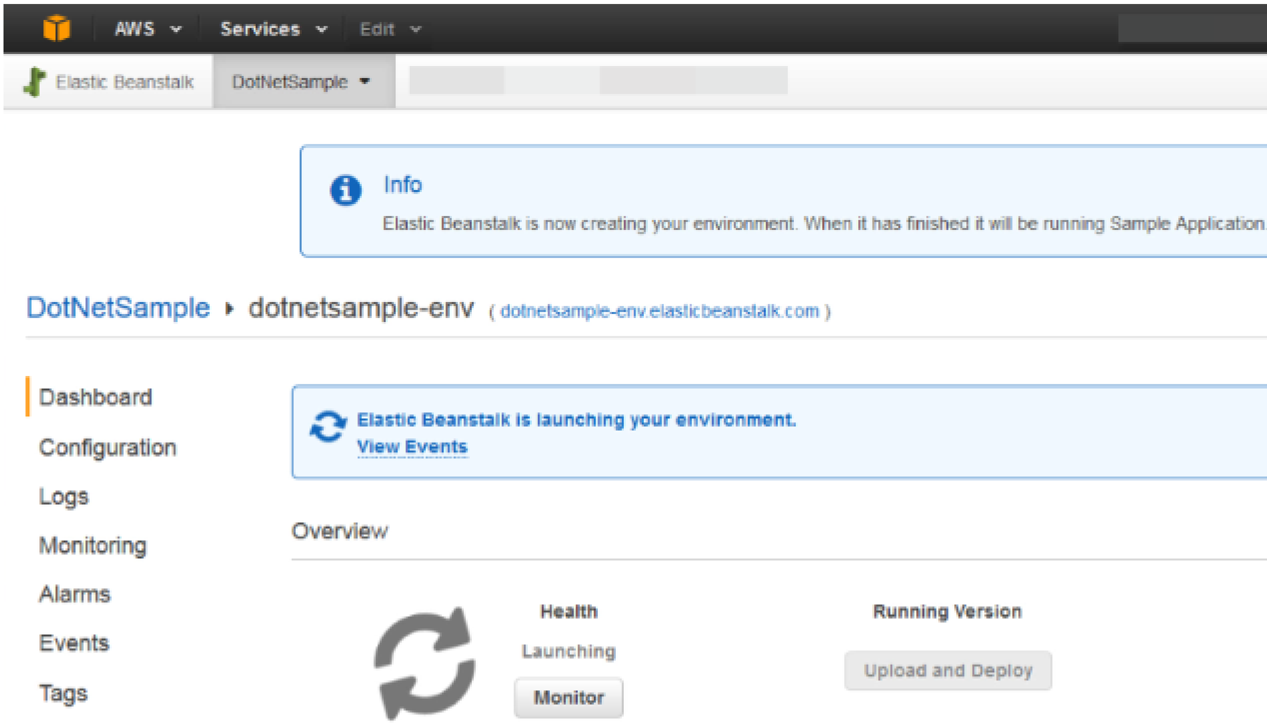
You must specify a value from 5 GB to 1024 GB.

Username:

Password:

14. Create a **Username** and **Password**, and then choose **Next**.
15. On the **Review Information** page, review the settings, and then choose **Launch**.

    To check launch status, see the **Dashboard** page in the Elastic Beanstalk console.



Info

Elastic Beanstalk is now creating your environment. When it has finished it will be running Sample Application

DotNetSample ▸ dotnetsample-env (dotnetsample-env.elasticbeanstalk.com)

Elastic Beanstalk is launching your environment.
View Events

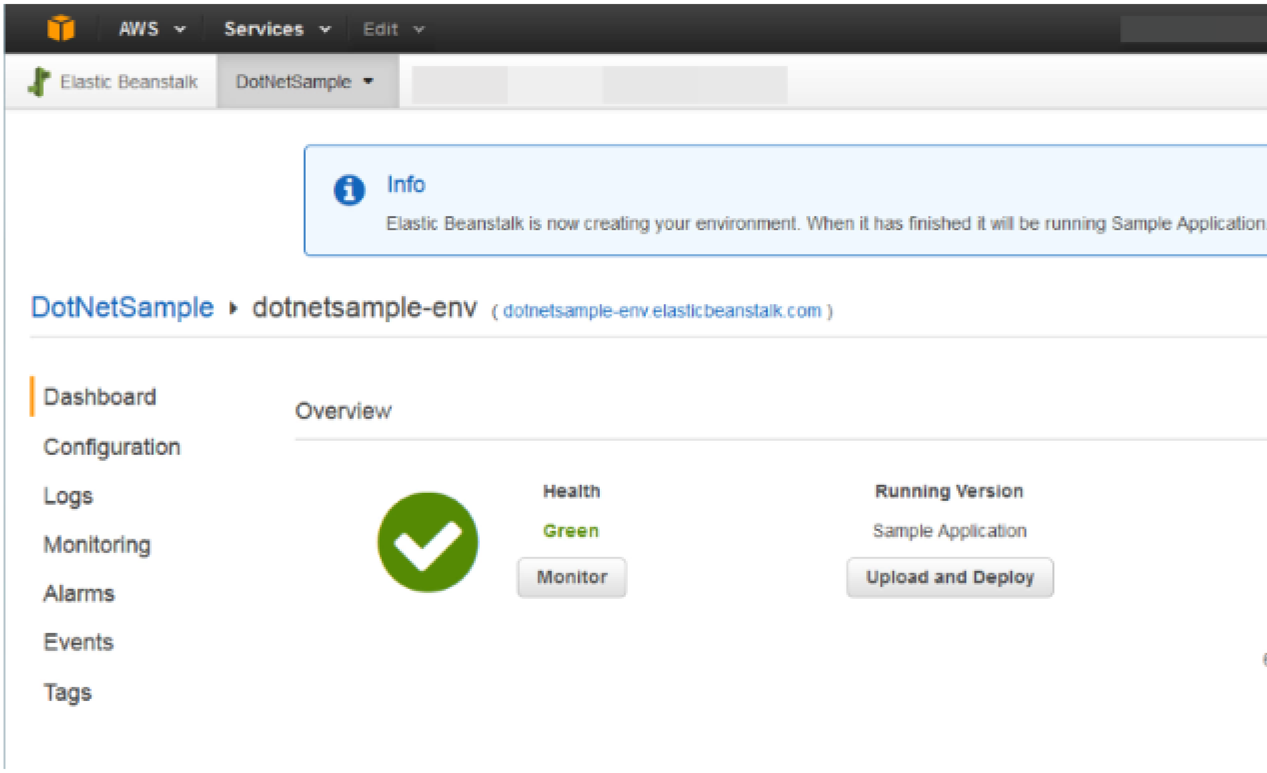
Overview

Step 2. Publish Your Application to Elastic Beanstalk

Use the AWS Toolkit for Visual Studio to publish your application to Elastic Beanstalk.

To publish your application to Elastic Beanstalk

1. Ensure that your environment launched successfully by checking the **Health** status in the Elastic Beanstalk console. It should be **Green**.



Info

Elastic Beanstalk is now creating your environment. When it has finished it will be running Sample Application

DotNetSample ▸ dotnetsample-env (dotnetsample-env.elasticbeanstalk.com)

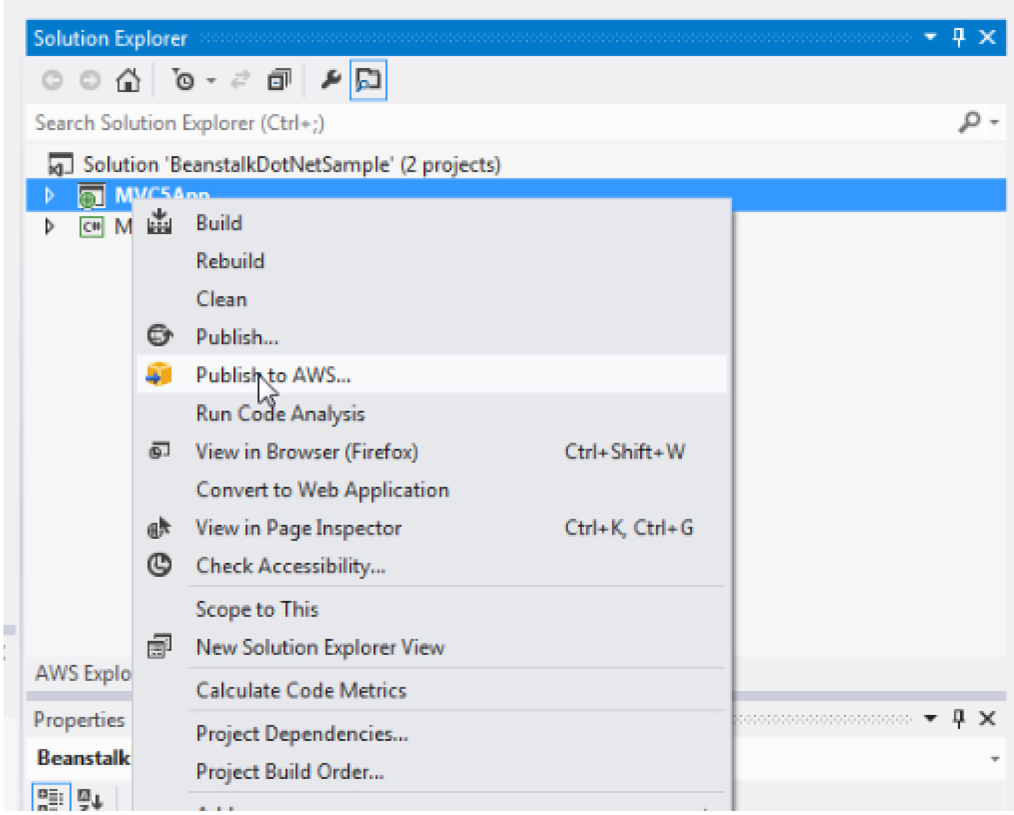
Health: Green

Running Version: Sample Application

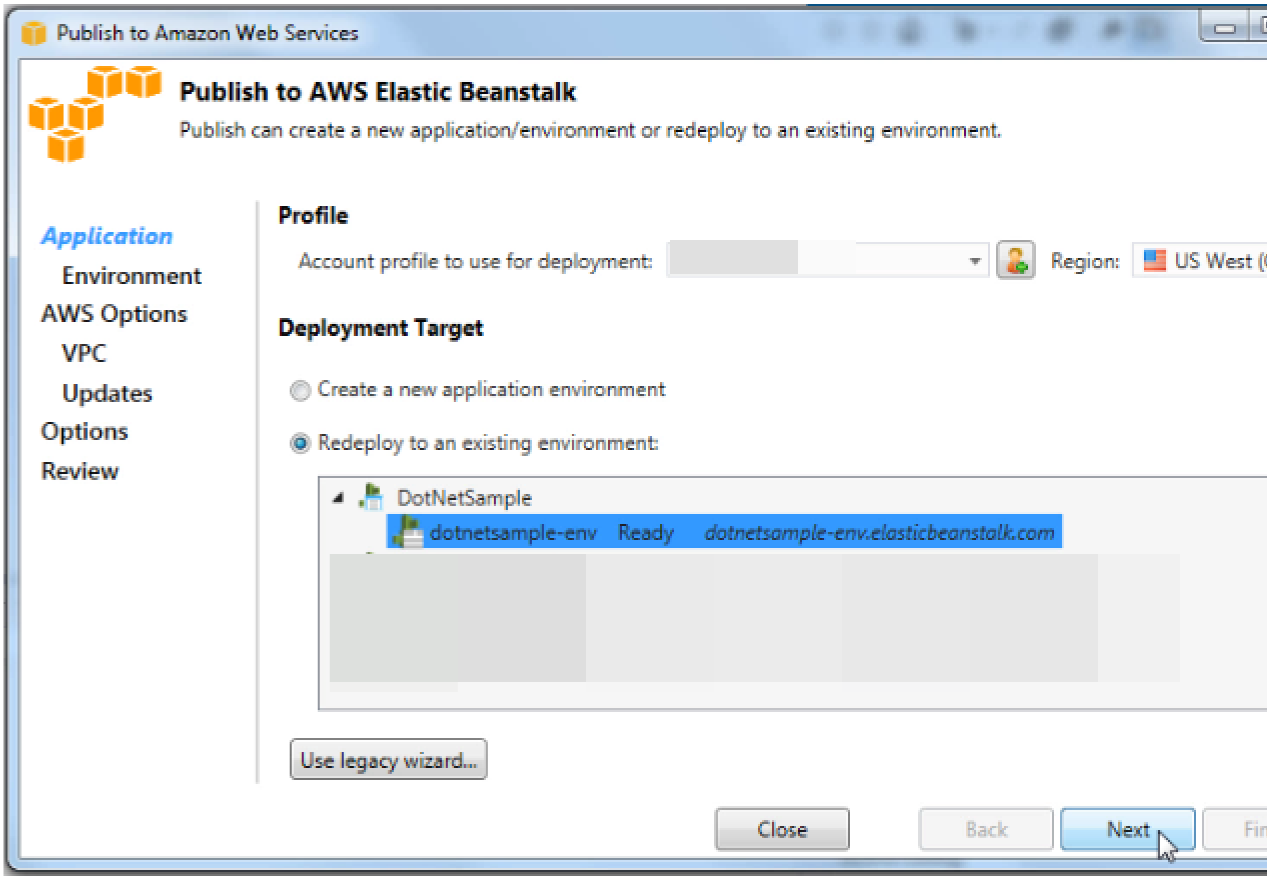
2. In Visual Studio, open **BeanstalkDotNetSample.sln**.

   Note
   If you haven't done so already, you can get the sample here.

3. On the **View** menu, choose **Solution Explorer**.
4. Expand **Solution ‘BeanstalkDotNetSample’ (2 projects)**.
5. Open the context (right-click) menu for **MVC5App**, and then choose **Publish to AWS**.



6. On the **Publish to AWS Elastic Beanstalk** page, for **Deployment Target**, choose the environment that you just created, and then choose **Next**.



Publish to AWS Elastic Beanstalk

Publish can create a new application/environment or redeploy to an existing environment.

Profile

Account profile to use for deployment: Region: US West (

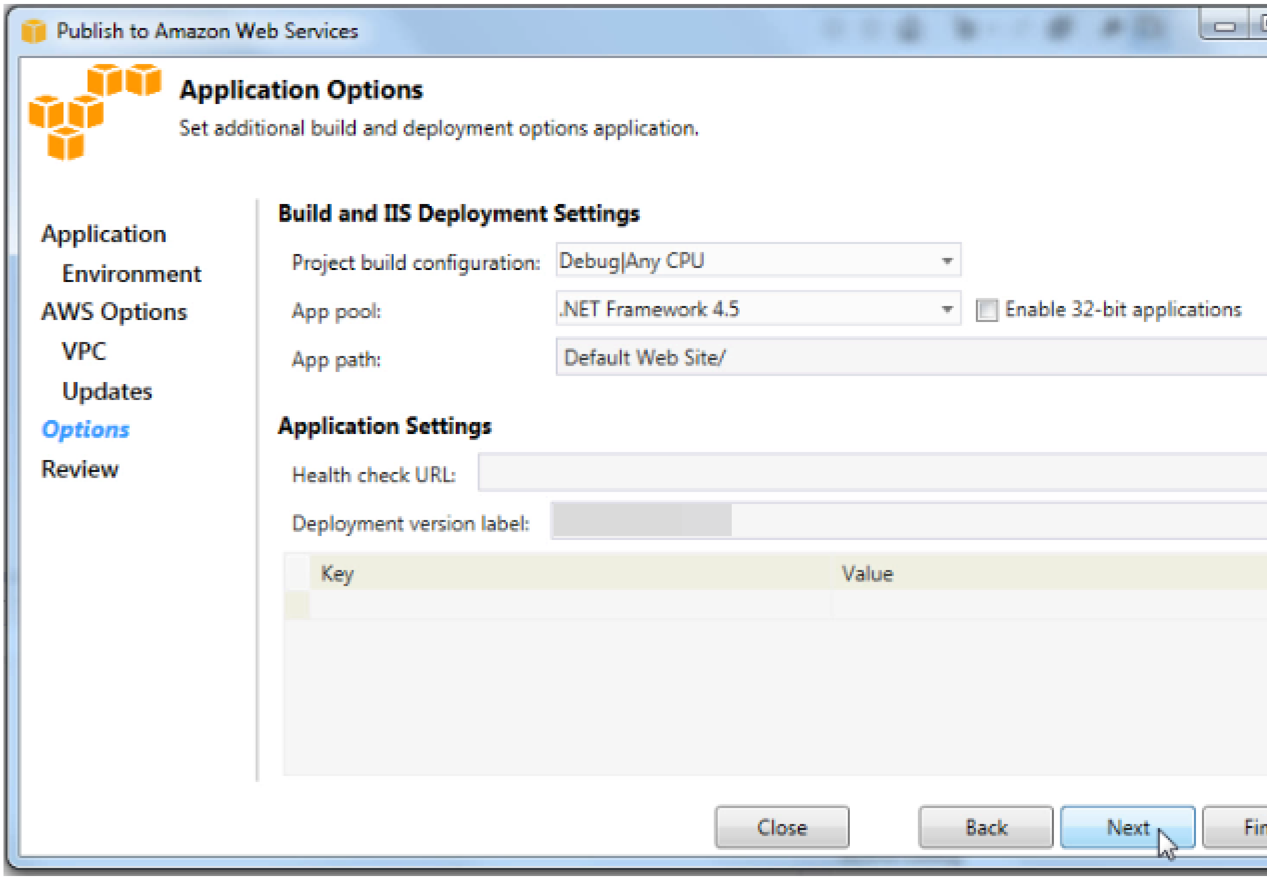
Deployment Target

○ Create a new application environment

● Redeploy to an existing environment:

DotNetSample
dotnetsample-env Ready *dotnetsample-env.elasticbeanstalk.com*

7. On the **Application Options** page, accept all of the defaults, and then choose **Next**.



Publish to Amazon Web Services

Application Options

Set additional build and deployment options application.

Build and IIS Deployment Settings

Project build configuration: Debug|Any CPU

App pool: .NET Framework 4.5 □ Enable 32-bit applications

App path: Default Web Site/

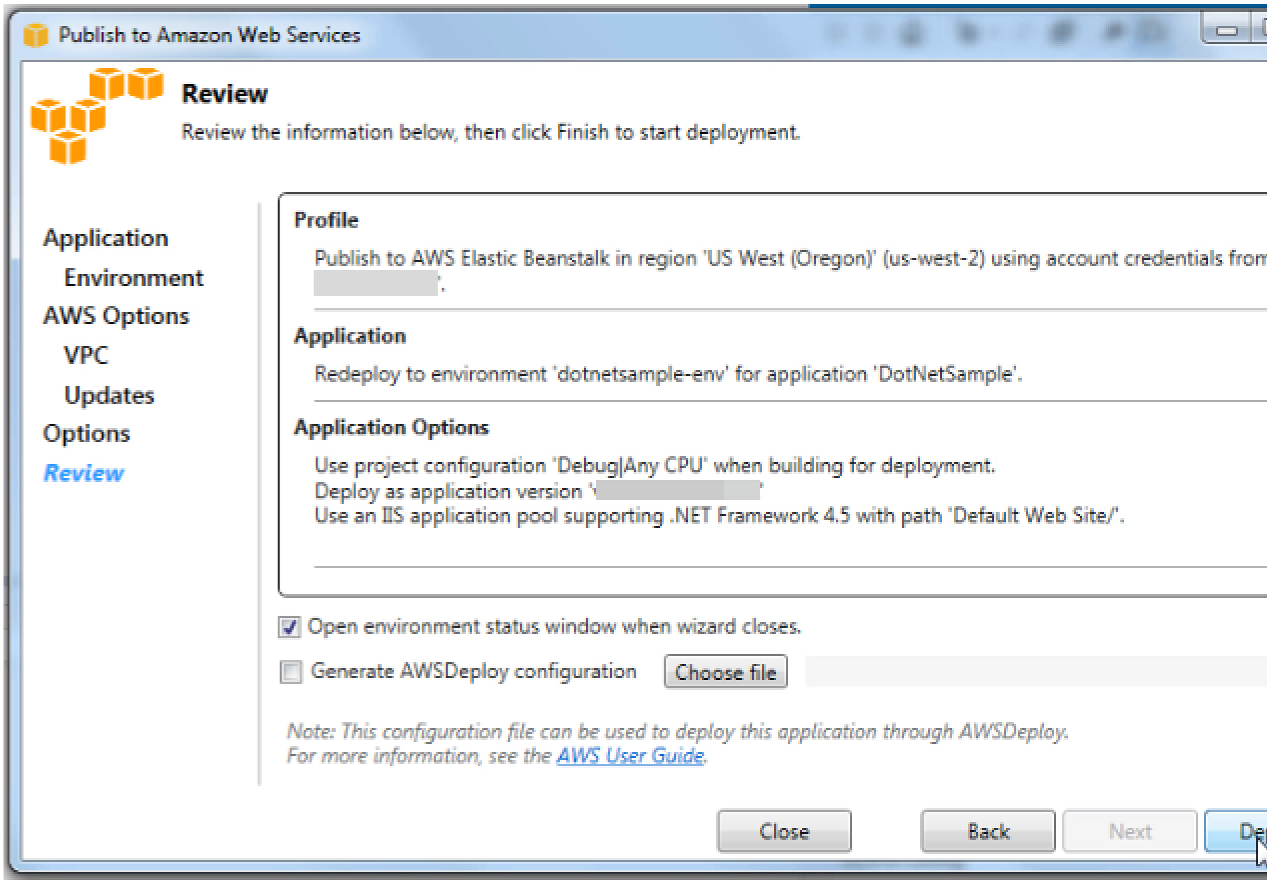
Application Settings

Health check URL:

Deployment version label:

Key	Value

8. On the **Review** page, choose **Deploy**.



Review

Review the information below, then click Finish to start deployment.

Profile

Publish to AWS Elastic Beanstalk in region 'US West (Oregon)' (us-west-2) using account credentials from [redacted].

Application

Redeploy to environment 'dotnetsample-env' for application 'DotNetSample'.

Application Options

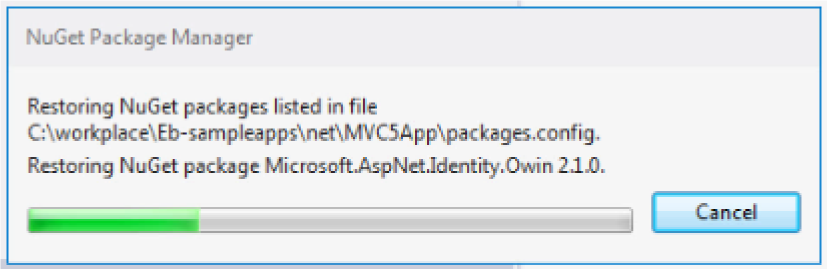
Use project configuration 'Debug|Any CPU' when building for deployment.
Deploy as application version '[redacted]'
Use an IIS application pool supporting .NET Framework 4.5 with path 'Default Web Site/'.

☑ Open environment status window when wizard closes.

☐ Generate AWSDeploy configuration Choose file

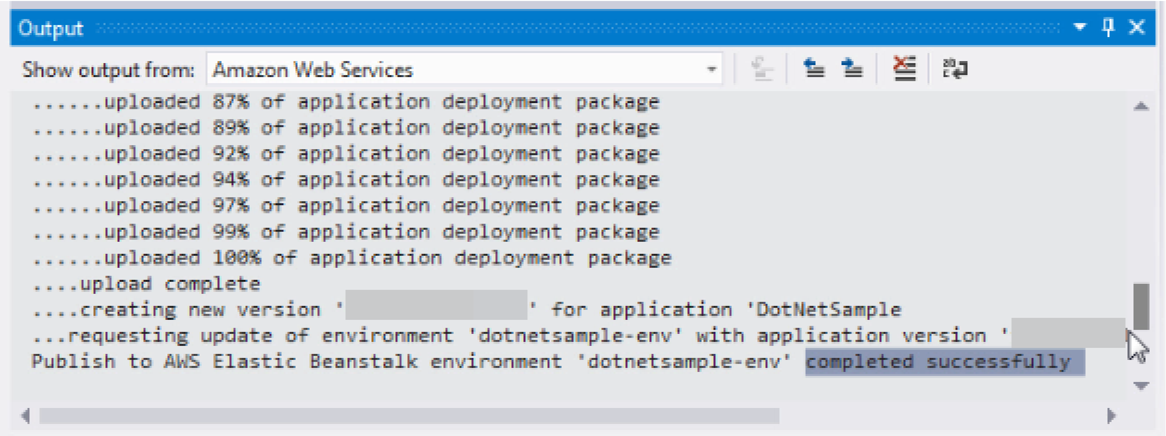
Note: This configuration file can be used to deploy this application through AWSDeploy. For more information, see the AWS User Guide.

9. If you want to monitor deployment status, use the **NuGet Package Manager** in Visual Studio.



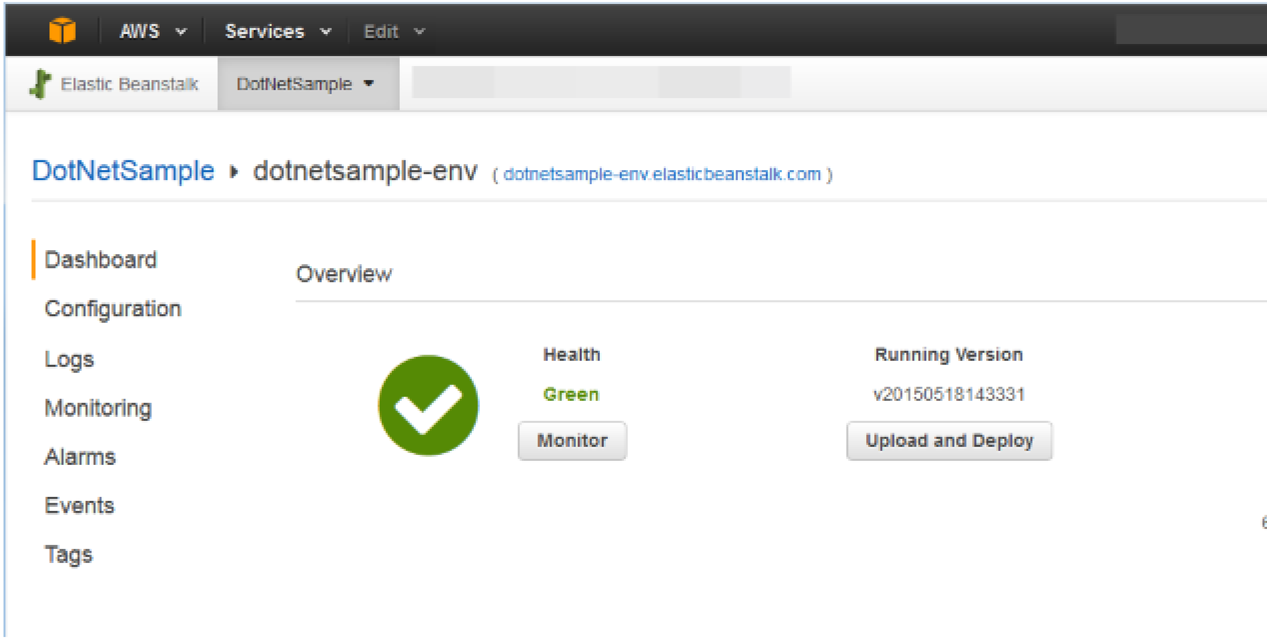
Restoring NuGet packages listed in file
C:\workplace\Eb-sampleapps\net\MVC5App\packages.config.
Restoring NuGet package Microsoft.AspNet.Identity.Owin 2.1.0.

When the application has successfully been deployed, the **Output** box displays **completed successfully**.



```
......uploaded 87% of application deployment package
......uploaded 89% of application deployment package
......uploaded 92% of application deployment package
......uploaded 94% of application deployment package
......uploaded 97% of application deployment package
......uploaded 99% of application deployment package
......uploaded 100% of application deployment package
....upload complete
....creating new version '[redacted]' for application 'DotNetSample
...requesting update of environment 'dotnetsample-env' with application version '[redacted]
Publish to AWS Elastic Beanstalk environment 'dotnetsample-env' completed successfully
```

10. Return to the Elastic Beanstalk console and choose the name of the application, which appears next to the environment name).

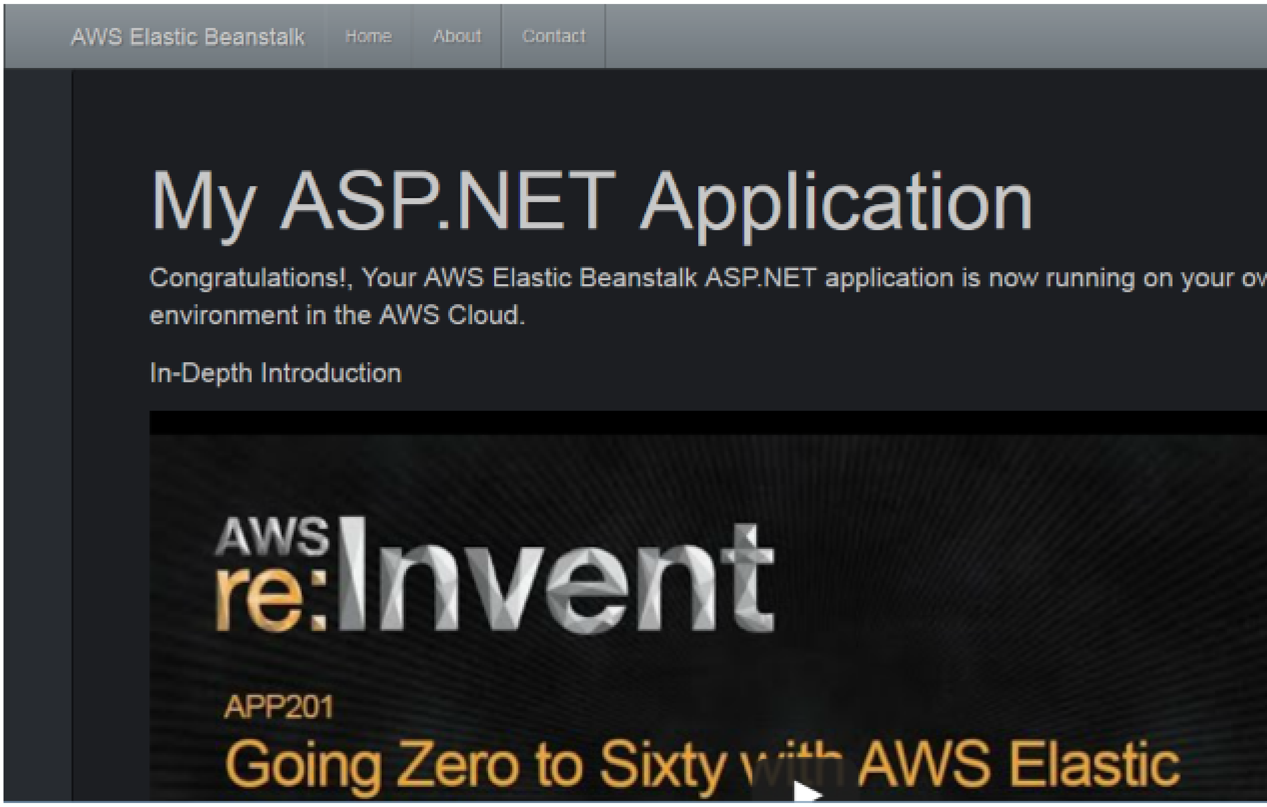


DotNetSample ▸ dotnetsample-env (dotnetsample-env.elasticbeanstalk.com)

Health
Green

Running Version
v20150518143331

Your ASP.NET application opens in a new tab.



My ASP.NET Application

Congratulations!, Your AWS Elastic Beanstalk ASP.NET application is now running on your ov
environment in the AWS Cloud.

In-Depth Introduction

Going Zero to Sixty with AWS Elastic

Step 3. Clean Up Your AWS Resources

After your application has deployed successfully, learn more about Elastic Beanstalk by watching the video in the application.

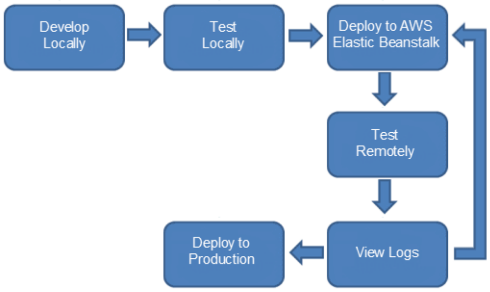
When you are finished, you can clean up your environment and delete any unused AWS resources such as Amazon RDS databases, Amazon S3 buckets, or your Elastic Beanstalk environment.

For more information, see Creating and Deploying Elastic Beanstalk Applications in .NET Using AWS Toolkit for Visual Studio (p. 612), the AWS .NET Development Blog, or the AWS Application Management Blog.

Develop, Test, and Deploy

Topics

The following diagram of a typical software development life cycle includes deploying your application to Elastic Beanstalk.



After developing and testing your application locally, you will typically deploy your application to AWS Elastic Beanstalk. After deployment, your application will be live at a URL such as http://myexampleapp-wpams3yrvj.elasticbeanstalk.com. Because your application is live, you should consider setting up multiple environments, such as a testing environment and a production environment. You can use the AWS Toolkit for Visual Studio if you want to set up different AWS accounts for testing, staging, and production. For more information about managing multiple accounts, see Managing Multiple Accounts (p. 649).

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. You can point your domain name to the Amazon Route 53 CNAME <*yourappname*>.elasticbeanstalk.com. Contact your DNS provider to set this up. For information about how to map your root domain to your Elastic Load Balancer, see Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295).

After you remotely test and debug your Elastic Beanstalk application, you can make any updates and then redeploy to Elastic Beanstalk. After you are satisfied with all of your changes, you can upload the latest version to your production environment. The following sections explain each stage of the software development life cycle.

Create a Project

Visual Studio provides templates for different programming languages and application types. You can start with any of these templates. The AWS Toolkit for Visual Studio also provides three project templates that bootstrap development of your application: AWS Console Project, AWS Web Project, and AWS Empty Project. For this example, you'll create a new ASP.NET Web Application.

To create a new ASP.NET Web Application project

1. In Visual Studio, on the **File** menu, click **New** and then click **Project**.
2. In the **New Project** dialog box, click **Installed Templates**, click **Visual C#**, and then click **Web**. Click **ASP.NET Empty Web Application**, type a project name, and then click **OK**.

To run a project

Do one of the following:

- Press **F5**.
- Select **Start Debugging** from the **Debug** menu.

Test Locally

Visual Studio makes it easy for you to test your application locally. To test or run ASP.NET web applications, you need a web server. Visual Studio offers several options, such as Internet Information Services (IIS), IIS Express, or the built-in Visual Studio Development Server. To learn about each of these options and to decide which one is best for you, go to Web Servers in Visual Studio for ASP.NET Web Projects .

Deploy to AWS Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk.

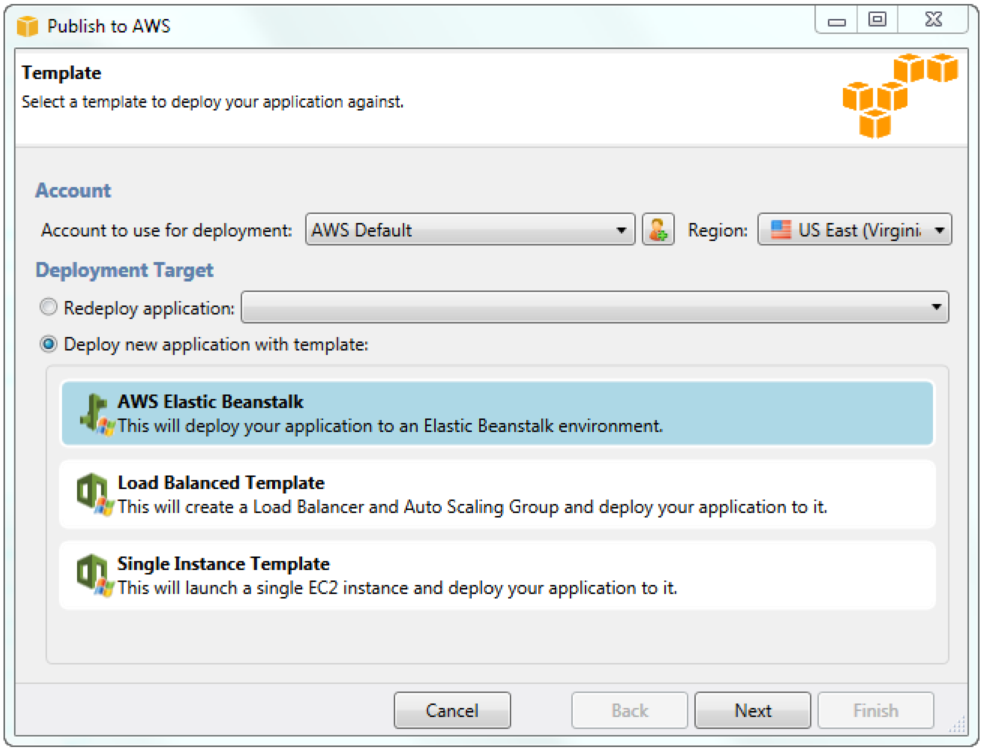
To deploy your application to Elastic Beanstalk using the AWS Toolkit for Visual Studio

1. In **Solution Explorer**, right-click your application and then select **Publish to AWS**.
2. In the **Publish to AWS** wizard, enter your account information.

   a. For **AWS account to use for deployment**, select your account or select **Other** to enter new account information.

   b. For **Region**, select the region where you want to deploy your application. For information about this product's regions, go to Regions and Endpoints in the *Amazon Web Services General Reference*. If you select a region that is not supported by Elastic Beanstalk, then the option to deploy to Elastic Beanstalk will become unavailable.

   c. Click **Deploy new application with template** and select **Elastic Beanstalk**. Then click **Next**.



Publish to AWS

Template

Select a template to deploy your application against.

Account

Account to use for deployment: AWS Default

Region: US East (Virgini

Deployment Target

○ Redeploy application:

● Deploy new application with template:

AWS Elastic Beanstalk
This will deploy your application to an Elastic Beanstalk environment.

Load Balanced Template
This will create a Load Balancer and Auto Scaling Group and deploy your application to it.

Single Instance Template
This will launch a single EC2 instance and deploy your application to it.

3. On the **Application** page, enter your application details.

   a. For **Name**, type the name of the application.

b. For **Description**, type a description of the application. This step is optional.

c. The version label of the application automatically appears in the **Deployment version label**

d. Select **Deploy application incrementally** to deploy only the changed files. An incremental deployment is faster because you are updating only the files that changed instead of all the files. If you choose this option, an application version will be set from the Git commit ID. If you choose to not deploy your application incrementally, then you can update the version label in the **Deployment version label** box.

e. Click **Next**.

4. On the **Environment** page, describe your environment details.

a. Select **Create a new environment for this application**.

b. For **Name**, type a name for your environment.

c. For **Description**, characterize your environment. This step is optional.

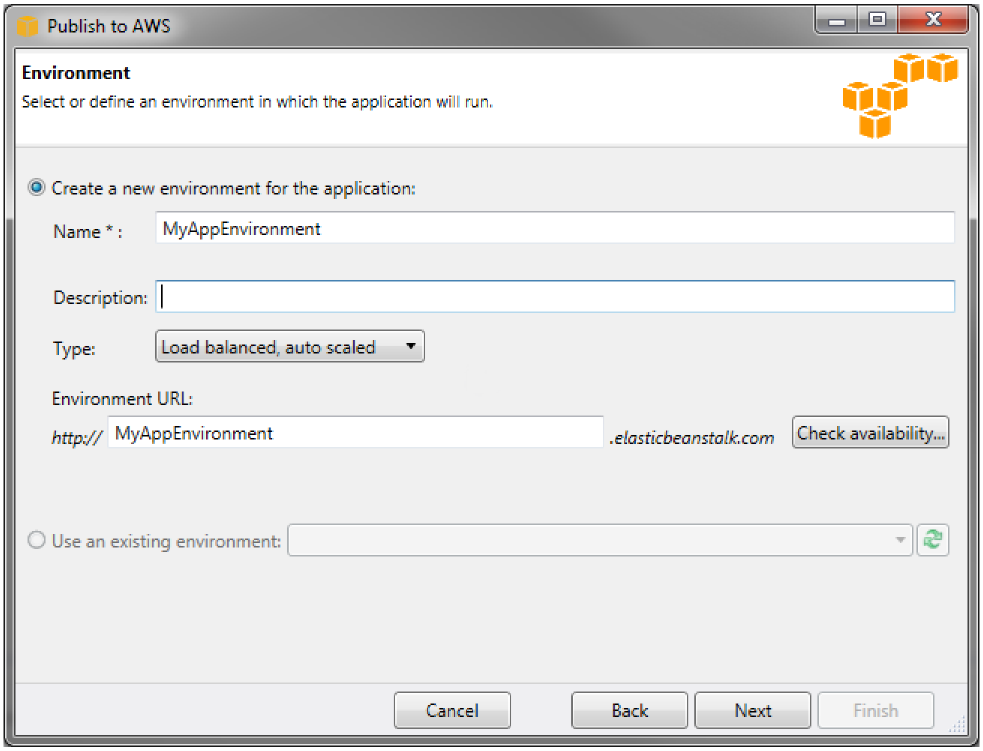
d. Select the **Type** of environment that you want.

You can select either **Load balanced, auto scaled** or a **Single instance** environment. For more information, see Environment Types (p. 68).

> **Note**
> For single-instance environments, load balancing, autoscaling, and the health check URL settings don't apply.

e. The environment URL automatically appears in the **Environment URL** once you move your cursor to that box.

f. Click **Check availability** to make sure the environment URL is available.



Publish to AWS

Environment

Select or define an environment in which the application will run.

● Create a new environment for the application:

Name * : MyAppEnvironment

Description:

Type: Load balanced, auto scaled

Environment URL:

http:// MyAppEnvironment .elasticbeanstalk.com

○ Use an existing environment:

g. Click **Next**.

5. On the **AWS Options** page, configure additional options and security information for your deployment.

   a. For **Container Type**, select **64bit Windows Server 2012 running IIS 8** or **64bit Windows Server 2008 running IIS 7.5**.

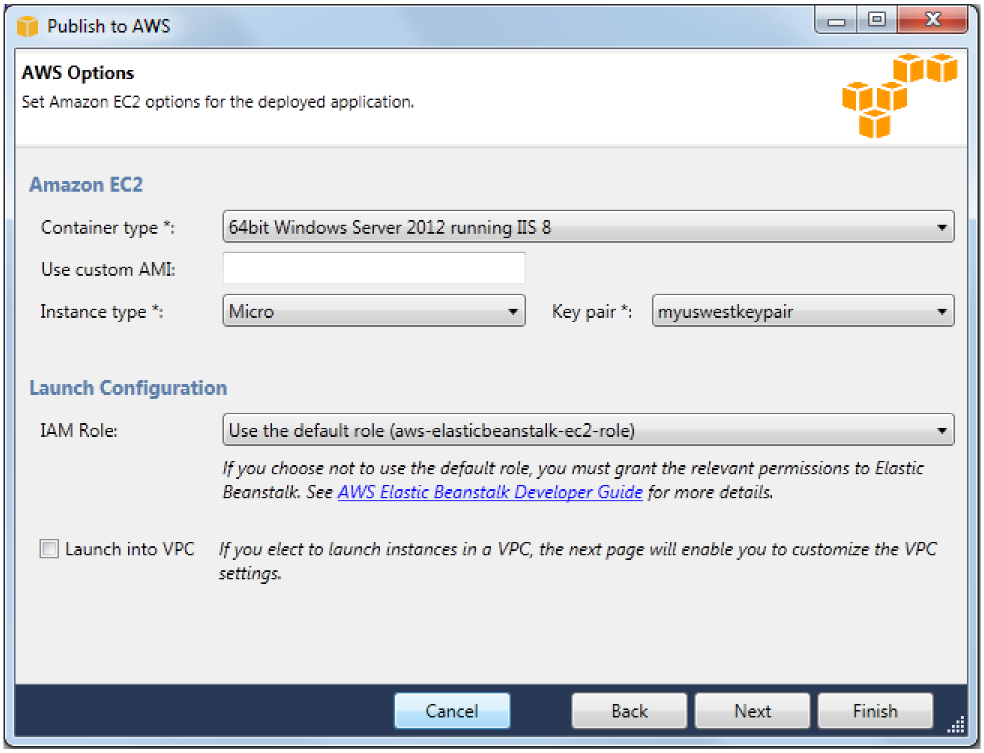
   b. For **Instance Type**, select **Micro**.

   c. For **Key pair**, select **Create new key pair**. Type a name for the new key pair—in this example, we use `myuswestkeypair`—and then click **OK**. A key pair enables remote-desktop access to your Amazon EC2 instances. For more information on Amazon EC2 key pairs, see Using Credentials in the *Amazon Elastic Compute Cloud User Guide*.

   d. Select an instance profile.

      If you do not have an instance profile, select **Create a default instance profile**. For information about using instance profiles with Elastic Beanstalk, see Using IAM Roles with Elastic Beanstalk (p. 332).

   e. If you are using a nonlegacy container, you have the option to create your environment inside an existing VPC; to do this, click **Launch into VPC**. You can configure the VPC information on the next page. For more information about Amazon VPC, go to Amazon Virtual Private Cloud (Amazon VPC). For a list of supported nonlegacy container types, see Why are some container types marked legacy? (p. 92).



AWS Options

Set Amazon EC2 options for the deployed application.

Amazon EC2

Container type *: 64bit Windows Server 2012 running IIS 8

Use custom AMI:

Instance type *: Micro

Key pair *: myuswestkeypair

Launch Configuration

IAM Role: Use the default role (aws-elasticbeanstalk-ec2-role)

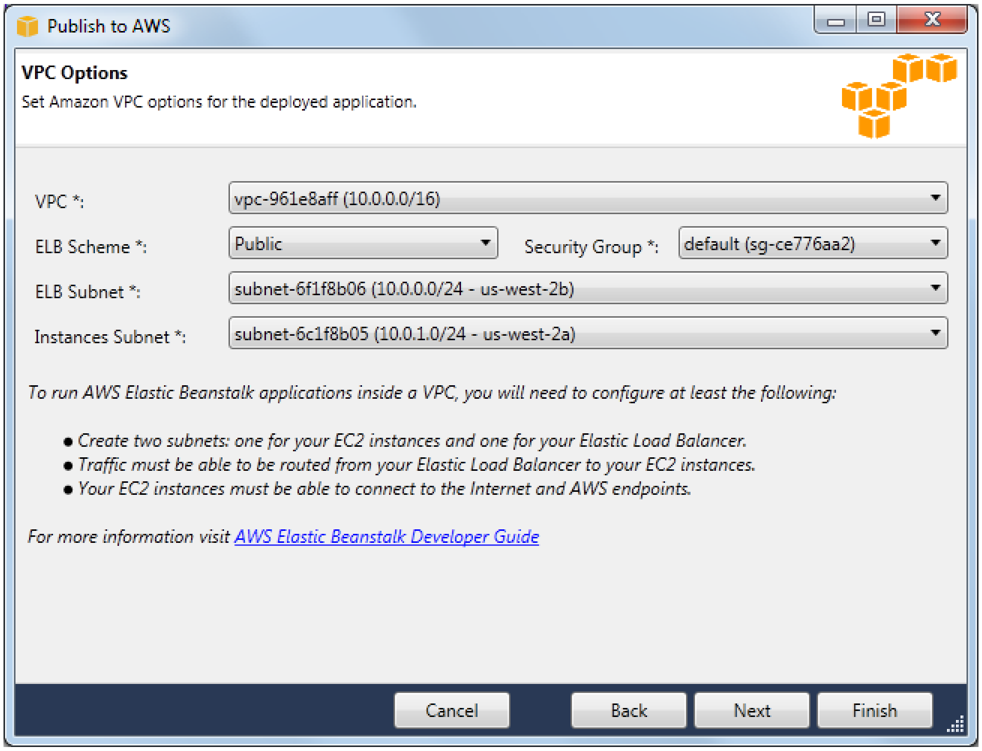
If you choose not to use the default role, you must grant the relevant permissions to Elastic Beanstalk. See AWS Elastic Beanstalk Developer Guide for more details.

□ Launch into VPC *If you elect to launch instances in a VPC, the next page will enable you to customize the VPC settings.*

f. Click **Next**.

6. If you selected to launch your environment inside a VPC, the **VPC Options** page appears; otherwise, the **Additional Options** page appears. Here you'll configure your VPC options.

VPC options for load-balanced, autoscaled environment



Publish to AWS

VPC Options

Set Amazon VPC options for the deployed application.

VPC *: vpc-961e8aff (10.0.0.0/16)

ELB Scheme *: Public

Security Group *: default (sg-ce776aa2)

ELB Subnet *: subnet-6f1f8b06 (10.0.0.0/24 - us-west-2b)

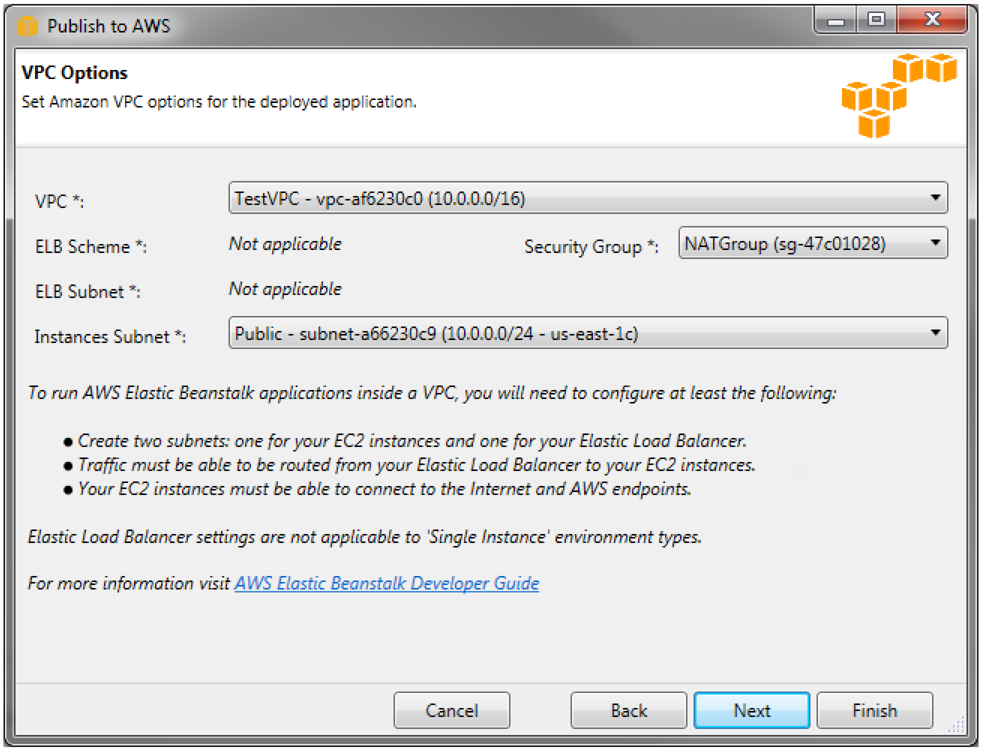
Instances Subnet *: subnet-6c1f8b05 (10.0.1.0/24 - us-west-2a)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- *Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.*
- *Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.*
- *Your EC2 instances must be able to connect to the Internet and AWS endpoints.*

For more information visit AWS Elastic Beanstalk Developer Guide

VPC options for single-instance environment



Publish to AWS

VPC Options

Set Amazon VPC options for the deployed application.

VPC *: TestVPC - vpc-af6230c0 (10.0.0.0/16)

ELB Scheme *: *Not applicable*

Security Group *: NATGroup (sg-47c01028)

ELB Subnet *: *Not applicable*

Instances Subnet *: Public - subnet-a66230c9 (10.0.0.0/24 - us-east-1c)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- *Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.*
- *Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.*
- *Your EC2 instances must be able to connect to the Internet and AWS endpoints.*

Elastic Load Balancer settings are not applicable to 'Single Instance' environment types.

For more information visit AWS Elastic Beanstalk Developer Guide

a. Select the VPC ID of the VPC in which you would like to launch your environment.

> **Note**
> If you do not see the VPC information, then you have not created a VPC in the same region in which you are launching your environment. To learn how to create a VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).

b. For a load-balanced, autoscaled environment, select **private** for **ELB Scheme** if you do not want your elastic load balancer to be available to the Internet.

   For a single-instance environment, this option is not applicable because the environment doesn't have a load balancer. For more information, see Environment Types (p. 68).

c. For a load-balanced, autoscaled environment, select the subnets for the elastic load balancer and the EC2 instances. If you created public and private subnets, make sure the elastic load balancer and the EC2 instances are associated with the correct subnet. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your existing subnets in the Amazon VPC console at https://console.aws.amazon.com/vpc/.

   For a single-instance environment, your VPC only needs a public subnet for the instance. Selecting a subnet for the load balancer is not applicable because the environment doesn't have a load balancer. For more information, see Environment Types (p. 68).

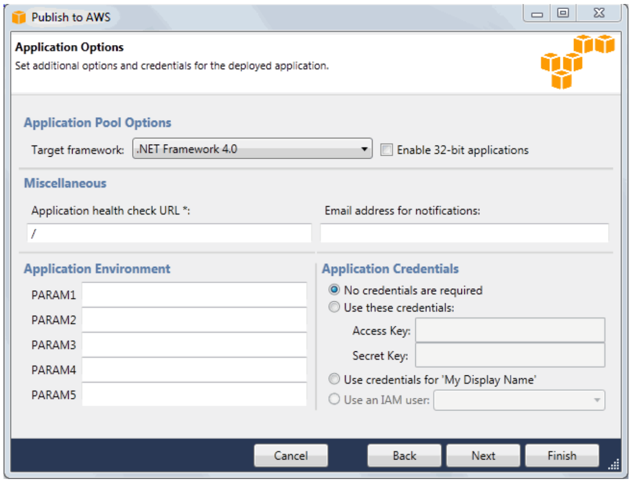
d. For a load-balanced, autoscaled environment, select the VPC security group for your NAT instance. For instructions on how to create this security group and update your default VPC security group, see Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 305).

   For a single-instance environment, you don't need a NAT. Select the default security group. Elastic Beanstalk assigns an Elastic IP address to the instance that lets the instance access the Internet.

e. Click **Next**.

7. On the **Application Options** page, configure your application options.

   a. For Target framework, select **.NET Framework 4.0**.
   b. Elastic Load Balancing uses a health check to determine whether the Amazon EC2 instances running your application are healthy. The health check determines an instance's health status by probing a specified URL at a set interval. You can override the default URL to match an existing resource in your application (e.g., `/myapp/index.aspx`) by entering it in the **Application health check URL** box. For more information about application health checks, see Health Checks (p. 190).
   c. Type an email address if you want to receive Amazon Simple Notification Service (Amazon SNS) notifications of important events affecting your application.
   d. The **Application Environment** section lets you specify environment variables on the Amazon EC2 instances that are running your application. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.
   e. Select the application credentials option you want to use to deploy your application.



Application Options

Set additional options and credentials for the deployed application.

Application Pool Options

Target framework: .NET Framework 4.0

□ Enable 32-bit applications

Miscellaneous

Application health check URL *: /

Email address for notifications:

Application Environment

PARAM1

PARAM2

PARAM3

PARAM4

PARAM5

Application Credentials

● No credentials are required

○ Use these credentials:

Access Key:

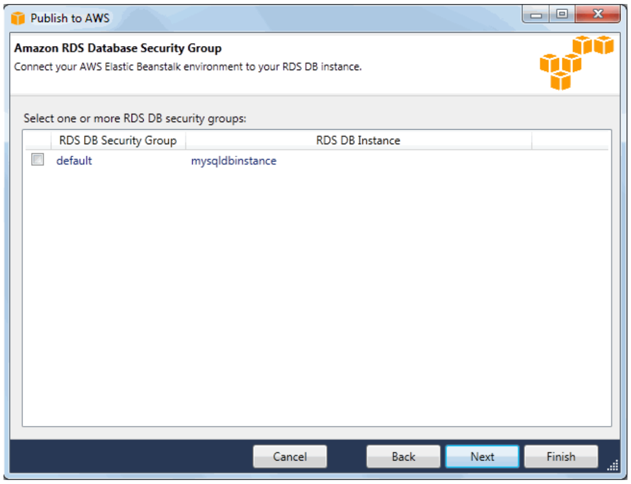
Secret Key:

○ Use credentials for 'My Display Name'

○ Use an IAM user:

f. Click **Next**.

8. If you have previously set up an Amazon RDS database, the **Amazon RDS DB Security Group** page appears. If you want to connect your Elastic Beanstalk environment to your Amazon RDS DB Instance, then select one or more security groups. Otherwise, go on to the next step. When you're ready, click **Next**.



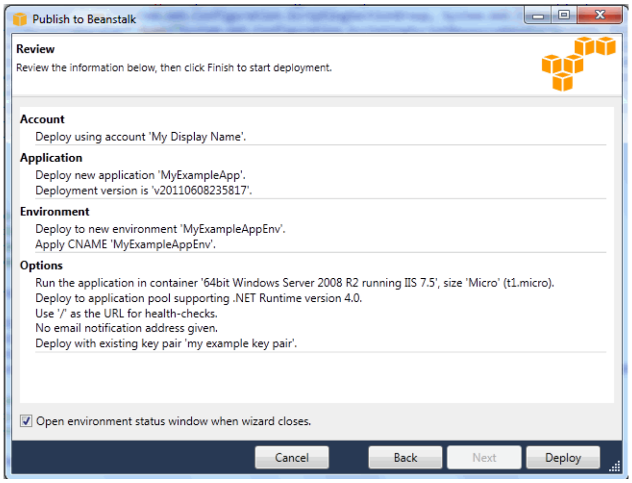
Amazon RDS Database Security Group

Connect your AWS Elastic Beanstalk environment to your RDS DB instance.

Select one or more RDS DB security groups:

	RDS DB Security Group	RDS DB Instance
□	default	mysqldbinstance

9. Review your deployment options. If everything is as you want, click **Deploy**.



Review

Review the information below, then click Finish to start deployment.

Account
Deploy using account 'My Display Name'.

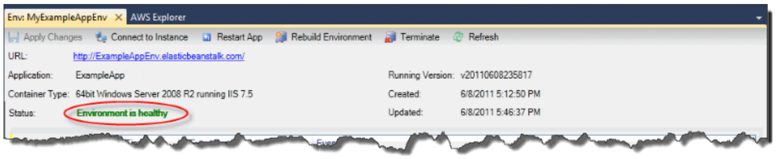
Application
Deploy new application 'MyExampleApp'.
Deployment version is 'v20110608235817'.

Environment
Deploy to new environment 'MyExampleAppEnv'.
Apply CNAME 'MyExampleAppEnv'.

Options
Run the application in container '64bit Windows Server 2008 R2 running IIS 7.5', size 'Micro' (t1.micro).
Deploy to application pool supporting .NET Runtime version 4.0.
Use '/' as the URL for health-checks.
No email notification address given.
Deploy with existing key pair 'my example key pair'.

☑ Open environment status window when wizard closes.

Your ASP.NET project will be exported as a web deploy file, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the env:<environment name> tab, you will see status for your environment.



URL: http://ExampleAppEnv.elasticbeanstalk.com/
Application: ExampleApp
Container Type: 64bit Windows Server 2008 R2 running IIS 7.5
Status: Environment is healthy
Running Version: v20110608235817
Created: 6/8/2011 5:12:50 PM
Updated: 6/8/2011 5:46:37 PM

Debug/View Logs

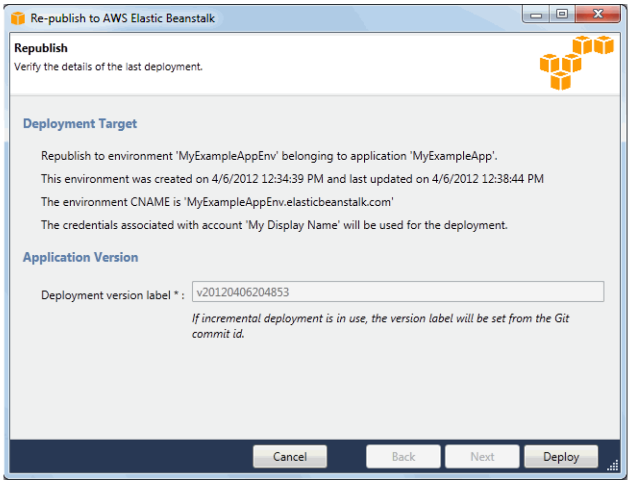
To investigate any issues, you can view logs. For information about viewing logs, see Instance Logs (p. 282). If you need to test remotely, you can connect to your EC2 instances. For instructions on how to connect to your instance, see Listing and Connecting to Server Instances (p. 280).

Edit the Application and Redeploy

Now that you have tested your application, it is easy to edit and redeploy your application and see the results in moments.

To edit and redeploy your ASP.NET web application

1. In **Solution Explorer**, right-click your application, and then click **Republish to Environment <*`your environment name`*>**. The **Re-publish to AWS Elastic Beanstalk** wizard opens.



Republish

Verify the details of the last deployment.

Deployment Target

Republish to environment 'MyExampleAppEnv' belonging to application 'MyExampleApp'.

This environment was created on 4/6/2012 12:34:39 PM and last updated on 4/6/2012 12:38:44 PM

The environment CNAME is 'MyExampleAppEnv.elasticbeanstalk.com'

The credentials associated with account 'My Display Name' will be used for the deployment.

Application Version

Deployment version label * : v20120406204853

If incremental deployment is in use, the version label will be set from the Git commit id.

2. Review your deployment details and click **Deploy**.

   Note
   If you want to change any of your settings, you can click **Cancel** and use the **Publish to AWS** wizard instead. For instructions, see Deploy to AWS Elastic Beanstalk (p. 629).

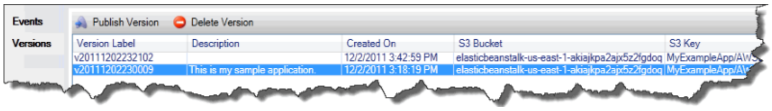
   Your updated ASP.NET web project will be exported as a web deploy file with the new version label, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature monitors your existing environment until it becomes available with the newly deployed code. On the **env:<*environment name*>** tab, you will see the status of your environment.

Deploy to Production

When you are satisfied with all of the changes you want to make to your application, you can deploy your application version to your production environment.

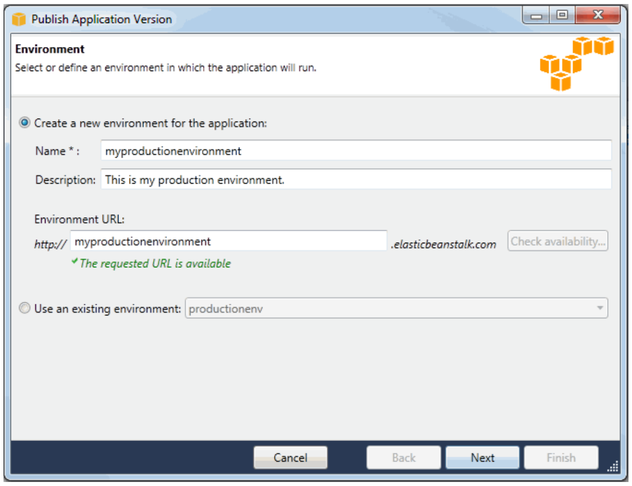
To deploy an application version to production

1. Right-click your Elastic Beanstalk application by expanding the Elastic Beanstalk node in **AWS Explorer**. Select **View Status**.
2. In the **App: <*application name*>** tab, click **Versions**.



Version Label	Description	Created On	S3 Bucket	S3 Key
v20111202232102		12/2/2011 3:42:59 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AWS
v20111202230009	This is my sample application.	12/2/2011 3:18:19 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AW

3. Click the application version you want to deploy and click **Publish Version**. The **Publish Application Version** wizard appears.



Publish Application Version

Environment

Select or define an environment in which the application will run.

● Create a new environment for the application:

Name *: myproductionenvironment

Description: This is my production environment.

Environment URL:

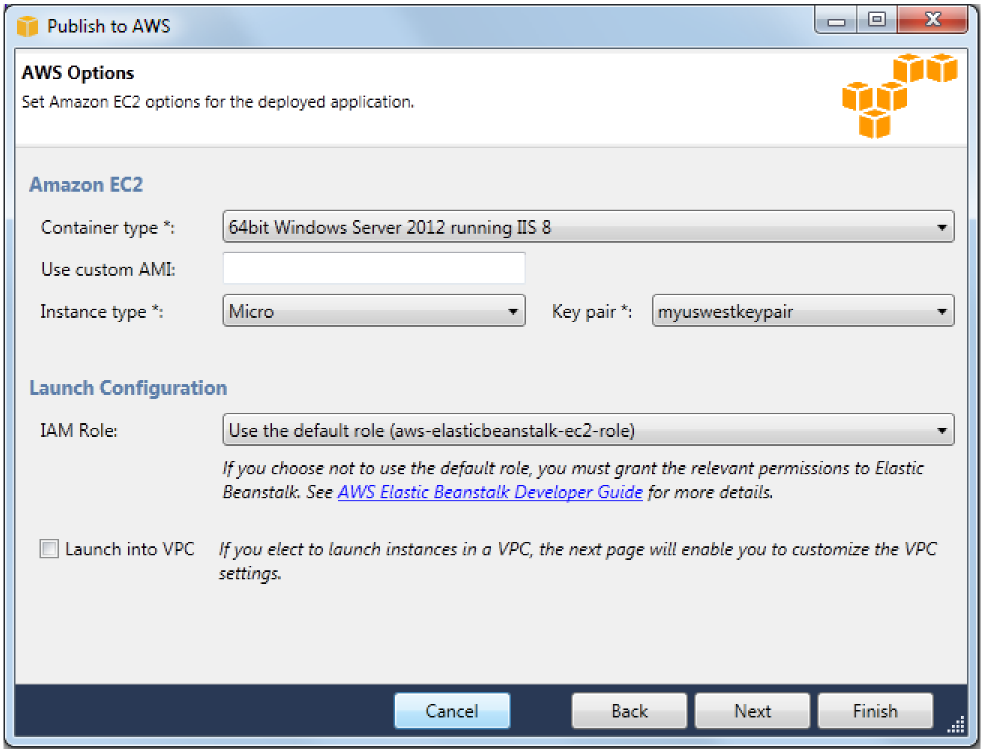
http:// myproductionenvironment .elasticbeanstalk.com

✓ *The requested URL is available*

○ Use an existing environment: productionenv

4. Describe your environment details.

   a. Select the **Create a new environment for this application** button.
   b. In the **Name** box, type a name for your environment name.
   c. In the **Description** box, type a description for your environment. This step is optional.
   d. The environment URL is auto-populated in the **Environment URL** box.
   e. Click **Check availability** to make sure the environment URL is available.
   f. Click **Next**. The **AWS Options** page appears.



Publish to AWS

AWS Options

Set Amazon EC2 options for the deployed application.

Amazon EC2

Container type *: 64bit Windows Server 2012 running IIS 8

Use custom AMI:

Instance type *: Micro

Key pair *: myuswestkeypair

Launch Configuration

IAM Role: Use the default role (aws-elasticbeanstalk-ec2-role)

If you choose not to use the default role, you must grant the relevant permissions to Elastic Beanstalk. See AWS Elastic Beanstalk Developer Guide for more details.

☐ Launch into VPC *If you elect to launch instances in a VPC, the next page will enable you to customize the VPC settings.*

5. Configure additional options and security information for your deployment.

a. In the **Container Type** list, click **64bit Windows Server 2012 running IIS 8** or **64bit Windows Server 2008 running IIS 7.5**.

b. In the **Instance Type** list, click **Micro**.

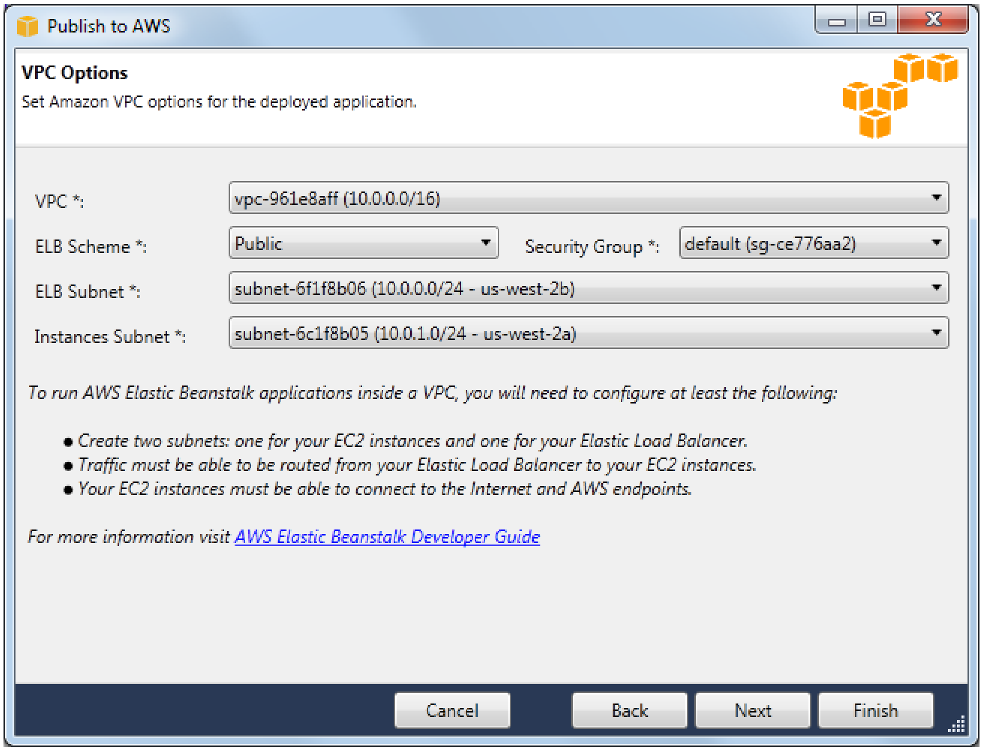
c. In the **Key pair** list, click **Create new key pair**. Type a name for the new key pair—in this example, we use **my example key pair**—and then click **OK**. A key pair enables you to remote desktop into your Amazon EC2 instances. For more information on Amazon EC2 key pairs, see Using Credentials in the *Amazon Elastic Compute Cloud User Guide*.

d. Select an instance profile.

   If you do not have an instance profile, select **Create a default instance profile**. For information about using instance profiles with Elastic Beanstalk, see Using IAM Roles with Elastic Beanstalk (p. 332).

e. If you are using a nonlegacy container, you have the option to create your environment inside an existing VPC; to do this, click **Launch into VPC**. You can configure the VPC information on the next page. For more information about Amazon VPC, go to Amazon Virtual Private Cloud (Amazon VPC). For a list of supported nonlegacy container types, see Why are some container types marked legacy? (p. 92).

f. Click **Next**. If you selected to launch your environment inside a VPC, the **VPC Options** page appears, otherwise the **Additional Options** page appears.



VPC Options

Set Amazon VPC options for the deployed application.

VPC *: vpc-961e8aff (10.0.0.0/16)

ELB Scheme *: Public

Security Group *: default (sg-ce776aa2)

ELB Subnet *: subnet-6f1f8b06 (10.0.0.0/24 - us-west-2b)

Instances Subnet *: subnet-6c1f8b05 (10.0.1.0/24 - us-west-2a)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- *Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.*
- *Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.*
- *Your EC2 instances must be able to connect to the Internet and AWS endpoints.*

For more information visit AWS Elastic Beanstalk Developer Guide

6. Configure your VPC options.

   a. Select the VPC ID of the VPC in which you would like to launch your environment.

      Note

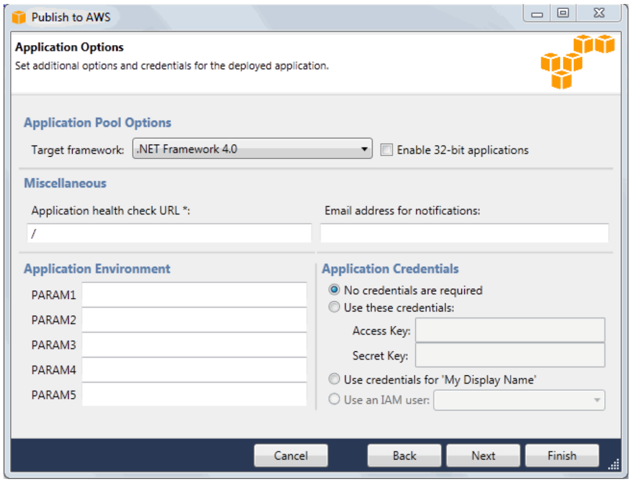
      If you do not see the VPC information, then you have not created a VPC in the same region in which you are launching your environment. To learn how to create a VPC, see Using Elastic Beanstalk with Amazon VPC (p. 297).

b. In the ELB Scheme list, select **private** if you do not want your elastic load balancer to be available to the Internet.

c. Select the subnets for the elastic load balancer and the EC2 instances. If you created public and private subnets, make sure the elastic load balancer and the EC2 instances are associated with the correct subnet. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your existing subnets in the Amazon VPC console at https://console.aws.amazon.com/vpc/.

d. Select the VPC security group for your NAT instance. For instructions on how to create this security group and update your default VPC security group, see Step 2: Configure the Default VPC Security Group for the NAT Instance (p. 305).

e. Click **Next**. The **Application Options** page appears.



Application Options

Set additional options and credentials for the deployed application.

Application Pool Options

Target framework: .NET Framework 4.0 ▾ □ Enable 32-bit applications

Miscellaneous

Application health check URL *: /

Email address for notifications:

Application Environment

PARAM1
PARAM2
PARAM3
PARAM4
PARAM5

Application Credentials

● No credentials are required
○ Use these credentials:
Access Key:
Secret Key:
○ Use credentials for 'My Display Name'
○ Use an IAM user:

7. Configure your application options.

a. Select **.NET Runtime 4.0** from the **Target runtime** pull-down menu.

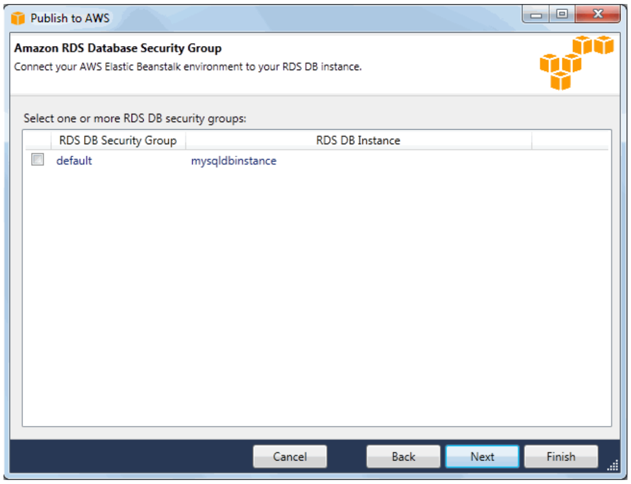
b. Elastic Load Balancing uses a health check to determine whether the Amazon EC2 instances running your application are healthy. The health check determines an instance's health status by probing a specified URL at a set interval. You can override the default URL to match an existing resource in your application (e.g., /myapp/index.aspx) by entering it in the **Application health check URL** box. For more information about application health checks, see Health Checks (p. 190).

c. Type an email address if you want to receive Amazon Simple Notification Service (Amazon SNS) notifications of important events affecting your application.

d. The **Application Environment** section lets you specify environment variables on the Amazon EC2 instances that are running your application. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.

e. Click the application credentials option you want to use to deploy your application.

f. Click **Next**. If you have previously set up an Amazon RDS database, the **Amazon RDS DB Security Group** page appears. Otherwise, skip the next step to review your deployment information.



Publish to AWS

Amazon RDS Database Security Group

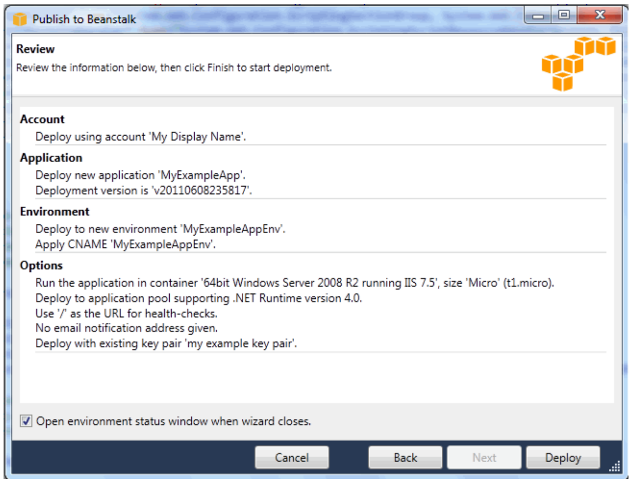
Connect your AWS Elastic Beanstalk environment to your RDS DB instance.

Select one or more RDS DB security groups:

	RDS DB Security Group	RDS DB Instance
□	default	mysqldbinstance

8. Configure the Amazon RDS Database security group.

   - If you want to connect your Elastic Beanstalk environment to your Amazon RDS DB Instance, select one or more security groups. Click **Next**. The **Review** page appears.



Publish to Beanstalk

Review

Review the information below, then click Finish to start deployment.

Account

Deploy using account 'My Display Name'.

Application

Deploy new application 'MyExampleApp'.
Deployment version is 'v20110608235817'.

Environment

Deploy to new environment 'MyExampleAppEnv'.
Apply CNAME 'MyExampleAppEnv'.

Options

Run the application in container '64bit Windows Server 2008 R2 running IIS 7.5', size 'Micro' (t1.micro).
Deploy to application pool supporting .NET Runtime version 4.0.
Use '/' as the URL for health-checks.
No email notification address given.
Deploy with existing key pair 'my example key pair'.

☑ Open environment status window when wizard closes.

9. Review your deployment options, and click **Deploy**.

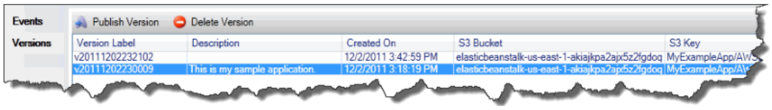
   Your ASP.NET project will be exported as a web deploy file and uploaded to Amazon S3. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the env:<*environment name*> tab, you will see status for your environment.

Deploy an Existing Application Version to an Existing Environment

You can deploy an existing application to an existing environment if, for instance, you need to roll back to a previous application version.

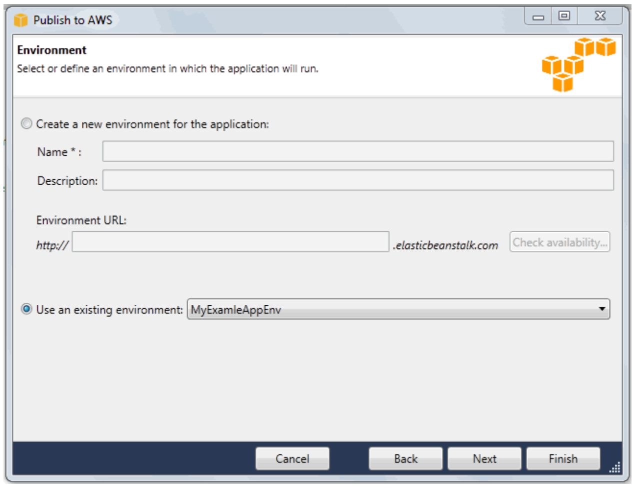
To deploy an application version to an existing environment

1. Right-click your Elastic Beanstalk application by expanding the Elastic Beanstalk node in **AWS Explorer**. Select **View Status**.
2. In the **App: <*application name*>** tab, click **Versions**.



Version Label	Description	Created On	S3 Bucket	S3 Key
v20111202232102		12/2/2011 3:42:59 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AWS
v20111202230009	This is my sample application.	12/2/2011 3:18:19 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AW

3. Click the application version you want to deploy and click **Publish Version**.
4. In the **Publish Application Version** wizard, click **Next**.


Publish to AWS

Environment

Select or define an environment in which the application will run.

○ Create a new environment for the application:

Name * :

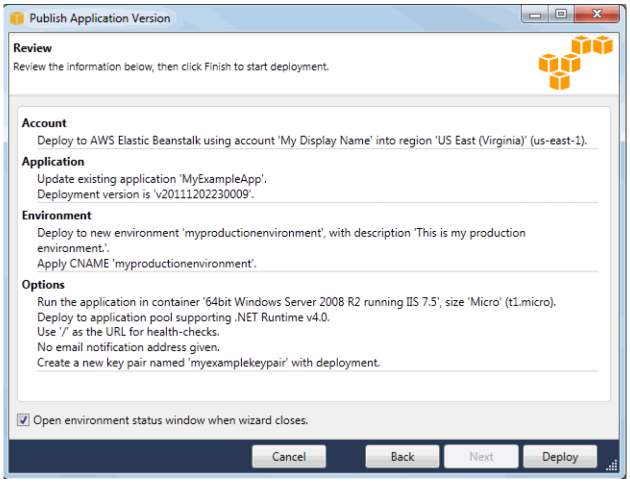
Description:

Environment URL:

http:// .elasticbeanstalk.com

● Use an existing environment: MyExamleAppEnv

5. Review your deployment options, and click **Deploy**.


Publish Application Version

Review

Review the information below, then click Finish to start deployment.

Account

Deploy to AWS Elastic Beanstalk using account 'My Display Name' into region 'US East (Virginia)' (us-east-1).

Application

Update existing application 'MyExampleApp'.
Deployment version is 'v20111202230009'.

Environment

Deploy to new environment 'myproductionenvironment', with description 'This is my production environment.'.
Apply CNAME 'myproductionenvironment'.

Options

Run the application in container '64bit Windows Server 2008 R2 running IIS 7.5', size 'Micro' (t1.micro).
Deploy to application pool supporting .NET Runtime v4.0.
Use '/' as the URL for health-checks.
No email notification address given.
Create a new key pair named 'myexamplekeypair' with deployment.

☑ Open environment status window when wizard closes.

Your ASP.NET project will be exported as a web deploy file and uploaded to Amazon S3. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the **env:<*environment name*>** tab, you will see status for your environment.

Configuring .NET Containers with Elastic Beanstalk

Customizing and Configuring a .NET Environment

When deploying your .NET application, you might want to customize and configure the behavior of your Amazon EC2 instances. You can easily customize your instances at the same time that you deploy your application version by including a configuration file with your source bundle. You might also want to customize your environment resources that are part of your Elastic Beanstalk environment (for example, Amazon SQS queues or Amazon ElastiCache clusters). For example, you might want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster. This section walks you through the process of creating a configuration file and bundling it with your source.

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

> **Note**
> This section does not apply to legacy .NET containers. If you are running an application using a legacy .NET container, then we recommend that you migrate to a nonlegacy .NET container. For instructions on how to check if you are running a legacy container and to migrate to a nonlegacy container, see Migrating Your Application from a Legacy Container Type (p. 91). If you require instructions for managing your environment for a legacy container using the Elastic Beanstalk console, command line interface, or API, see Managing Environments (p. 50).

To customize and configure your .NET environment

1. Create a configuration file with the extension `.config` (for example, `myapp.config`) and place it in an `.ebextensions` top-level directory of your source bundle. You can have multiple configuration files in your `.ebextensions` directory. For information about the file format and contents of the configuration file, see Using Configuration Files (p. 99).

   > **Note**
   > For Visual Studio, `.ebextensions` needs to be part of the project to be included in the archive. Alternatively, instead of including the configuration files in the project, you can use Visual Studio to deploy all files in the project folder. In **Solution Explorer**, right-click the project name, and then click **Properties**. Click the **Package/Publish Web** tab. In the **Items to deploy** section, select **All Files in the Project Folder** in the drop-down list.

   The following is an example snippet of a configuration file.

```
# If you do not specify a namespace, the default used is aws:elasticbean
stalk:application:environment
container_commands:
  foo:
    command: set > c:\\myapp\\set.txt
    leader_only: true
 waitAfterCompletion: 0

option_settings:
  - option_name: PARAM1
    value: somevalue
```

Note
You can specify any key–value pairs in the `aws:elasticbeanstalk:application:environment` namespace, and they will be passed in as environment variables on your Amazon EC2 instances.

2. Deploy your application version.

For an example walkthrough of deploying a .NET application and using custom CloudWatch metrics, see Example: Using Custom Amazon CloudWatch Metrics (p. 167).

Accessing Environment Configuration Settings

The parameters specified in the `option_settings` section of the configuration file are passed in and used as application settings.

You might have a code snippet that looks similar to the following to access the keys and parameters:

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string param1 = appConfig["PARAM1"];
```

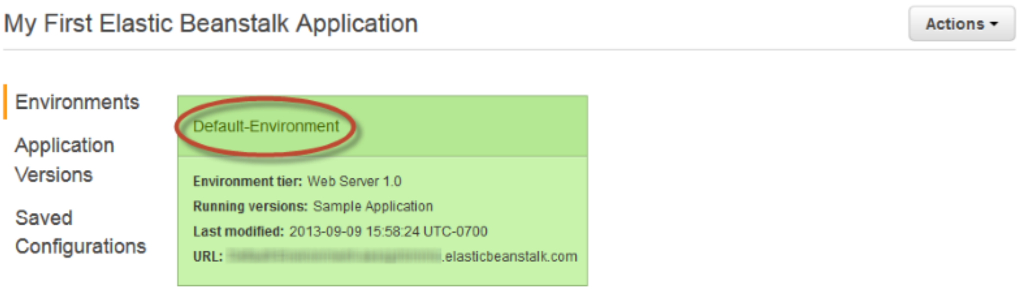
Note
Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

For a list of configuration settings, see .NET Container Options (p. 122).

AWS Management Console

To access the .NET container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

.NET Container Options

For your application's version Framework, you can choose either 2.0 or 4.0. Select **Enable 32-bit Applications** if you plan to run 32-bit applications.

Amazon S3 Log Rotation

Elastic Beanstalk can copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application on an hourly basis. To enable this feature, select **Enable log file rotation to Amazon S3**.

Environment Properties

The **Environment Properties** section lets you specify application settings. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.

You can configure the following application settings:

- Specify AWS credentials using the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_KEY** text boxes.

  Note
  Elastic Beanstalk uses instance profiles so that your application can use temporary security credentials to access AWS resources. To learn more, see Service Roles, Instance Profiles, and User Policies (p. 19).

- Specify up to five additional key-value pairs by entering them in the **PARAM** boxes.

  You might have a code snippet that looks similar to the following to access the keys and parameters:

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string param1 = appConfig["PARAM1"];
```

Note
These settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Command Line Interface CLI)

To access the Container/.NET Options panel for your Elastic Beanstalk application

- Update an application's environment settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

ptions.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:container:dotnet:apppool",
        "OptionName": "Target Runtime",
        "Value": "4.0"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:dotnet:apppool",
        "OptionName": "Enable 32-bit Applications",
        "Value": "false"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_ACCESS_KEY_ID",
        "Value": "AKIAIOSFODNN7EXAMPLE"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_SECRET_KEY",
        "Value": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "myvar",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To access the Container/.NET Options panel for your Elastic Beanstalk application

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:container:dotnet:apppool`
  - *`OptionSettings.member.1.OptionName`* = `Target Runtime`
  - *`OptionSettings.member.1.Value`* = `4.0`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:container:dotnet:apppool`
  - *`OptionSettings.member.2.OptionName`* = `Enable 32-bit Applications`
  - *`OptionSettings.member.2.Value`* = `false`
  - *`OptionSettings.member.3.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.3.OptionName`* = `AWS_ACCESS_KEY_ID`
  - *`OptionSettings.member.3.Value`* = `AKIAIOSFODNN7EXAMPLE`
  - *`OptionSettings.member.4.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.4.OptionName`* = `AWS_SECRET_KEY`
  - *`OptionSettings.member.4.Value`* = `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
  - *`OptionSettings.member.5.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.5.OptionName`* = `myvar`

- *OptionSettings.member.5.Value* = somevalue
- *OptionSettings.member.6.Namespace* = aws:elasticbeanstalk:hostmanager
- *OptionSettings.member.6.OptionName* = LogPublicationControl
- *OptionSettings.member.6.Value* = false

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=MySampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Adot
net%3Aapppool
&OptionSettings.member.1.OptionName=Target%20Runtime
&OptionSettings.member.1.Value=4.0
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Adot
net%3Aapppool
&OptionSettings.member.2.OptionName=Enable%2032-bit%20Applications
&OptionSettings.member.2.Value=false
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.3.OptionName=AWS_ACCESS_KEY_ID
&OptionSettings.member.3.Value=AKIAIOSFODNN7EXAMPLE
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.4.OptionName=AWS_SECRET_KEY
&OptionSettings.member.4.Value=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.5.OptionName=myvar
&OptionSettings.member.5.Value=somevalue
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.6.OptionName=LogPublicationControl
&OptionSettings.member.6.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Using Amazon RDS

With Amazon Relational Database Service (Amazon RDS), you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. For more information, go to Amazon Relational Database.

You can use Amazon RDS with your Elastic Beanstalk .NET application using the AWS Management Console. You can also use the AWS Toolkit for Visual Studio to create your Amazon RDS DB instance. For an example walkthrough using SQL Server, see Tutorial: How to Deploy a .NET Sample Application Using AWS Elastic Beanstalk (p. 613).

> **Note**
> The instructions in this topic are for nonlegacy container types. If you have deployed an Elastic Beanstalk application using a legacy container type, we recommend that you migrate to a nonlegacy container type to gain access to new features. For instructions on how to check the container type and migrate your application, see Migrating Your Application from a Legacy Container Type (p. 91).

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. Download and install a database driver.
3. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
4. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with .NET

You can create a new Amazon RDS DB instance using the console or use the RDS connection information with your .NET application.

To create an Amazon RDS DB instance with .NET

1. Create an Amazon RDS DB instance. You can create an RDS DB instance in one of the following ways:

   - Create an Amazon RDS DB instance when you create a new application version. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34).
   - Create an Amazon RDS DB instance when you launch a new environment with an existing application version. For instructions using the AWS Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
   - If you already deployed an application to AWS Elastic Beanstalk, you can create an RDS DB instance and attach it to an existing environment. For instructions using the AWS Elastic Beanstalk console, see Step 4: Deploy New Version (p. 6).

2. Download and install a database driver for your development environment.
3. Establish a database connection in your code using your Amazon RDS DB instance's connectivity information. You can access your connectivity information using application settings. The following shows how you would connect to the database on an RDS instance.

   ```
   NameValueCollection appConfig = ConfigurationManager.AppSettings;

   string dbname = appConfig["RDS_DB_NAME"];
   string username = appConfig["RDS_USERNAME"];
   string password = appConfig["RDS_PASSWORD"];
   string hostname = appConfig["RDS_HOSTNAME"];
   string port = appConfig["RDS_PORT"];
   ```

   Build your connection string:

   ```
   string cs = "server=" + hostname + ";user=" + username + ";database=" + dbname
    + ";port=" + port + ";password=" + password + ";";
   ```

   The following connection string is an example using MySQL.

```
"server=mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com;user=sa;data
base=mydb;port=3306;password=******;";
```

4. Deploy your updated application to your existing Elastic Beanstalk environment. For information on how to deploy a new application version to an existing environment using the Elastic Beanstalk Console, see Step 4: Deploy New Version (p. 6). For information on how to deploy your application using Visual Studio, see Develop, Test, and Deploy (p. 627).

Using an Existing Amazon RDS DB Instance with .NET

Amazon Relational Database Service (Amazon RDS) lets you quickly and easily provision and maintain a MySQL Server instance in the cloud. This topic discusses how you can use Amazon RDS and the MySQL .NET Connector with your Elastic Beanstalk application.

To use an existing Amazon RDS DB instance and .NET from your Elastic Beanstalk application

1. Create an Elastic Beanstalk environment in one of the following ways:
   - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For instructions using Visual Studio, see Develop, Test, and Deploy (p. 627). You do not need to create an RDS DB instance with this environment because you already have an existing RDS DB instance.
   - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.
2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Download and install a database driver for your development environment.
4. Create a connection string using your Amazon RDS DB instance's public DNS name, port number, and (optionally) database name and login credentials. The following example shows a connection string that would connect to a database, mydb, on an RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name `sa` and the password `mypassword`.

```
string cs = "server=mydbinstance.abcdefghijkl.us-west-2.rds.amazon
aws.com;user=sa;database=mydb;port=3306;password=******;";
```

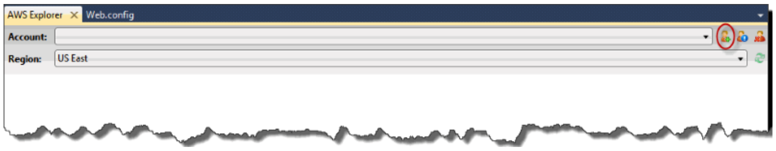
5. Deploy your updated application to your existing Elastic Beanstalk environment. For information on how to deploy a new application version to an existing environment using the Elastic Beanstalk Console, see Step 4: Deploy New Version (p. 6). For information on how to deploy your application using Visual Studio, see Develop, Test, and Deploy (p. 627).

Managing Multiple Accounts

If you want to set up different AWS accounts to perform different tasks, such as testing, staging, and production, you can add, edit, and delete accounts using the AWS Toolkit for Visual Studio.

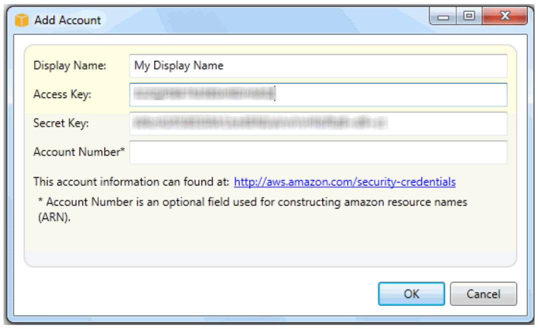
To manage multiple accounts

1. In Visual Studio, on the **View** menu, click **AWS Explorer**.
2. Beside the **Account** list, click the **Add Account** button.



Account:
Region: US East

The **Add Account** dialog box appears.



Display Name: My Display Name
Access Key: [redacted]
Secret Key: [redacted]
Account Number*

This account information can found at: http://aws.amazon.com/security-credentials

* Account Number is an optional field used for constructing amazon resource names (ARN).

3. Fill in the requested information.
4. Your account information now appears on the **AWS Explorer** tab. When you publish to Elastic Beanstalk, you can select which account you would like to use.

Monitoring Application Health

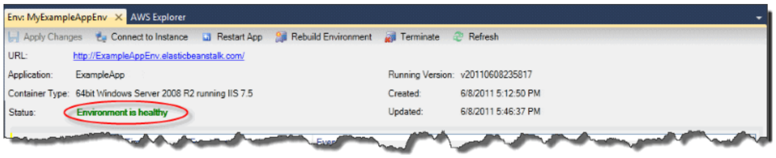
When you are running a production website, it is important to know that your application is available and responding to requests. To assist with monitoring your application's responsiveness, Elastic Beanstalk provides features where you can monitor statistics about your application and create alerts that trigger when thresholds are exceeded.

For information about the health monitoring provided by Elastic Beanstalk, see Basic Health Reporting (p. 247).

Viewing Application Health and Environment Status

You can access operational information about your application by using either the AWS Toolkit for Visual Studio or the AWS Management Console.

The toolkit displays your environment's status and application health in the **Status** field.



URL: http://ExampleAppEnv.elasticbeanstalk.com/

Application: ExampleApp

Container Type: 64bit Windows Server 2008 R2 running IIS 7.5

Status: Environment is healthy

Running Version: v20110608235817

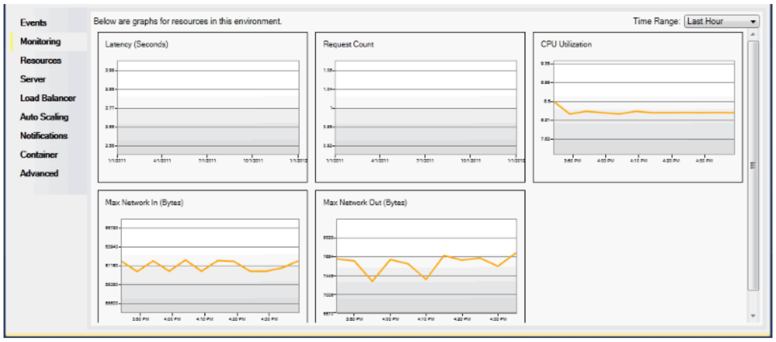
Created: 6/8/2011 5:12:50 PM

Updated: 6/8/2011 5:46:37 PM

To monitor application health

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node, and then expand your application node.
2. Right-click your Elastic Beanstalk environment, and then click **View Status**.
3. On your application environment tab, click **Monitoring**.

   The **Monitoring** panel includes a set of graphs showing resource usage for your particular application environment.



Below are graphs for resources in this environment.

Time Range: Last Hour

Note

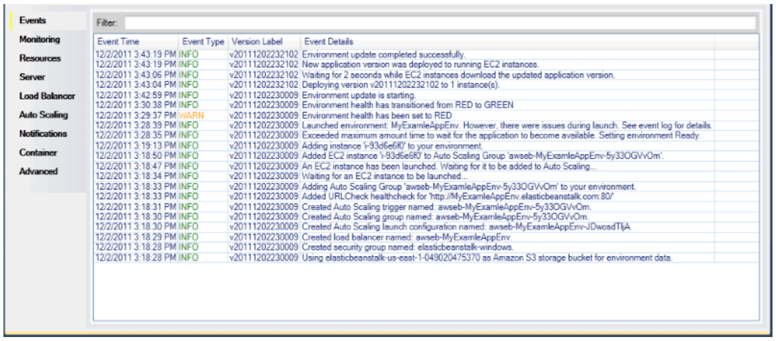
By default, the time range is set to the last hour. To modify this setting, in the **Time Range** list, click a different time range.

Viewing Events

You can use the AWS Toolkit for Visual Studio or the AWS Management Console to view events associated with your application. For information on the most common events, see Understanding Environment Launch Events (p. 531). For instructions on how to use the AWS Management Console to view events, see View Events (p. 278). This section steps you through viewing events using the AWS Toolkit for Visual Studio.

To view application events

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node and your application node.
2. Right-click your Elastic Beanstalk environment in **AWS Explorer** and then click **View Status**.
3. In your application environment tab, click **Events**.



Event Time	Event Type	Version Label	Event Details
12/2/2011 3:43:19 PM	INFO	v20111202232102	Environment update completed successfully.
12/2/2011 3:43:19 PM	INFO	v20111202232102	New application version was deployed to running EC2 instances.
12/2/2011 3:43:06 PM	INFO	v20111202232102	Waiting for 2 seconds while EC2 instances download the updated application version.
12/2/2011 3:43:04 PM	INFO	v20111202232102	Deploying version v20111202232102 to 1 instance(s).
12/2/2011 3:42:59 PM	INFO	v20111202230009	Environment update is starting.
12/2/2011 3:30:38 PM	INFO	v20111202230009	Environment health has transitioned from RED to GREEN
12/2/2011 3:29:37 PM	WARN	v20111202230009	Environment health has been set to RED
12/2/2011 3:28:39 PM	INFO	v20111202230009	Launched environment: MyExamleAppEnv. However, there were issues during launch. See event log for details.
12/2/2011 3:28:35 PM	INFO	v20111202230009	Exceeded maximum amount time to wait for the application to become available. Setting environment Ready.
12/2/2011 3:19:13 PM	INFO	v20111202230009	Adding instance 'i-93d6e6f0' to your environment.
12/2/2011 3:18:50 PM	INFO	v20111202230009	Added EC2 instance 'i-93d6e6f0' to Auto Scaling Group 'awseb-MyExamleAppEnv-5y33OGVvOm'.
12/2/2011 3:18:47 PM	INFO	v20111202230009	An EC2 instance has been launched. Waiting for it to be added to Auto Scaling...
12/2/2011 3:18:34 PM	INFO	v20111202230009	Waiting for an EC2 instance to be launched...
12/2/2011 3:18:33 PM	INFO	v20111202230009	Adding Auto Scaling Group 'awseb-MyExamleAppEnv-5y33OGVvOm' to your environment.
12/2/2011 3:18:33 PM	INFO	v20111202230009	Added URLCheck healthcheck for 'http://MyExamleAppEnv.elasticbeanstalk.com:80/'
12/2/2011 3:18:31 PM	INFO	v20111202230009	Created Auto Scaling trigger named: awseb-MyExamleAppEnv-5y33OGVvOm.
12/2/2011 3:18:30 PM	INFO	v20111202230009	Created Auto Scaling group named: awseb-MyExamleAppEnv-5y33OGVvOm.
12/2/2011 3:18:30 PM	INFO	v20111202230009	Created Auto Scaling launch configuration named: awseb-MyExamleAppEnv-JDwosdTljA.
12/2/2011 3:18:29 PM	INFO	v20111202230009	Created load balancer named: awseb-MyExamleAppEnv.
12/2/2011 3:18:28 PM	INFO	v20111202230009	Created security group named: elasticbeanstalk-windows.
12/2/2011 3:18:28 PM	INFO	v20111202230009	Using elasticbeanstalk-us-east-1-049020475370 as Amazon S3 storage bucket for environment data.

Managing Your Elastic Beanstalk Application Environments

With the AWS Toolkit for Visual Studio and the AWS Management Console, you can change the provisioning and configuration of the AWS resources used by your application environments. For information on how to manage your application environments using the AWS Management Console, see Managing Environments (p. 50). This section discusses the specific service settings you can edit in the AWS Toolkit for Visual Studio as part of your application environment configuration.

Changing Environment Configurations Settings

When you deploy your application, Elastic Beanstalk configures a number of AWS cloud computing services. You can control how these individual services are configured using the AWS Toolkit for Visual Studio.

To edit an application's environment settings

- Expand the Elastic Beanstalk node and your application node. Then right-click your Elastic Beanstalk environment in **AWS Explorer**. Select **View Status**.

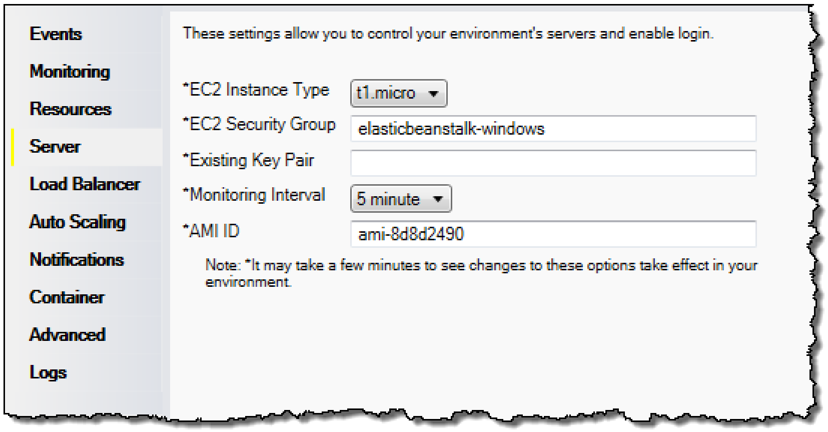
  You can now configure settings for the following:

  - Server
  - Load balancing
  - Autoscaling
  - Notifications
  - Environment properties

Configuring EC2 Server Instances Using the AWS Toolkit for Visual Studio

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that you use to launch and manage server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to Amazon EC2.

You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Server** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



These settings allow you to control your environment's servers and enable login.

*EC2 Instance Type: t1.micro

*EC2 Security Group: elasticbeanstalk-windows

*Existing Key Pair:

*Monitoring Interval: 5 minute

*AMI ID: ami-8d8d2490

Note: *It may take a few minutes to see changes to these options take effect in your environment.

Amazon EC2 Instance Types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations may require more CPU or memory. Elastic Beanstalk regularly checks your running instances to ensure they are healthy. If your application consumes 95 percent or greater of the CPU, Elastic Beanstalk will trigger an event. For more information about this event, see CPU Utilization Exceeds 95.00% (p. 532).

> **Note**
> You cannot change between 32-bit and 64-bit instance types. For example, if your application is built on a 32-bit platform, only 32-bit instance types appear in the list.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see Instance Types in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Security Groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the AWS Management Console or by using the AWS Toolkit for Visual Studio. You can specify which Amazon EC2 Security Groups control access to your Elastic Beanstalk application by entering the names of one or more Amazon EC2 security group names (delimited by commas) into the **EC2 Security Groups** text box.

> **Note**
> Make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see Health Checks (p. 655).

To create a security group using the AWS Toolkit for Visual Studio

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node, and then double-click **Security Groups**.
2. Click **Create Security Group**, and enter a name and description for your security group.

3. Click **OK**.

For more information on Amazon EC2 Security Groups, see Using Security Groups in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 Key Pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

> **Important**
> You must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances. You can create your key pair using the **Publish to AWS** wizard inside the AWS Toolkit for Visual Studio when you deploy your application to Elastic Beanstalk. If you want to create additional key pairs using the Toolkit, follow the steps below. Alternatively, you can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the Amazon Elastic Compute Cloud Getting Started Guide.

The **Existing Key Pair** text box lets you specify the name of an Amazon EC2 key pair you can use to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.

To specify the name of an Amazon EC2 key pair

1. Expand the **Amazon EC2** node and double-click **Key Pairs**.
2. Click **Create Key Pair** and enter the key pair name.
3. Click **OK**.

For more information about Amazon EC2 key pairs, go to Using Amazon EC2 Credentials in the *Amazon Elastic Compute Cloud User Guide*. For more information about connecting to Amazon EC2 instances, see Listing and Connecting to Server Instances (p. 660).

Monitoring Interval

By default, only basic Amazon CloudWatch metrics are enabled; they return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

> **Note**
> Amazon CloudWatch service charges can apply for one-minute interval metrics. See Amazon CloudWatch for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

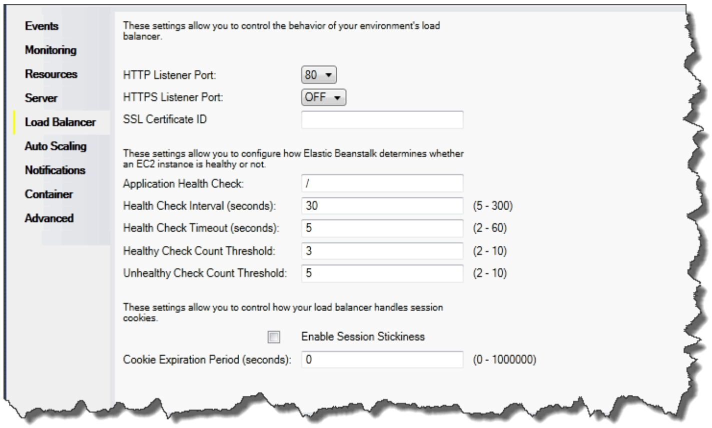
> **Important**
> Using your own AMI is an advanced task and should be done with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing Using the AWS Toolkit for Visual Studio

Elastic Load Balancing is an Amazon web service that helps you improve the availability and scalability of your application. This service makes it easy for you to distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing enables availability through redundancy and supports traffic growth for your application.

Elastic Load Balancing lets you automatically distribute and balance the incoming application traffic among all the instances you are running. The service also makes it easy to add new instances when you need to increase the capacity of your application.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Load Balancer** tab inside your application environment tab in AWS Toolkit for Visual Studio.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port: 80

HTTPS Listener Port: OFF

SSL Certificate ID

These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check: /

Health Check Interval (seconds): 30 (5 - 300)

Health Check Timeout (seconds): 5 (2 - 60)

Healthy Check Count Threshold: 3 (2 - 10)

Unhealthy Check Count Threshold: 5 (2 - 10)

These settings allow you to control how your load balancer handles session cookies.

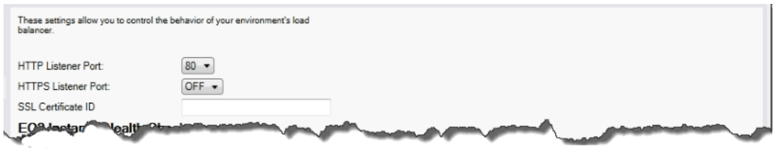
□ Enable Session Stickiness

Cookie Expiration Period (seconds): 0 (0 - 1000000)

The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. At least one of the ports (either HTTP or HTTPS) must be turned on.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port: 80

HTTPS Listener Port: OFF

SSL Certificate ID

Important

Make sure that the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application.

Controlling the HTTP Port

To turn off the HTTP port, select **OFF** for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**) from the list.

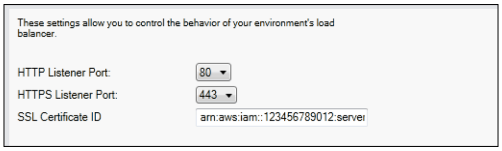
> **Note**
>
> If you want to access your environment using a different port other than the default port 80 (e.g., port 8080), you can add a listener to the existing load balancer and configure the new listener to listen on that port. For example, using the Elastic Load Balancing API Tools, type the following command replacing `<yourloadbalancername>` with the name of your load balancer for Elastic Beanstalk.
>
> ```
> elb-create-lb-listeners --lb <yourloadbalancername> --listener "pro
> tocol=http, lb-port=8080, instance-port=80"
> ```
>
> If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS Port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances use plaintext encryption. By default, the HTTPS port is turned off.

To turn on the HTTPS port

1. Create and upload a certificate and key to the AWS Identity and Access Management (IAM) service. The IAM service will store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate you've uploaded. For more information about creating and uploading certificates, see the Managing Server Certificates section of *Using AWS Identity and Access Management*.
2. Specify the HTTPS port by selecting a port for **HTTPS Listener Port**.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port: 80
HTTPS Listener Port: 443
SSL Certificate ID arn:aws:iam::123456789012:server

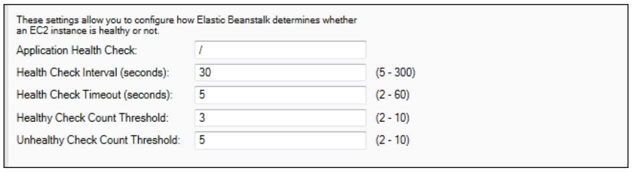
3. For **SSL Certificate ID**, enter the Amazon Resources Name (ARN) of your SSL certificate (e.g., **`arn:aws:iam::123456789012:server-certificate/abc/certs/build`**). Use the SSL certificate that you created and uploaded in step 1. For information on viewing the certificate's ARN, see Verify the Certificate Object topic in the *Creating and Uploading Server Certificates* section of the *Using IAM Guide*.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health Checks

The health check definition includes a URL to be queried for instance health. By default, Elastic Beanstalk uses TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override the default URL to match an existing resource in your application (e.g., `/myapp/default.aspx`) by entering it in the **Application Health Check URL** box. If you override the default URL, then Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see Why are some container types marked legacy? (p. 92).

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.



These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check:	/	
Health Check Interval (seconds):	30	(5 - 300)
Health Check Timeout (seconds):	5	(2 - 60)
Healthy Check Count Threshold:	3	(2 - 10)
Unhealthy Check Count Threshold:	5	(2 - 10)

The health check definition includes a URL to be queried for instance health. Override the default URL to match an existing resource in your application (e.g., `/myapp/index.jsp`) by entering it in the **Application Health Check URL** box.

The following list describes the health check parameters you can set for your application.

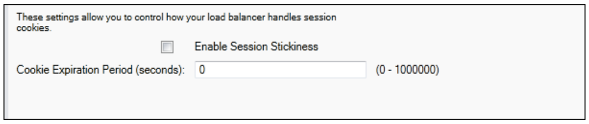
- For **Health Check Interval (seconds)**, enter the number of seconds Elastic Load Balancing waits between health checks for your application's Amazon EC2 instances.
- For **Health Check Timeout (seconds)**, specify the number of seconds Elastic Load Balancing waits for a response before it considers the instance unresponsive.
- For **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold**, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying **5** for **Unhealthy Check Count Threshold** means that the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check failed.

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer–generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer–generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If there is no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

You can use the **Sessions** section on the **Load Balancer** tab to specify whether or not the load balancer for your application allows session stickiness.



These settings allow you to control how your load balancer handles session cookies.

□ Enable Session Stickiness

Cookie Expiration Period (seconds): 0 (0 - 1000000)

For more information on Elastic Load Balancing, go to the Elastic Load Balancing Developer Guide.

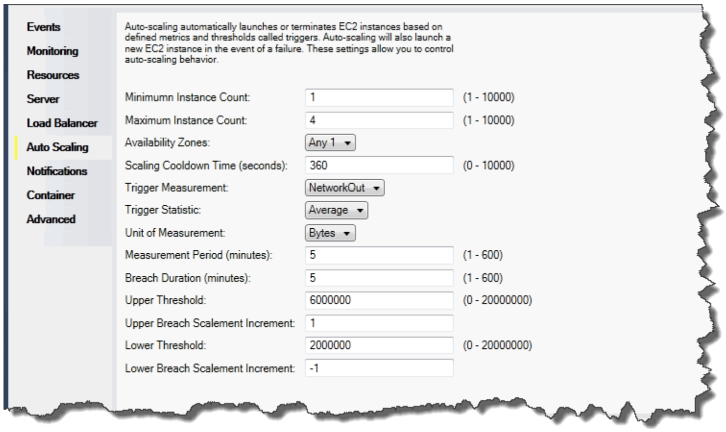
Configuring Auto Scaling Using the AWS Toolkit for Visual Studio

Auto Scaling is an Amazon web service designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. Users can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Auto Scaling for your application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Auto Scaling** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

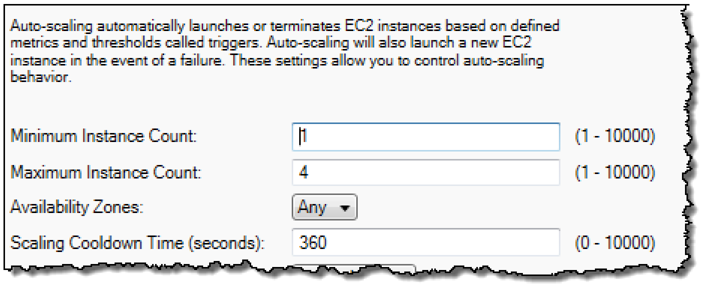
Minimumn Instance Count:	1	(1 - 10000)
Maximum Instance Count:	4	(1 - 10000)
Availability Zones:	Any 1	
Scaling Cooldown Time (seconds):	360	(0 - 10000)
Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	(0 - 20000000)
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	(0 - 20000000)
Lower Breach Scalement Increment:	-1	

The following section discusses how to configure Auto Scaling parameters for your application.

Launch the Configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Auto Scaling resources.

The **Minimum Instance Count** and **Maximum Instance Count** boxes let you specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:	1	(1 - 10000)
Maximum Instance Count:	4	(1 - 10000)
Availability Zones:	Any	
Scaling Cooldown Time (seconds):	360	(0 - 10000)

> **Note**
>
> To maintain a fixed number of Amazon EC2 instances, set **Minimum Instance Count** and **Maximum Instance Count** to the same value.

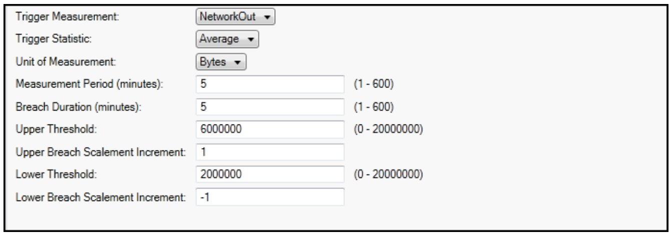
The **Availability Zones** box lets you specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications. If one Availability Zone goes down, your instances will still be running in your other Availability Zones.

> **Note**
>
> Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Auto Scaling mechanism that you set to tell the system when you want to increase (*scale out*) the number of instances, and when you want to decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch, such as CPU utilization, and determine if the conditions you specified have been met. When the upper or lower thresholds of the conditions you have specified for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *Scaling Activity*.

You can define a scaling trigger for your Elastic Beanstalk application using AWS Toolkit for Visual Studio.



Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	(0 - 20000000)
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	(0 - 20000000)
Lower Breach Scalement Increment:	-1	

Auto Scaling triggers work by watching a specific Amazon CloudWatch metric for an instance. Triggers include CPU utilization, network traffic, and disk activity. Use the **Trigger Measurement** setting to select a metric for your trigger.

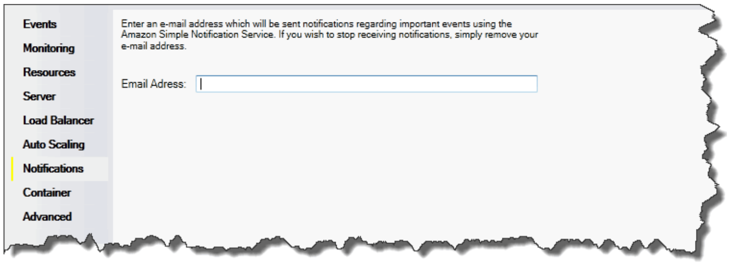
The following list describes the trigger parameters you can configure using the AWS Management Console.

- You can specify which statistic the trigger should use. You can select **Minimum**, **Maximum**, **Sum**, or **Average** for **Trigger Statistic**.
- For **Unit of Measurement**, specify the unit for the trigger measurement.
- The value in the **Measurement Period** box specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The **Breach Duration** is the amount of time a metric can be beyond its defined limit (as specified for the **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Upper Breach Scale Increment** and **Lower Breach Scale Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Auto Scaling, go to the Auto Scaling documentation.

Configuring Notifications Using AWS Toolkit for Visual Studio

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, simply enter your email address in the **Email Address** box. To disable these notifications, remove your email address from the box.



Enter an e-mail address which will be sent notifications regarding important events using the Amazon Simple Notification Service. If you wish to stop receiving notifications, simply remove your e-mail address.

Email Adress:

Configuring .NET Containers Using the AWS Toolkit for Visual Studio

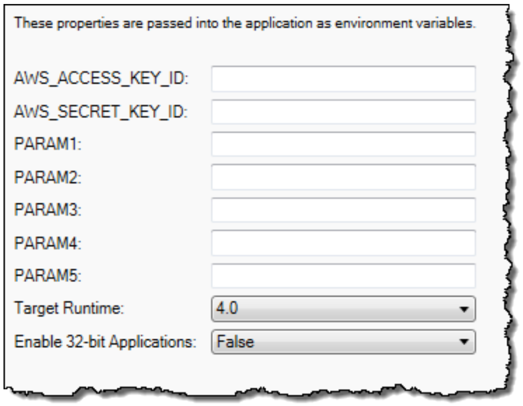
The **Container/.NET Options** panel lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can use the AWS Toolkit for Visual Studio to configure your container information.

> **Note**
> You can modify your configuration settings with zero downtime by swapping the CNAME for your environments. For more information, see Deploying a new Application Version with Zero Downtime (CNAME swap) (p. 83).

If you want to, you can extend the number of parameters. For information about extending parameters, see Option_settings (p. 166).

To access the Container/.NET Options panel for your Elastic Beanstalk application

1. In AWS Toolkit for Visual Studio, expand the Elastic Beanstalk node and your application node.
2. In **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the **Overview** pane, click the **Configuration** tab.
4. Under **Container**, you can configure container options.



These properties are passed into the application as environment variables.

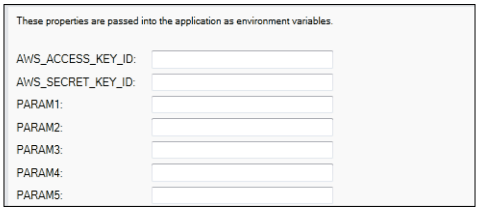
AWS_ACCESS_KEY_ID:
AWS_SECRET_KEY_ID:
PARAM1:
PARAM2:
PARAM3:
PARAM4:
PARAM5:
Target Runtime: 4.0
Enable 32-bit Applications: False

.NET Container Options

You can choose the version of .NET Framework for your application. Choose either 2.0 or 4.0 for **Target runtime**. Select **Enable 32-bit Applications** if you want to enable 32-bit applications.

Application Settings

This section of the **Container** panel lets you specify application settings. These settings enable greater portability by eliminating the need to recompile your source code as you move between environments.



These properties are passed into the application as environment variables.

AWS_ACCESS_KEY_ID:
AWS_SECRET_KEY_ID:
PARAM1:
PARAM2:
PARAM3:
PARAM4:
PARAM5:

You can configure the following application settings:

- Specify AWS credentials using the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_KEY** text boxes.

  Note
  Elastic Beanstalk uses instance profiles so that your application can use temporary security credentials to access AWS resources. To learn more, see Service Roles, Instance Profiles, and User Policies (p. 19).
- Specify up to five additional key-value pairs by entering them in the **PARAM** boxes.

  You might have a code snippet that looks similar to the following to access the keys and parameters:

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string param1 = appConfig["PARAM1"];
```

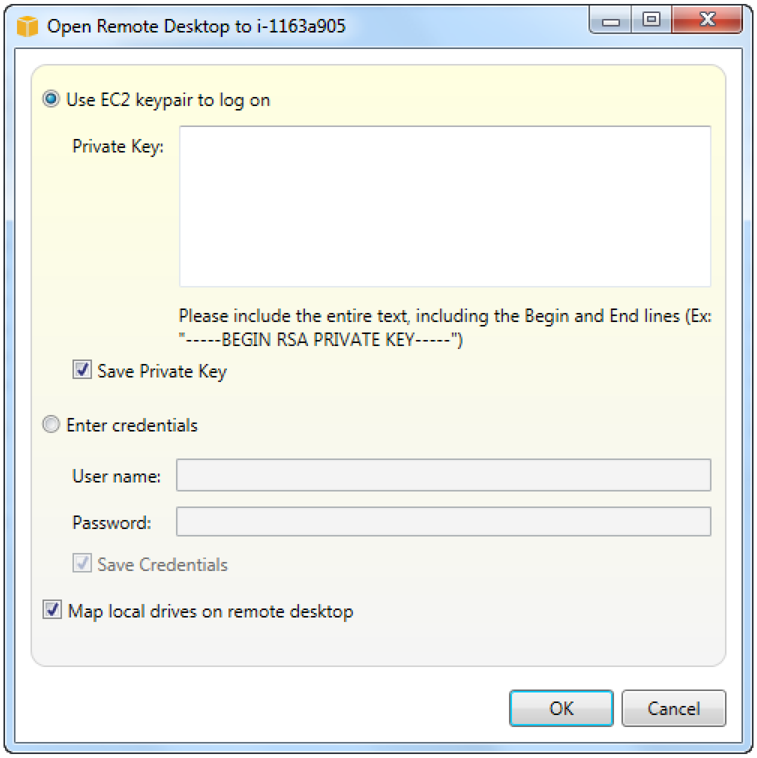
Note
These settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Listing and Connecting to Server Instances

You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Toolkit for Visual Studio or from the AWS Management Console. You can connect to these instances using Remote Desktop Connection. For information about listing and connecting to your server instances using the AWS Management Console, see Listing and Connecting to Server Instances (p. 280). The following section steps you through viewing and connecting you to your server instances using the AWS Toolkit for Visual Studio.

To view and connect to Amazon EC2 instances for an environment

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node and double-click **Instances**.
2. Right-click the instance ID for the Amazon EC2 instance running in your application's load balancer in the **Instance** column and select **Open Remote Desktop** from the context menu.



Open Remote Desktop to i-1163a905

● Use EC2 keypair to log on

Private Key:

Please include the entire text, including the Begin and End lines (Ex: "-----BEGIN RSA PRIVATE KEY-----")

☑ Save Private Key

○ Enter credentials

User name:

Password:

☑ Save Credentials

☑ Map local drives on remote desktop

3. Select **Use EC2 keypair to log on** and paste the contents of your private key file that you used to deploy your application in the **Private key** box. Alternatively, enter your user name and password in the **User name** and **Password** text boxes.

   Note
   If the key pair is stored inside the Toolkit, the text box does not appear.

4. Click **OK**.

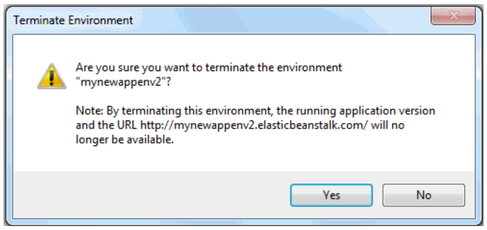
Terminating an Environment

To avoid incurring charges for unused AWS resources, you can terminate a running environment using the AWS Toolkit for Visual Studio.

Note
You can always launch a new environment using the same version later.

To terminate an environment

1. Expand the Elastic Beanstalk node and the application node in **AWS Explorer**. Right-click your application environment and select **Terminate Environment**.
2. When prompted, click **Yes** to confirm that you want to terminate the environment. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.



Terminate Environment

Are you sure you want to terminate the environment "mynewappenv2"?

Note: By terminating this environment, the running application version and the URL http://mynewappenv2.elasticbeanstalk.com/ will no longer be available.

Note
When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

Tools

Topics

The AWS SDK for .NET makes it even easier for Windows developers to build .NET applications that tap into the cost-effective, scalable, and reliable AWS cloud. Using the SDK, developers will be able to build solutions for AWS infrastructure services, including Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2). You can use the AWS Toolkit for Visual Studio to add the AWS .NET SDK to an existing project, or create a new .NET project based on the AWS .NET SDK.

AWS SDK for .NET

With the AWS SDK for .NET, you can get started in minutes with a single, downloadable package complete with Visual Studio project templates, the AWS .NET library, C# code samples, and documentation. You can build .NET applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides .NET developer-friendly APIs that hide much of the lower-level plumbing associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in C# for how to use the libraries to build applications. Online video tutorials and reference documentation are provided to help you learn how to use the libraries and code samples. For more information, go to AWS SDK for .NET.

AWS Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is a plug-in for .NET that makes it easier to develop and debug .NET applications using Amazon Web Services. You can get started quickly with building solutions for the AWS cloud using Visual Studio Project Templates. For more information about prerequisites, installation instructions and running code samples, go to AWS Toolkit for Microsoft Visual Studio.

Deploying Elastic Beanstalk Applications in .NET Using the Deployment Tool

The AWS Toolkit for Visual Studio includes a deployment tool, a command line tool that provides the same functionality as the deployment wizard in the AWS Toolkit. You can use the deployment tool in your build pipeline or in other scripts to automate deployments to Elastic Beanstalk.

The deployment tool supports both initial deployments and redeployments. If you previously deployed your application using the deployment tool, you can redeploy using the deployment wizard within Visual Studio. Similarly, if you have deployed using the wizard, you can redeploy using the deployment tool.

This chapter walks you through deploying a sample .NET application to Elastic Beanstalk using the deployment tool, and then redeploying the application using an incremental deployment. For a more in-depth discussion about the deployment tool, including the parameter options, go to Deployment Tool.

Prerequisites

In order to deploy your web application using the deployment tool, you need to package it as a `.zip` file. For more information about how to package your application for deployment, go to How to: Deploy a Web Application Project Using a Web Deployment Package at MSDN.

To use the deployment tool, you need to install the AWS Toolkit for Visual Studio. For information on prerequisites and installation instructions, go to AWS Toolkit for Microsoft Visual Studio.

The deployment tool is typically installed in one of the following directories on Windows:

32-bit	64-bit
C:\Program Files\AWS Tools\Deployment Tool\awsdeploy.exe	C:\Program Files (x86)\AWS Tools\Deployment Tool\awsdeploy.exe

Deploy to Elastic Beanstalk

To deploy the sample application to Elastic Beanstalk using the deployment tool, you first need to modify the `ElasticBeanstalkDeploymentSample.txt` configuration file, which is provided in the `Samples` directory. This configuration file contains the information necessary to deploy your application, including the application name, application version, environment name, and your AWS access credentials. After modifying the configuration file, you then use the command line to deploy the sample application. Your web deploy file is uploaded to Amazon S3 and registered as a new application version with Elastic Beanstalk. It will take a few minutes to deploy your application. Once the environment is healthy, the deployment tool outputs a URL for the running application.

To deploy a .NET application to Elastic Beanstalk

1. From the `Samples` subdirectory where the deployment tool is installed, open `ElasticBeanstalkDeploymentSample.txt` and enter your AWS access key and AWS secret key as in the following example.

   For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see How Do I Get Security Credentials? in the *AWS General Reference*.

   ```
   ### AWS Access Key and Secret Key used to create and deploy the application
    instance
   AWSAccessKey = AKIAIOSFODNN7EXAMPLE
   AWSSecretKey = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   ```

2. At the command line prompt, type the following:

```
C:\Program Files (x86)\AWS Tools\Deployment Tool>awsdeploy.exe /w
Samples\ElasticBeanstalkDeploymentSample.txt
```

It takes a few minutes to deploy your application. If the deployment succeeds, you will see the message, `Application deployment completed; environment health is Green`.

> **Note**
>
> If you receive the following error, the CNAME already exists.
>
> ```
> [Error]: Deployment to AWS Elastic Beanstalk failed with exception:
> DNS name (MyAppEnv.elasticbeanstalk.com) is not available.
> ```
>
> Because a CNAME must be unique, you need to change `Environment.CNAME` in `ElasticBeanstalkDeploymentSample.txt`.

3. In your web browser, navigate to the URL of your running application. The URL will be in the form <CNAME.elasticbeanstalk.com> (e.g., **`MyAppEnv.elasticbeanstalk.com`**).

Redeploy to Elastic Beanstalk

You can redeploy your application using an incremental deployment. Incremental deployments are faster because you are updating only the files that have changed instead of all the files. This section walks you through redeploying the sample application you deployed in Deploy to Elastic Beanstalk (p. 663).

To edit and redeploy a .NET application to Elastic Beanstalk

1. Extract `AWSDeploymentSampleApp.zip` from the Samples directory to a location on your computer such as `c:\mydeploymentarchive\AWSDeploymentSampleApp`.
2. Modify one of the files in the `AWSDeploymentSampleApp` directory. For example, you can modify the title in `default.aspx`.
3. In the `ElasticBeanstalkDeploymentSample.txt` configuration file, do the following:

   - Specify the location where you extracted the files. This means modifying the value for the `DeploymentPackage` key in the `Incremental Deployment Settings` section in `ElasticBeanstalkDeploymentSample.txt`. For example:

     ```
     C:\mydeploymentarchive\AWSDeploymentSampleApp
     ```

   - Remove the # in front of `IncrementalPushRepository` and `DeploymentPackage`.
   - Add a # in front of `DeploymentPackage` in the `Non-Incremental Deployment Settings`.

4. At the command line, type the following:

   ```
   C:\Program Files (x86)\AWS Tools\Deployment Tool>awsdeploy.exe /r
   Samples\ElasticBeanstalkDeploymentSample.txt
   ```

   If this command succeeds, you should see something similar to the following:

   ```
   ...environment 'MyAppEnvironment' found and available for redeployment
   (configuration parameters not required for redeployment will be ignored)
   ...starting redeployment to AWS Elastic Beanstalk environment 'MyAppEnviron
   ment'
   ```

```
...starting incremental deployment to environment 'MyAppEnvironment'
...finished incremental deployment in 9199.9199 ms
```

5. In your web browser, navigate to the same URL as in Deploy to Elastic Beanstalk (p. 663). You should see your updated application.

Resources

There are several places you can go to get additional help when developing your .NET applications:

Resource	Description
.NET Development Forum	Post your questions and get feedback.
.NET Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.
AWS SDK for .NET Documentation	Read about setting up the SDK and running code samples, features of the SDK, and detailed information about the API operations for the SDK.

Deploying Node.js Applications to AWS Elastic Beanstalk

Topics

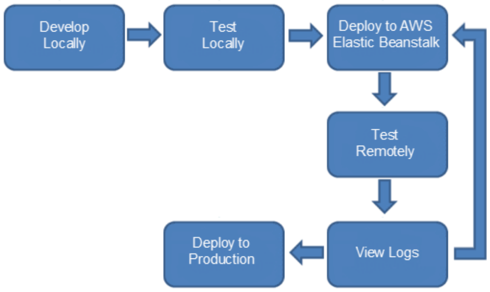
Elastic Beanstalk for Node.js makes it easy to deploy, manage, and scale your Node.js web applications using Amazon Web Services. Elastic Beanstalk for Node.js is available to anyone developing or hosting a web application using Node.js. This section provides step-by-step instructions for deploying your Node.js web application to Elastic Beanstalk using EB Command Line Interface (CLI). This chapter also provides walkthroughs for common frameworks such as Express and Geddy.

After you deploy your Elastic Beanstalk application, you can continue to use EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs. You can also use the Elastic Beanstalk console to upload your Node.js files using a .zip file. For more information, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

> **Note**
> When support for the version of Node.js that you are using is removed from the platform configuration, you must change the version setting prior to doing a platform upgrade (p. 90). This may occur when a security vulnerability is identified for one or more versions of Node.js When this occurs, attempting to upgrade to a new version of the platform that does not support the configured NodeVersion (p. 123) will fail. To avoid needing to create a new environment, change the *NodeVersion* configuration option to a version that is supported by both the old configuration version and the new one, and then perform the platform upgrade.

Develop, Test, and Deploy

The following diagram illustrates a typical software development life cycle including deploying your application to Elastic Beanstalk.



Typically, after developing and testing your application locally, you will deploy your application to Elastic Beanstalk. At this point, your application will be live at a URL such as http://myexampleapp-wpams3yrvj.elasticbeanstalk.com. Because your application will be live, you should consider setting up multiple environments, such as a testing environment and a production environment. You can point your domain name to the Amazon Route 53 (a highly available and scalable Domain Name System (DNS) web service) CNAME <*yourappname*>.elasticbeanstalk.com. Contact your DNS provider to set this up. For information about how to map your root domain to your Elastic Load Balancer, see Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295). After you remotely test and debug your Elastic Beanstalk application, you can then make any updates and redeploy to Elastic Beanstalk. After you are satisfied with all of your changes, you can upload your latest version to your production environment. The following sections provide more details explaining each stage of the software development life cycle.

Get Set Up

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Develop Locally

After installing EB CLI 3.x on your local computer, you use the Git command line as you normally would to create your local repository and add and commit changes. You create your Node.js application as you normally would with your favorite editor. If you don't already have a Node.js application ready, you can use a simple "Hello World" application. Type the following program into your favorite editor, and save it as `server.js` in your root project directory.

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(process.env.PORT || 8888);
```

Next, test your program, add it to your repository, and commit your change.

```
node server.js
git add server.js
git commit -m "initial check-in"
```

> **Note**
> For information about Git commands, go to Git - Fast Version Control System.

Test Locally

Normally, at this point you would test your application locally before deploying to Elastic Beanstalk. Suppose you find a few issues you would like to fix. Using the above "Hello World" application, add a "!" after "Hello World" and check in your changes. Update your `server.js` file, and then type the following commands to check in your updated file.

```
node server.js
git add server.js
git commit -m "my second check-in"
```

After you commit your changes, you should see a response similar to the following:

```
[master 0535814] my second check-in
1 files changed, 1 insertions(+), 1 deletions(-)
```

Note the commit ID that is generated. This ID is used to generate a version label for your application.

Deploy to AWS Elastic Beanstalk

After testing your application locally, use the EB CLI to deploy it to Elastic Beanstalk

If you get a "command not found" error when you run `eb --version`, ensure that the directory that contains the `eb` executable is in your PATH environment variable

Linux, OS X, or Unix

```
$ export PATH=/eb/executable/directory:$PATH
```

Windows

```
> set PATH=%PATH%;C:\eb\executable\directory
```

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application to use, type the number corresponding to the option **Create new Application**. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use HelloWorld.

   ```
   Enter Application Name (default is "tmp-dev"): HelloWorld
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.
4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.
5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```

6. Create your running environment.

   ```
   eb create
   ```

7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   Note
   If you have a space in your application name, make sure you do not have a space in your environment name.
8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

   ```
   Enter DNS CNAME prefix
   (default is HelloWorld):
   ```

EB CLI will display your environment details and the status of the `create` operation.

View the application by typing the following:

```
eb open
```

To update the sample application with your local application

1. Make changes to your code, and then type the following command:

   ```
   eb deploy
   ```

   Elastic Beanstalk will attempt to start app.js, then server.js, and then "npm start" in that order.
2. If everything worked as expected, you should see something similar to the following:

   ```
   Counting objects: 5, done.
   Delta compression using up to 4 threads.
   Compressing objects:100% (2/2), done.
   Writing objects: 100% (3/3), 298 bytes, done.
   Total 3 (delta 1), reused 0 (delta 0)
   To https://<some long string>@git.elasticbeanstalk.us-west-
   2.amazon.com/helloworld/helloworldenv
    44c7066..b1f11a1 master -> master
   ```

3. Verify that your application has been updated by refreshing your web browser.

Debug/View Logs

You can use the `eb logs` command to investigate any issues. If you do not specify any flags with the command, logs will be displayed in the command window. For other ways to retrieve logs using this command, see logs (p. 428).

Edit the Application and Redeploy

Now that you have tested your application, it is easy to edit your application, redeploy, and see the results in moments. First, make changes to your application and commit your changes. Then deploy a new application version to your existing Elastic Beanstalk environment.

```
git add server.js
git commit -m "my third check-in"
eb deploy
```

A new application version will be uploaded to your Elastic Beanstalk environment.

You can use the AWS Management Console, CLI, or APIs to manage your Elastic Beanstalk environment. For more information, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

You can also configure Git to push from a specific branch to a specific environment. For more information, see Deploying a Git Branch to a Specific Environment (p. 451).

Deploy to Production

When you are satisfied with all of the changes you want to make to your application, you can deploy it to your production environment. To deploy your application to a new environment, do the following:

1. Commit your changes
2. Create a branch

3. Create and launch your new environment
4. Deploy your application to your new production environment

When you update your application using EB CLI, Elastic Beanstalk will create a new application version. For information on how to deploy an already existing application version to a new environment, see Launching a New AWS Elastic Beanstalk Environment (p. 55). The following steps walk you through committing your new changes and then updating your environment with a new application version using EB CLI and Git.

To deploy to production using EB CLI

1. Commit your changes.

   ```
   git add .
   git commit -m "final checkin"
   ```

2. Create a branch and switch to it.

   ```
   git checkout -b prodenv
   eb use prod
   ```

3. When prompted, type your new environment name, and accept all settings from your previous environment.
4. When you are ready, deploy your new application version to Elastic Beanstalk.

   ```
   eb deploy
   ```

Deploy an Existing Application Version to an Existing Environment

If you need to deploy an existing application to an existing environment, you can do so using the AWS Management Console, CLI, or APIs. You may want to do this if, for instance, you need to roll back to a previous application version. For instructions on how to deploy an existing application version to an existing environment, see Deploying Applications to AWS Elastic Beanstalk Environments (p. 78).

Deploying an Express Application to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using EB CLI and Git, and then updating the application to use the Express framework.

> **Note**
> This example uses Amazon RDSs, and you may be charged for its usage. For more information about pricing, go to Amazon Relational Database Service (RDS) Pricing. If you are a new customer, you can make use of the AWS Free Usage Tier. For details, go to AWS Free Usage Tier.

Step 1: Set Up Your Git Repository

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Step 2: Set Up Your Express Development Environment

Set up Express and create the project structure. The following walks you through setting up Express on a Linux operating system.

To set up your Express development environment on your local computer

1. Install node.js. For instructions, go to http://nodejs.org/. Verify you have a successful installation before proceeding to the next step.

   ```
   $ node -v
   ```

   Note
   For information about what Node.js versions are supported, see Supported Platforms (p. 24).

2. Create a directory for your express application.

   ```
   $ mkdir node-express
   $ cd node-express
   ```

3. Install npm if you don't already have it installed. Here's one example of how to install npm.

   ```
   node-express# cd . && yum install npm
   ```

4. Install Express globally so that you have access to the `express` command.

   ```
   node-express# npm install -g express-generator
   ```

5. Depending on your operating system, you may need to set your path to run the `express` command. If you need to set your path, use the output from the previous step when you installed Express. The following is an example.

   ```
   node-express# export:PATH=$PATH:/usr/local/share/npm/bin/express
   ```

6. Run the `express` command. This generates `package.json`.

   ```
   node-express# express
   ```

When prompted if you want to continue, type **y**.

7. Set up local dependencies.

   ```
   node-express# cd . && npm install
   ```

8. Verify it works.

   ```
   node-express# npm start
   ```

   You should see output similar to the following:

   ```
   Express server listening on port 3000
   ```

   Press **Ctrl+C** to stop the server.
9. Initialize the Git repository.

   ```
   node-express# git init
   ```

10. Edit the `.gitignore` file and add the following files and directories to it. These files will be excluded from being added to the repository. This step is not required, but it is recommended.

    ```
    node-express# cat > .gitignore <<EOT
    node_modules/
    .gitignore
    .elasticbeanstalk/
    EOT
    ```

Step 3: Configure Elastic Beanstalk

The following instructions use the Elastic Beanstalk command line interface (p. 384) (EB CLI) to configure an Elastic Beanstalk application repository in your local project directory.

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application to use, type the number corresponding to the option **Create new Application**. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use expressapp.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "node-express"): expressapp
   ```

> **Note**
> If you have a space in your application name, make sure you do not use quotation marks.

4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.
5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```

6. Create your running environment.

   ```
   eb create
   ```

7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   > **Note**
   > If you have a space in your application name, make sure you do not have a space in your environment name.

8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

   ```
   Enter DNS CNAME prefix
   (default is HelloWorld):
   ```

EB CLI will display your environment details and the status of the `create` operation.

To deploy a sample application

- From the directory where you created your local repository, type the following command:

  ```
  eb deploy
  ```

  This process may take several minutes to complete. Elastic Beanstalk will provide status updates during the process. If at any time you want to stop polling for status updates, press **Ctrl+C**. Once the environment status is Green, Elastic Beanstalk will output a URL for the application. You can copy and paste the URL into your web browser to view the application.

Step 4: View the Application

To view the application

- To open your application in a browser window, type the following:

```
eb open
```

Step 5: Update the Application

After you have deployed a sample application, you can update it with your own application. In this step, we update the sample application to use the Express framework. You can download the final source code from http://elasticbeanstalk-samples-us-east-1.s3.amazonaws.com/nodejs-example-express.zip.

To update your application to use Express

1. Stage the files.

   ```
   node-express# git add .
   node-express# git commit -m "First express app"
   ```

2. Deploy the changes.

   ```
   node-express# eb deploy
   ```

3. Update your configuration to run `npm start` to start the application server. Run **`eb config`**, modify the configuration entry for `aws:elasticbeanstalk:container:nodejs:NodeCommand` to equal `'npm start'` instead of `null`, and then save and close the file to apply the settings and update your environment.

   ```
   node-express# eb config
   ApplicationName: node-express
   DateUpdated: 2015-02-26 23:52:39+00:00
   EnvironmentName: node-express-dev
   settings:
     AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
       LowerBreachScaleIncrement: '-1'
     AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
       UpperBreachScaleIncrement: '1'
     AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
       UpperThreshold: '6000000'
     AWSEBCloudwatchAlarmLow.aws:autoscaling:trigger:
       BreachDuration: '5'
       EvaluationPeriods: '1'
   ...
     aws:elasticbeanstalk:container:nodejs:
       GzipCompression: 'false'
       NodeCommand: 'npm start'
   ```

4. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says "Welcome to Express".

Next, let's update the Express application to serve static files and add a new page.

To configure static files and add a new page to your Express application

1. On your local computer, create an `.ebextensions` directory in the top-level directory of your source bundle. In this example, we use `node-express/.ebextensions`.
2. Create a configuration file, `/node-express/.ebextensions/static.config`. For more information about the configuration file, see Customizing and Configuring a Node.js Environment (p. 684). Type the following inside the configuration file to configure static files:

```
option_settings:
  - namespace: aws:elasticbeanstalk:container:nodejs:staticfiles
    option_name: /public
    value: /public
```

3. On your local computer, comment out the static mapping in `node-express/app.js`. This step is not required, but it is a good test to see if the static mappings are configured correctly.

```
//  app.use(express.static(path.join(__dirname, 'public')));
```

4. Add your updated files to your local repository and commit your changes.

```
node-express# git add .ebextensions/ app.js
node-express# git commit -m "Making stylesheets be served by nginx."
```

5. On your local computer, add `node-express/routes/hike.js`. Type the following:

```
exports.index = function(req, res) {
 res.render('hike', {title: 'My Hiking Log'});
};

exports.add_hike = function(req, res) {
};
```

6. On your local computer, update `node-express/app.js` to include three new lines.

   First, add the following line to add a `require` for this route:

```
 , hike = require('./routes/hike');
```

   Your file should look similar to the following snippet:

```
var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , http = require('http')
  , path = require('path')
  , hike = require('./routes/hike');
```

   Then, add the following two lines to `node-express/app.js` after app.get('/users', users.list);

```
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

Your file should look similar to the following snippet:

```
app.get('/', routes.index);
app.get('/users', user.list);
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

7. On your local computer, copy `node-express/views/index.jade` to `node-express/views/hike.jade`.

```
node-express# cp views/index.jade views/hike.jade
```

8. Add your files to the local repository, commit your changes, and deploy your updated application.

```
node-express# git add .
node-express# git commit -m "added new file"
node-express# eb deploy
```

9. Your environment will be updated after a few minutes. After your environment is green and ready, verify it worked by refreshing your browser and appending **`hikes`** at the end of the URL (e.g., http://node-express-env-syypntcz2q.elasticbeanstalk.com/hikes).

   You should see a web page titled **My Hiking Log**.

Next, let's update the application to add a database.

To update your application with a database

1. On your local computer, update `node-express/app.js` to add the database information and add a record to the database. You can also copy app.js from http://elasticbeanstalk-samples-us-east-1.s3.amazonaws.com/nodejs-example-express.zip.

```
/**
 * Module dependencies.
 */

var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , hike = require('./routes/hike')
  , http = require('http')
  , path = require('path')
  , mysql = require('mysql')
  , async = require('async');

var app = express();

app.configure(function(){
  app.set('port', process.env.PORT || 3000);
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  app.use(express.favicon());
  app.use(express.logger('dev'));
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(app.router);
//  app.use(express.static(path.join(__dirname, 'public')));
});

app.configure('development', function() {
  console.log('Using development settings.');
  app.set('connection', mysql.createConnection({
    host: '',
    user: '',
    port: '',
    password: ''}));
  app.use(express.errorHandler());
});

app.configure('production', function() {
  console.log('Using production settings.');
  app.set('connection', mysql.createConnection({
    host: process.env.RDS_HOSTNAME,
    user: process.env.RDS_USERNAME,
    password: process.env.RDS_PASSWORD,
    port: process.env.RDS_PORT}));
});

function init() {
  app.get('/', routes.index);
  app.get('/users', user.list);
  app.get('/hikes', hike.index);
  app.post('/add_hike', hike.add_hike);

  http.createServer(app).listen(app.get('port'), function(){
    console.log("Express server listening on port " + app.get('port'));
  });
}

var client = app.get('connection');
async.series([
  function connect(callback) {
    client.connect(callback);
  },
  function clear(callback) {
    client.query('DROP DATABASE IF EXISTS mynode_db', callback);
  },
  function create_db(callback) {
    client.query('CREATE DATABASE mynode_db', callback);
  },
  function use_db(callback) {
    client.query('USE mynode_db', callback);
  },
  function create_table(callback) {
     client.query('CREATE TABLE HIKES (' +
                         'ID VARCHAR(40), ' +
                         'HIKE_DATE DATE, ' +
                         'NAME VARCHAR(40), ' +
                         'DISTANCE VARCHAR(40), ' +
                         'LOCATION VARCHAR(40), ' +
                         'WEATHER VARCHAR(40), ' +
                         'PRIMARY KEY(ID))', callback);
  },
  function insert_default(callback) {
    var hike = {HIKE_DATE: new Date(), NAME: 'Hazard Stevens',
         LOCATION: 'Mt Rainier', DISTANCE: '4,027m vertical', WEATHER:'Bad'};

    client.query('INSERT INTO HIKES set ?', hike, callback);
  }
], function (err, results) {
  if (err) {
    console.log('Exception initializing database.');
    throw err;
  } else {
    console.log('Database initialization complete.');
    init();
  }
});
```

2. On your local computer, update `node-express/views/hike.jade` to display a record from the database.

```
extends layout

block content
  h1= title
  p Welcome to #{title}

  div
    h3 Hikes
    table(border="1")
      tr
        td Date
        td Name
        td Location
        td Distance
        td Weather
      each hike in hikes
        tr
          td #{hike.HIKE_DATE.toDateString()}
          td #{hike.NAME}
          td #{hike.LOCATION}
          td #{hike.DISTANCE}
          td #{hike.WEATHER}
```

3. On your local computer, update `node-express/routes/hike.js` to configure the route to show the record.

```
var uuid = require('node-uuid');

exports.index = function(req, res) {
  res.app.get('connection').query( 'SELECT * FROM HIKES', function(err,
rows) {
    if (err) {
      res.send(err);
    } else {
      console.log(JSON.stringify(rows));
      res.render('hike', {title: 'My Hiking Log', hikes: rows});
  }});
};

exports.add_hike = function(req, res){
};
```

4. On your local computer, update `node-express/package.json` to add dependencies.

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app"
  },
  "dependencies": {
    "express": "3.1.0",
    "jade": "*",
    "mysql": "*",
    "async": "*",
    "node-uuid": "*"
  }
}
```

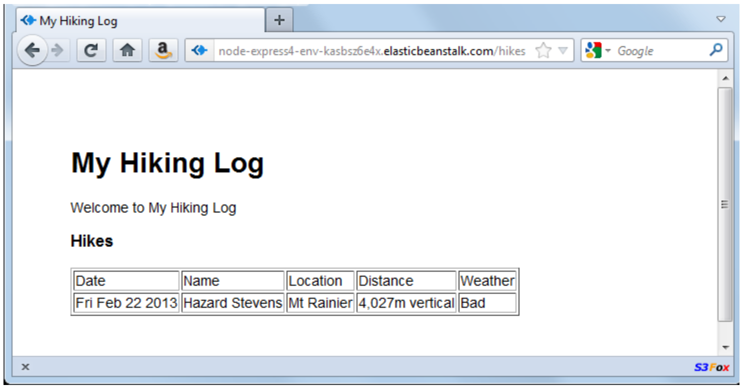
5. On your local computer, update `node-express/.ebextensions/static.config` to add a production flag to the environment variables.

```
option_settings:
  - namespace: aws:elasticbeanstalk:container:nodejs:staticfiles
    option_name: /public
    value: /public
  - option_name: NODE_ENV
    value: production
```

6. Add your files to the local repository, commit your changes, and deploy your updated application.

```
node-express# git add .
node-express# git commit -m "updated files"
node-express# eb deploy
```

7. Your environment will be updated after a few minutes. After your environment is green and ready, verify it worked by refreshing your URL. Remember to append **`hikes`** at the end of the URL. You should see the following page.



My Hiking Log

Welcome to My Hiking Log

Hikes

Date	Name	Location	Distance	Weather
Fri Feb 22 2013	Hazard Stevens	Mt Rainier	4,027m vertical	Bad

Next, update the application to accept new entries and display records from the database.

To update the application to allow new entries into the database

1. On your local computer, update `node-express/views/hike.jade` so the user can enter new entries. Add the form block inside the `block content`.

```
extends layout

block content
  h1= title
  p Welcome to #{title}

  form(action="/add_hike", method="post")
    table(border="1")
      tr
        td Your Name
        td
          input(name="hike[NAME]", type="textbox")
      tr
        td Location
        td
          input(name="hike[LOCATION]", type="textbox")
      tr
        td Distance
        td
          input(name="hike[DISTANCE]", type="textbox")
      tr
        td Weather
        td
          input(name="hike[WEATHER]", type="radio", value="Good")
          | Good
          input(name="hike[WEATHER]", type="radio", value="Bad")
          | Bad
         input(name="hike[WEATHER]", type="radio", value="Seattle", checked)

          | Seattle
      tr
        td(colspan="2")
          input(type="submit", value="Record Hike")

  div
```

```
    h3 Hikes
    table(border="1")
      tr
        td Date
        td Name
        td Location
        td Distance
        td Weather
      each hike in hikes
        tr
          td #{hike.HIKE_DATE.toDateString()}
          td #{hike.NAME}
          td #{hike.LOCATION}
          td #{hike.DISTANCE}
          td #{hike.WEATHER}
```

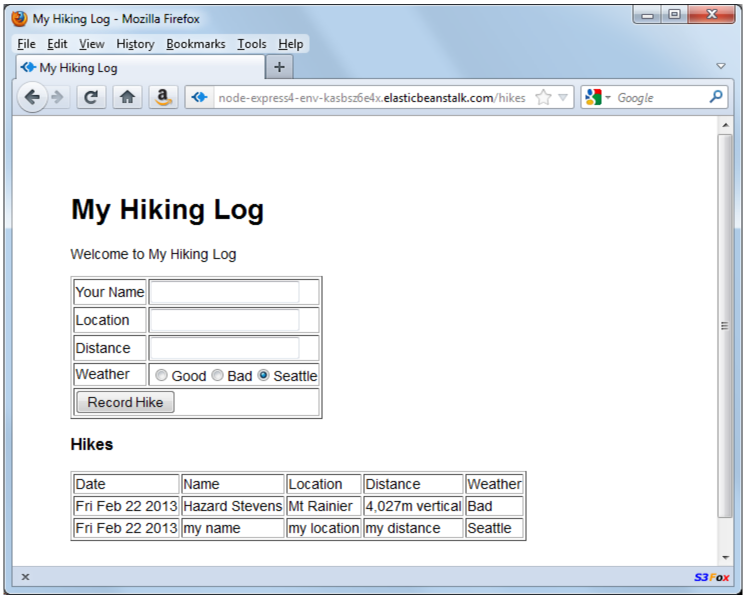
2. On your local computer, update `node-express/routes/hike.js` to accept new entries. Update `exports.add_hike` to be the following.

```
exports.add_hike = function(req, res){
  var input = req.body.hike;
  var hike = { HIKE_DATE: new Date(), ID: uuid.v4(), NAME: input.NAME,
     LOCATION: input.LOCATION, DISTANCE: input.DISTANCE, WEATHER: input.WEATH
ER};

  console.log('Request to log hike:' + JSON.stringify(hike));
  req.app.get('connection').query('INSERT INTO HIKES set ?', hike, func
tion(err) {
    if (err) {
      res.send(err);
    } else {
      res.redirect('/hikes');
    }
  });
};
```

3. Add your files to the local repository, commit your changes, and deploy your updated application.

```
node-express# git add .
node-express# git commit -m "added new file"
node-express# eb deploy
```

4. Your environment will be updated after a few minutes. After your environment is green and ready, verify it worked by refreshing your URL and adding a couple of entries. Remember to append **`hikes`** at the end of the URL. You should see a page similar to the following diagram.


My Hiking Log - Mozilla Firefox
File Edit View History Bookmarks Tools Help
My Hiking Log
node-express4-env-kasbsz6e4x.elasticbeanstalk.com/hikes
Google
S3Fox

My Hiking Log

Welcome to My Hiking Log

Your Name	
Location	
Distance	
Weather	○ Good ○ Bad ● Seattle
Record Hike	

Hikes

Date	Name	Location	Distance	Weather
Fri Feb 22 2013	Hazard Stevens	Mt Rainier	4,027m vertical	Bad
Fri Feb 22 2013	my name	my location	my distance	Seattle

Step 6: Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `terminate` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

- From the directory where you created your local repository, type the following command:

```
eb terminate
```

  This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

  > **Note**
  > If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Configuring Node.js Containers with Elastic Beanstalk

You can fine-tune the behavior of your Amazon EC2 instances by using a configuration file to configure your container settings. For instructions on customizing and configuring a Node.js container, see Customizing and Configuring a Node.js Environment (p. 684). For a list of container options, see Node.js Container Options (p. 123).

The **Container/Node.js Options** configuration also lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the AWS Management Console.

Customizing and Configuring a Node.js Environment

When deploying your Node.js application, you may want to customize and configure the behavior of your EC2 instances. You can easily customize your instances at the same time that you deploy your application version by including a configuration file with your source bundle. This section walks you through the process of creating a configuration file and bundling it with your source. For an example walkthrough using configuration files, see Deploying an Express Application to Elastic Beanstalk (p. 671).

To customize and configure your Node.js environment

1. Create a configuration file with the extension `.config` (e.g., `myapp.config`) and place it in an `.ebextensions` top-level directory of your source bundle. You can have multiple configuration files in your `.ebextensions` directory. These files are executed in alphabetical order. For example, `.ebextensions/01run.config` is executed before `.ebextensions/02do.config`.

   Note
   Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

   The following is an example snippet of a configuration file. For a full list of Node.js container options, see Node.js Container Options (p. 123).

   ```
   option_settings:
     - namespace: aws:elasticbeanstalk:container:nodejs
       option_name: ProxyServer
       value: nginx
     - namespace: aws:elasticbeanstalk:container:nodejs:staticfiles
       option_name: /public
       value: /public
   ```

   Note
   You can specify any key-value pairs in the `aws:elasticbeanstalk:application:environment` namespace, and they will be passed in as environment variables on your EC2 instances.

2. Create a `package.json` file and place it in the top-level directory of your source bundle. A typical Node.js application will have dependencies on other third-party packages. You specify all the packages you need (as well as their versions) in a single package.json file. For more information about the requirements file, see package.json on the NPM website. The following is an example package.json file for the Express framework.

   ```
   {
     "name": "application-name",
     "version": "0.0.1",
     "private": true,
     "scripts": {
       "start": "node app"
   ```

```
  },
  "dependencies": {
    "express": "3.1.0",
    "jade": "*",
    "mysql": "*",
    "async": "*",
    "node-uuid": "*"
  }
}
```

3. Deploy your application version.

For an example walkthrough of deploying an Express application, see Deploying an Express Application to Elastic Beanstalk (p. 671) . For an example walkthrough of deploying a Geddy application with Amazon ElastiCache, see Deploying a Geddy Application with Clustering to Elastic Beanstalk (p. 690).

Accessing Environment Configuration Settings

Inside the Node.js environment running in AWS Elastic Beanstalk, you can access the environment variables using `process.env.ENV_VARIABLE` similar to the following example.

```
process.env.PARAM1
process.env.PARAM2
```

> **Note**
> Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

For a list of configuration settings, see Node.js Container Options (p. 123).

Example: Using Configuration Files to Configure Nginx and Apache

You can use configuration files to make modifications to Apache. For example, if you want to configure Nginx or Apache to server application/json gzipped, which is not on by default, you would create a configuration file with the following snippets.

Example 1. Example configuring Nginx

```
files:
  /etc/nginx/conf.d/gzip.conf:
    content: |
      gzip_types application/json;
```

Example 2. Example configuring Apache

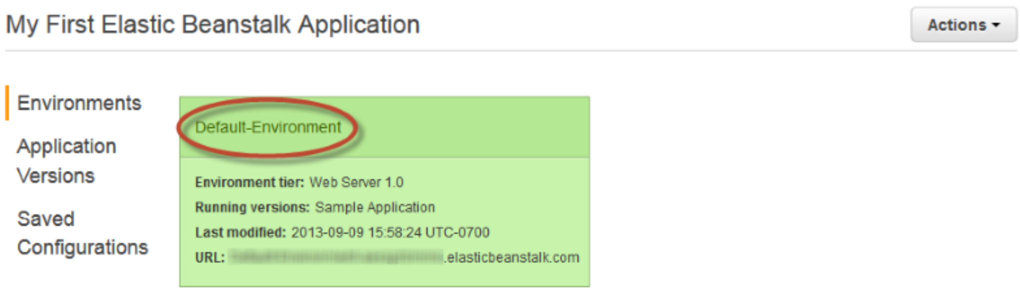
```
files:
  /etc/httpd/conf.d/gzip.conf:
    content: |
      AddOutputFilterByType DEFLATE application/json
```

AWS Management Console

The Node.js settings lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the AWS Management Console.

To access the Node.js container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

Container Options

On the configuration page, specify the following:

- **Proxy Server**–Specifies which web server to use to proxy connections to Node.js. By default, Nginx is used. If you select **none**, static file mappings do not take affect, and gzip compression is disabled.
- **Node Version**–Specifies the version of Node.js. For information about what versions are supported, see Supported Platforms (p. 24).

  Note
  When support for the version of Node.js that you are using is removed from the platform configuration, you must change the version setting prior to doing a platform upgrade (p. 90). This may occur when a security vulnerability is identified for one or more versions of Node.js When this occurs, attempting to upgrade to a new version of the platform that does not support the configured NodeVersion (p. 123) will fail. To avoid needing to create a new environment, change the *NodeVersion* configuration option to a version that is supported by both the old configuration version and the new one, and then perform the platform upgrade.

- **Gzip Compression**–Specifies whether gzip compression is enabled. By default, gzip compression is enabled.
- **Node Command**–Lets you enter the command used to start the Node.js application. An empty string (the default) means Elastic Beanstalk will use `app.js`, then `server.js`, and then `npm start` in that order.

Log Options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3**–Specifies whether log files for your application's Amazon EC2 instances should be copied to your Amazon S3 bucket associated with your application.

Static Files

To improve performance, you may want to configure Nginx or Apache to server static files (for example, HTML or images) from a set of directories inside your web application. You can set the virtual path and directory mapping on the **Container** tab in the **Static Files** section. To add multiple mappings, click **Add Path**. To remove a mapping, click **Remove**.

Environment Properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Accessing Environment Configuration Settings

Inside the Node.js environment running in AWS Elastic Beanstalk, you can access the environment variables using `process.env.ENV_VARIABLE` similar to the following example.

```
process.env.PARAM1
process.env.PARAM2
```

> **Note**
> Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

For a list of configuration settings, see Node.js Container Options (p. 123).

Command Line Interface (CLI)

To edit an application's environment configuration settings

- Update an application's environment configuration settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:container:nodejs:staticfiles",
        "OptionName": "/public",
        "Value": "/public"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:nodejs",
        "OptionName": "ProxyServer",
        "Value": "nginx"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:nodejs",
        "OptionName": "NodeCommand",
        "Value": ""
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:nodejs",
        "OptionName": "NodeVersion",
        "Value": "0.8.18"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:nodejs",
        "OptionName": "GzipCompression",
        "Value": "true"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_ACCESS_KEY_ID",
        "Value": "AKIAIOSFODNN7EXAMPLE"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_SECRET_KEY",
        "Value": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "myvar",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To edit an application's environment configuration settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:container:nodejs:staticfiles`
  - *`OptionSettings.member.1.OptionName`* = `/public`
  - *`OptionSettings.member.1.Value`* = `/public`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:container:nodejs`
  - *`OptionSettings.member.2.OptionName`* = `ProxyServer`
  - *`OptionSettings.member.2.Value`* = `nginx`

- *OptionSettings.member.3.Namespace* = aws:elasticbeanstalk:container:nodejs
- *OptionSettings.member.3.OptionName* = NodeCommand
- *OptionSettings.member.3.Value* = ""
- *OptionSettings.member.4.Namespace* = aws:elasticbeanstalk:container:nodejs
- *OptionSettings.member.4.OptionName* = NodeVersion
- *OptionSettings.member.4.Value* = 0.8.18
- *OptionSettings.member.5.Namespace* = aws:elasticbeanstalk:container:nodejs
- *OptionSettings.member.5.OptionName* = GzipCompression
- *OptionSettings.member.5.Value* = true
- *OptionSettings.member.6.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.6.OptionName* = AWS_ACCESS_KEY_ID
- *OptionSettings.member.6.Value* = AKIAIOSFODNN7EXAMPLE
- *OptionSettings.member.7.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.7.OptionName* = AWS_SECRET_KEY
- *OptionSettings.member.7.Value* = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
- *OptionSettings.member.8.Namespace* = aws:elasticbeanstalk:application:environment
- *OptionSettings.member.8.OptionName* = myvar
- *OptionSettings.member.8.Value* = somevalue
- *OptionSettings.member.9.Namespace* = aws:elasticbeanstalk:hostmanager
- *OptionSettings.member.9.OptionName* = LogPublicationControl
- *OptionSettings.member.9.Value* = false

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3An
odejs%3Astaticfiles
&OptionSettings.member.1.OptionName=/public
&OptionSettings.member.1.Value=/public
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Anodejs
&OptionSettings.member.2.OptionName=ProxyServer
&OptionSettings.member.2.Value=nginx
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Anodejs
&OptionSettings.member.3.OptionName=NodeCommand
&OptionSettings.member.3.Value=""
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Anodejs
&OptionSettings.member.4.OptionName=NodeVersion
&OptionSettings.member.4.Value=0.8.18
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Acontainer%3Anodejs
&OptionSettings.member.5.OptionName=GzipCompression
&OptionSettings.member.5.Value=true
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.6.OptionName=AWS_ACCESS_KEY_ID
&OptionSettings.member.6.Value=AKIAIOSFODNN7EXAMPLE
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.7.OptionName=AWS_SECRET_KEY
&OptionSettings.member.7.Value=wJalrXUtnFEMI%2FK7MDENG%2FbPxRfiCYEXAMPLEKEY
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.8.OptionName=myvar
&OptionSettings.member.8.Value=somevalue
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.9.OptionName=LogPublicationControl
&OptionSettings.member.9.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Deploying a Geddy Application with Clustering to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using EB Command Line Interface (CLI) 3.x and Git, and then updating the application to use the Geddy framework and Amazon ElastiCache for clustering. Clustering enhances your web application's high availability, performance, and security. To learn more about Amazon ElastiCache, go to Introduction to ElastiCache in the *Amazon ElastiCache User Guide*.

Note

This example creates AWS resources, which you may be charged for. For more information about AWS pricing, go to http://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you may test drive these services for free. Go to http://aws.amazon.com/free/ for more information.

Step 1: Set Up Your Git Repository

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Step 2: Set Up Your Geddy Development Environment

Set up Geddy and create the project structure. The following steps walk you through setting up Geddy on a Linux operating system.

To set up your Geddy development environment on your local computer

1. Install Node.js. For instructions, go to http://nodejs.org/. Verify you have a successful installation before proceeding to the next step.

   ```
   $ node -v
   ```

   Note
   For information about what Node.js versions are supported, see Supported Platforms (p. 24).

2. Create a directory for your Geddy application.

   ```
   $ mkdir node-geddy
   $ cd node-geddy
   ```

3. Install npm.

   ```
   node-geddy# cd . && yum install npm
   ```

4. Install Geddy globally so that you have geddy generators or start the server.

   ```
   node-geddy# npm install -g geddy
   ```

5. Depending on your operating system, you may need to set your path to run the `geddy`code> command. If you need to set your path, use the output from the previous step when you installed Geddy. The following is an example.

   ```
   node-geddy# export:PATH=$PATH:/usr/local/share/npm/bin/geddy
   ```

6. Create the directory for your application.

```
node-geddy# geddy app myapp
node-geddy# cd myapp
```

7. Start the server. Verify everything is working, and then stop the server.

   ```
   myapp# geddy
   myapp# curl localhost:4000 (or use web browser)
   ```

   Press **Ctrl+C** to stop the server.
8. Initialize the Git repository.

   ```
   myapp# git init
   ```

9. Exclude the following files from being added to the repository. This step is not required, but it is recommended.

   ```
   myapp# cat > .gitignore <<EOT
   log/
   .gitignore
   .elasticbeanstalk/
   EOT
   ```

Step 3: Configure Elastic Beanstalk

The following instructions use the Elastic Beanstalk command line interface (p. 384) (EB CLI) to configure an Elastic Beanstalk application repository in your local project directory.

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application to use, type the number corresponding to the option **Create new Application**. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use geddyapp.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "myapp"): geddyapp
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.
4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.

5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```

6. Create your running environment.

   ```
   eb create
   ```

7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   > **Note**
   > If you have a space in your application name, make sure you do not have a space in your environment name.

8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

   ```
   Enter DNS CNAME prefix
   (default is HelloWorld):
   ```

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key.

Step 5: View the Application

To view the application

- To open your application in a browser window, type the following:

  ```
  eb open
  ```

Step 6: Update the Application

After you have deployed a sample application, you can update it with your own application. In this step, we update the sample application to use the Geddy framework. You can download the final source code from http://elasticbeanstalk-samples-us-east-1.s3.amazonaws.com/nodejs-example-geddy.zip.

To update your application to use Geddy

1. On your local computer, create a file called `node-geddy/myapp/package.json`. This file contains the necessary dependencies.

   ```
   {
       "name": "Elastic_Beanstalk_Geddy",
       "version": "0.0.1",
       "dependencies": {
           "geddy": "0.6.x"
       }
   }
   ```

2. On your local computer, create a file called `node-geddy/maypp/app.js` as an entry point to the program.

   ```
   var geddy = require('geddy');

   geddy.startCluster({
       hostname: '0.0.0.0',
       port: process.env.PORT || '3000',
       environment: process.env.NODE_ENV || 'development'
   });
   ```

   The preceding snippet uses an environment variable for the environment setting. You can manually set the environment to production (`environment: 'production'`), or you can create an environment variable and use it like in the above example. We'll create an environment variable and set the environment to production in the next procedure.

3. Test locally.

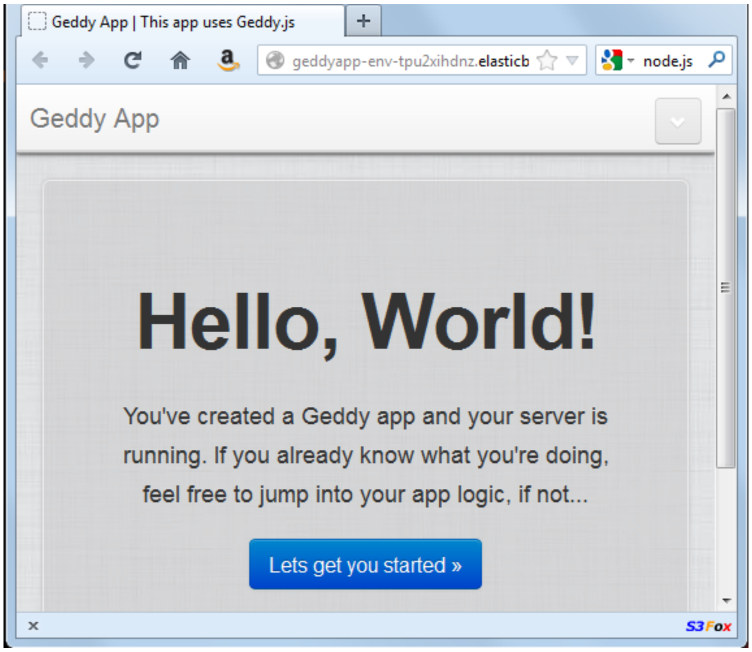
   ```
   myapp# npm install
   myapp# node app
   ```

   The server should start. Press **Ctrl+C** to stop the server.

4. Deploy to Elastic Beanstalk.

   ```
   myapp# git add .
   myapp# git commit -m "First Geddy app"
   myapp# eb deploy
   ```

5. Your environment will be updated after a few minutes. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says "Hello, World!".


Geddy App | This app uses Geddy.js
geddyapp-env-tpu2xihdnz.elasticb
node.js
Geddy App
S3Fox

Hello, World!

You've created a Geddy app and your server is running. If you already know what you're doing, feel free to jump into your app logic, if not...

Lets get you started »

You can access the logs for your EC2 instances running your application. For instructions on accessing your logs, see Instance Logs (p. 282).

Next, let's create an environment variable and set the environment to production.

To create an environment variable

1. On your local computer in your project directory (e.g., `myapp/`), create a directory called `.ebextensions`.
2. On your local computer, create a file called `node-geddy/myapp/.ebextensions/myapp.config` with the following snippet to set the environment to production.

   Note

   Note
   Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

```
option_settings:
  - option_name: NODE_ENV
    value: production
```

   For more information about the configuration file, see Customizing and Configuring a Node.js Environment (p. 684)

3. Run "geddy secret" to get the secret value. You'll need the secret value to successfully deploy your application.

```
myapp# geddy secret
```

You can add node-geddy/myapp/config/secrets.json to .gitignore, or you can put the secret value in an environment variable and create a command to write out the contents. For this example, we'll use a command.

4. Add the secret value from node-geddy/myapp/config/secrets.json to the node-geddy/myapp/.elasticbeanstalk/optionsettings.gettyapp-env file. (The name of the optionsettings file contains the same extension as your environment name). Your file should look similar to the following:

```
[aws:elasticbeanstalk:application:environment]
secret=your geddy secret
PARAM1=
```

5. Update your Elastic Beanstalk environment with your updated option settings.

```
myapp# eb update
```

   Verify that your environment is green and ready before proceeding to the next step.

6. On your local computer, create a configuration file node-geddy/myapp/.ebextensions/write-secret.config with the following command.

```
container_commands:
  01write:
    command: |
      cat > ./config/secrets.json << SEC_END
        { "secret":   "`{ "Fn::GetOptionSetting": {"OptionName": "secret",
 "Namespace":"aws:elasticbeanstalk:application:environment"}}`" }
      SEC_END
```

7. Add your files to the local repository, commit your changes, and deploy your updated application.

```
myapp# git add .
myapp# git commit -m "added config files"
myapp# eb deploy
```

   Your environment will be updated after a few minutes. After your environment is green and ready, refresh your browser to make sure it worked. You should still see "Hello, World!".

Next, let's update the Geddy application to use Amazon ElastiCache.

To updated your Geddy application to use Amazon ElastiCache

1. On your local computer, create a configuration file node-geddy/myapp/.ebextensions/elasticache-iam-with-script.config with the following snippet. This configuration file adds the elasticache resource to the environment and creates a listing of the nodes in the elasticache on disk at /var/nodelist. You can also copy the file from http://elasticbeanstalk-samples-us-east-1.s3.amazonaws.com/nodejs-example-geddy.zip. For more information on the ElastiCache properties, see Example Snippets: ElastiCache (p. 130). For a more complete reference, see AWS Resource Types Reference (p. 800).

```
Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
         Fn::GetOptionSetting:
             OptionName : CacheNodeType
             DefaultValue: cache.m1.small
      NumCacheNodes:
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
      Engine:
           Fn::GetOptionSetting:
             OptionName : Engine
             DefaultValue: memcached
      CacheSecurityGroupNames:
        - Ref: MyCacheSecurityGroup
  MyCacheSecurityGroup:
    Type: AWS::ElastiCache::SecurityGroup
    Properties:
      Description: "Lock cache down to webserver access only"
  MyCacheSecurityGroupIngress:
    Type: AWS::ElastiCache::SecurityGroupIngress
    Properties:
      CacheSecurityGroupName:
        Ref: MyCacheSecurityGroup
      EC2SecurityGroupName:
        Ref: AWSEBSecurityGroup
  AWSEBAutoScalingGroup :
    Metadata :
      ElastiCacheConfig :
        CacheName :
          Ref : MyElastiCache
        CacheSize :
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
  WebServerUser :
    Type : AWS::IAM::User
    Properties :
      Path : "/"
      Policies:
        -
          PolicyName: root
          PolicyDocument :
            Statement :
              -
                Effect : Allow
                Action :
                  - cloudformation:DescribeStackResource
                  - cloudformation:ListStackResources
                  - elasticache:DescribeCacheClusters
                Resource : "*"
  WebServerKeys :
    Type : AWS::IAM::AccessKey
    Properties :
      UserName :
        Ref: WebServerUser

Outputs:
  WebsiteURL:
    Description: sample output only here to show inline string function
parsing
    Value: |
      http://`{ "Fn::GetAtt" : [ "AWSEBLoadBalancer", "DNSName" ] }`
  MyElastiCacheName:
    Description: Name of the elasticache
    Value:
      Ref : MyElastiCache
  NumCacheNodes:
    Description: Number of cache nodes in MyElastiCache
    Value:
      Fn::GetOptionSetting:
        OptionName : NumCacheNodes
        DefaultValue: 1

files:
  "/etc/cfn/cfn-credentials" :
    content : |
      AWSAccessKeyId=`{ "Ref" : "WebServerKeys" }`
      AWSSecretKey=`{ "Fn::GetAtt" : ["WebServerKeys", "SecretAccessKey"]
}`
    mode : "000400"
    owner : root
    group : root

  "/etc/cfn/get-cache-nodes" :
    content : |
      # Define environment variables for command line tools
      export AWS_ELASTICACHE_HOME="/home/ec2-user/elasticache/$(ls /home/ec2-
user/elasticache/)"
      export AWS_CLOUDFORMATION_HOME=/opt/aws/apitools/cfn
      export PATH=$AWS_CLOUDFORMATION_HOME/bin:$AWS_ELASTICACHE_HOME/bin:$PATH

      export AWS_CREDENTIAL_FILE=/etc/cfn/cfn-credentials
      export JAVA_HOME=/usr/lib/jvm/jre

      # Grab the Cache node names and configure the PHP page
      cfn-list-stack-resources `{ "Ref" : "AWS::StackName" }` --region `{
"Ref" : "AWS::Region" }` | grep MyElastiCache | awk '{print $3}' | xargs -
I {} elasticache-describe-cache-clusters {} --region `{ "Ref" : "AWS::Region"
 }` --show-cache-node-info | grep CACHENODE | awk '{print $4 ":" $6}' > `{
 "Fn::GetOptionSetting" : { "OptionName" : "NodeListPath", "DefaultValue"
: "/var/www/html/nodelist" } }`
    mode : "000500"
    owner : root
    group : root

  "/etc/cfn/hooks.d/cfn-cache-change.conf" :
    "content": |
      [cfn-cache-size-change]
      triggers=post.update
      path=Resources.AWSEBAutoScalingGroup.Metadata.ElastiCacheConfig
      action=/etc/cfn/get-cache-nodes
      runas=root

sources :
  "/home/ec2-user/elasticache" : "https://s3.amazonaws.com/elasticache-
```

```
downloads/AmazonElastiCacheCli-latest.zip"

commands:
  make-elasticache-executable:
    command: chmod -R ugo+x /home/ec2-user/elasticache/*/bin/*

packages :
  "yum" :
    "aws-apitools-cfn"  : []

container_commands:
  initial_cache_nodes:
    command: /etc/cfn/get-cache-nodes
```

2. On your local computer, create a configuration file `node-geddy/myapp/.ebextensions/elasticache_settings.config` with the following snippet.

```
option_settings:
  "aws:elasticbeanstalk:customoption" :
     CacheNodeType : cache.m1.small
     NumCacheNodes : 1
     Engine : memcached
     NodeListPath : /var/nodelist
```

3. On your local computer, update `node-geddy/myapp/config/production.js`. Add the following line to the top of the file (just below the header).

```
var fs = require('fs')
```

   Then, add the following snippet just above `modules.exports`.

```
var data = fs.readFileSync('/var/nodelist', 'UTF8', function(err) {
    if (err) throw err;
});

var nodeList = [];
if (data) {
    var lines = data.split('\n');
    for (var i = 0 ; i < lines.length ; i++) {
        if (lines[i].length > 0) {
            nodeList.push(lines[i]);
        }
    }
}

if (nodeList) {
    config.sessions = {
      store: 'memcache',
      servers: nodeList,
      key: 'sid',
      expiry: 14*24*60*60
    }
}
```

4. On your local computer, update `node-geddy/myapp/package.json` to include `memcached`.

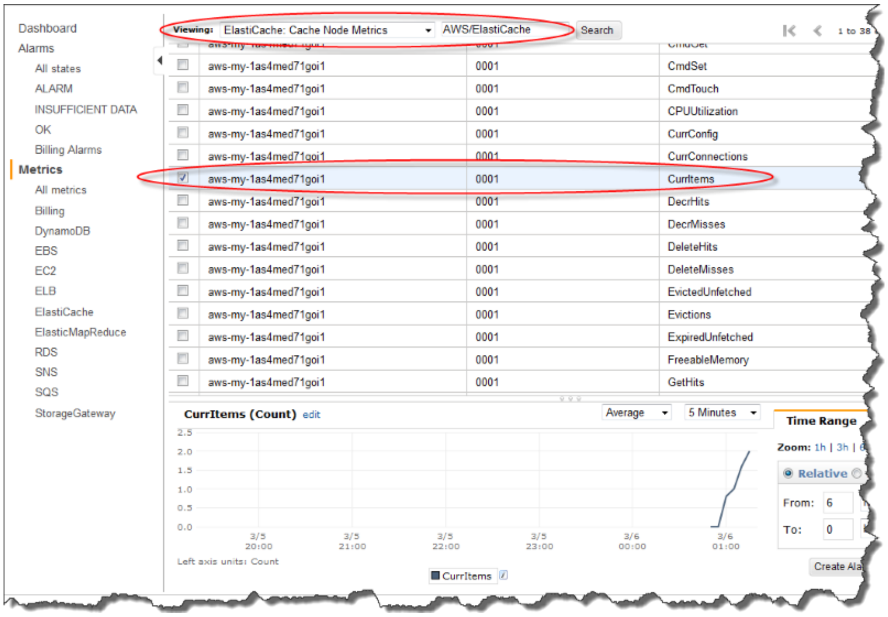
```
{
   "name": "Elastic_Beanstalk_Geddy",
   "version": "0.0.1",
   "dependencies": {
      "geddy": "0.6.x",
      "memcached": "*"
   }
}
```

5. Add your files to the local repository, commit your changes, and deploy your updated application.

```
myapp# git add .
myapp# git commit -m "added elasticache functionality"
myapp# git aws.push
```

6. Your environment will be updated after a few minutes. After your environment is green and ready, verify everything worked.

   a. Check the Amazon CloudWatch console to view your ElastiCache metrics. To view your ElastiCache metrics, click **ElastiCache** in the left pane, and then select **ElastiCache: Cache Node Metrics** from the **Viewing** list.



Viewing: ElastiCache: Cache Node Metrics AWS/ElastiCache

□	aws-my-1as4med71goi1	0001	CmdSet
□	aws-my-1as4med71goi1	0001	CmdTouch
□	aws-my-1as4med71goi1	0001	CPUUtilization
□	aws-my-1as4med71goi1	0001	CurrConfig
□	aws-my-1as4med71goi1	0001	CurrConnections
☑	aws-my-1as4med71goi1	0001	CurrItems
□	aws-my-1as4med71goi1	0001	DecrHits
□	aws-my-1as4med71goi1	0001	DecrMisses
□	aws-my-1as4med71goi1	0001	DeleteHits
□	aws-my-1as4med71goi1	0001	DeleteMisses
□	aws-my-1as4med71goi1	0001	EvictedUnfetched
□	aws-my-1as4med71goi1	0001	Evictions
□	aws-my-1as4med71goi1	0001	ExpiredUnfetched
□	aws-my-1as4med71goi1	0001	FreeableMemory
□	aws-my-1as4med71goi1	0001	GetHits

CurrItems (Count) edit — Average — 5 Minutes

From: 6
To: 0

Note

Make sure you are looking at the same region that you deployed your application to.

If you copy and paste your application URL into another web browser, you should see your CurrItem count go up to 2 after 5 minutes.

   b. Take a snapshot of your logs, and look in `/var/log/nodejs/nodejs.log`. For more information about logs, see Instance Logs (p. 282). You should see something similar to the following:

```
"sessions": {
    "key": "sid",
    "expiry": 1209600,
    "store": "memcache",
    "servers": [
      "aws-my-1awjsrz10lnxo.ypsz3t.0001.usw2.cache.amazonaws.com:11211"

    ]
  },
```

Step 7: Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `terminate` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

- From the directory where you created your local repository, type the following command:

  ```
  eb terminate
  ```

  This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

  > **Note**
  > If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Deploying a Node.js Application to Elastic Beanstalk Using the Elastic Beanstalk Console

If you prefer to use a graphical user interface to deploy your Node.js application, you can use the Elastic Beanstalk console. When using the console, you need to do the following:

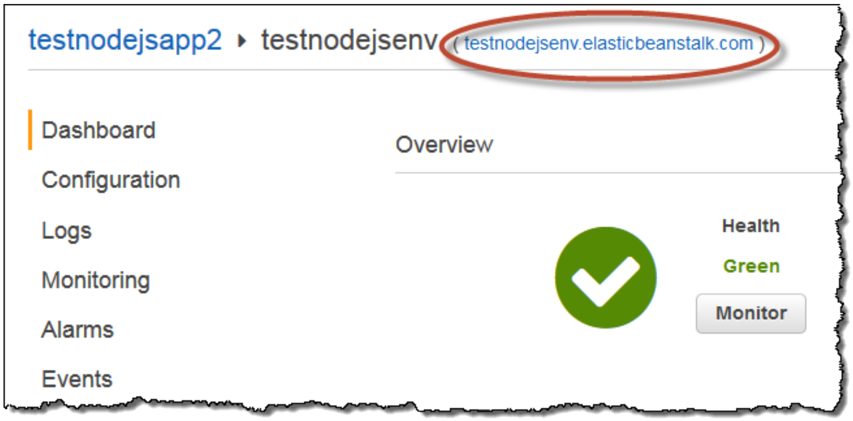
1. Create a .zip file containing your application files.
2. Upload your .zip file to Elastic Beanstalk.

To deploy your Node.js application using the Elastic Beanstalk console

1. Create a `package.json` file and place it in the top-level directory of your source bundle. The following is an example package.json file for the Express framework.

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app"
  },
  "dependencies": {
    "express": "3.1.0",
    "jade": "*",
    "mysql": "*",
    "async": "*",
    "node-uuid": "*"
  }
}
```

2. Create a .zip file containing your application files. By default, Elastic Beanstalk looks for your application in top-level directory of your source bundle.
3. Using the Elastic Beanstalk console, create a new application and upload your .zip file. For instructions, see Create an Application (p. 34).
4. Once your environment is green and ready, click the URL link on the environment dashboard to view your application.



Using Amazon RDS with Node.js

With Amazon Relational Database Service (Amazon RDS), you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. This topic explains how you can use Amazon RDS and Node.js with your Elastic Beanstalk application. For more information about Amazon RDS, go to http://aws.amazon.com/rds/.

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. Install a Node.js driver. For information about drivers, go to https://npmjs.org/.
3. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
4. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with Node.js

This topic walks you through creating a new Amazon RDS DB Instance and using it with your Node.js application.

To use a new Amazon RDS DB Instance and Node.js from your Elastic Beanstalk application

1. Create an Amazon RDS DB Instance. You can create an RDS DB Instance in one of the following ways:

   - Create an Amazon RDS DB instance when you create a new application version. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of an Express application deployment with Amazon RDS using eb, see Deploying an Express Application to Elastic Beanstalk (p. 671).
   - Create an Amazon RDS DB instance when you launch a new environment with an existing application version. For instructions using the AWS Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
   - If you already deployed an application to Elastic Beanstalk, you can create an Amazon RDS DB instance and attach it to an existing environment. For instructions using the AWS Elastic Beanstalk console, see Configuring Databases with Elastic Beanstalk (p. 195). If you use the EB CLI, use the `--database` option when you run `eb create`.

2. Install the driver you want to use to make your database connection. For more information, go to https://npmjs.org/.
3. Establish a database connection in your code using your Amazon RDS DB Instance's connectivity information. You can access your connectivity information using environment variables. The following shows how you would connect to the database on an RDS instance.

   Example 1. Example using node-mysql to connect to an RDS database

   ```
   var mysql = require('mysql');

   var connection = mysql.createConnection({
     host     : process.env.RDS_HOSTNAME,
     user     : process.env.RDS_USERNAME,
     password : process.env.RDS_PASSWORD,
     port     : process.env.RDS_PORT
   });
   ```

   For more information about constructing a connection string using node-mysql, go to https://npmjs.org/package/mysql.

4. Deploy the updated application to your Elastic Beanstalk environment.

Using an Existing Amazon RDS DB Instance with Node.js

You can update your Node.js application to use an Amazon RDS DB Instance that you have previously created. This topic walks you through how to update your Node.js application using an existing Amazon RDS DB Instance and deploy your application to Elastic Beanstalk.

To use an existing Amazon RDS DB Instance and Node.js from your Elastic Beanstalk application

1. Create a new Elastic Beanstalk environment in one of the following ways:

    - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For instructions using eb, see Develop, Test, and Deploy (p. 667). You do not need to create an RDS DB Instance with this environment because you already have an existing RDS DB Instance.
    - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.

2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Install the driver you want to use to make your database connection. For more information, go to https://npmjs.org/.
4. Establish a database connection in your code using your Amazon RDS DB instance's connectivity information. The following examples show how you could connect to the database on an RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

    Example 1. Example using node-mysql to connect to an RDS database

    ```
    var mysql = require('mysql');

    var connection = mysql.createConnection({
      host     : 'mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com',
      user     : 'sa',
      password : 'mypassword',
      port     : '3306'
    });
    ```

    For more information about constructing a connection string using node-mysql, go to https://npmjs.org/package/mysql.

5. Deploy the updated application to your Elastic Beanstalk environment.

Tools

AWS SDK for Node.js

With the AWS SDK for Node.js, you can get started in minutes with a single, downloadable package complete with the AWS Node.js library, code samples, and documentation. You can build Node.js applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in Node.js for how to use the libraries to build applications. Online video tutorials and reference documentation are provided to help you learn how to use the libraries and code samples. For more information about the AWS SDK for Node.js, go to http://aws.amazon.com/sdkfornodejs/.

Git Deployment Via Eb

Eb is an updated command line interface for AWS Elastic Beanstalk that enables you to deploy applications quickly and more easily. To learn how to get started deploying a Node.js application to AWS Elastic Beanstalk using eb and Git, see Develop, Test, and Deploy (p. 667).

Resources

There are several places you can go to get additional help when developing your Node.js applications:

Resource	Description
GitHub	Install the AWS SDK for Node.js using GitHub.
Node.js Development Forum	Post your questions and get feedback.
AWS SDK for Node.js (Developer Preview)	One-stop shop for sample code, documentation, tools, and additional resources.

Deploying Elastic Beanstalk Applications in PHP

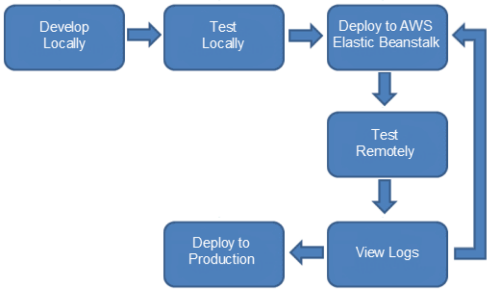
Topics

Elastic Beanstalk for PHP makes it easy to deploy, manage, and scale your PHP web applications using Amazon Web Services. Elastic Beanstalk for PHP is available to anyone developing or hosting a web application using PHP. This section provides instructions for deploying your PHP web application to Elastic Beanstalk. You can deploy your application in just a few minutes using EB Command Line Interface (CLI) 3.x and Git or by using the Elastic Beanstalk management console. It also provides walkthroughs for common frameworks such as CakePHP and Symfony2. For instructions on managing your application and environments using the console, CLIs, or APIs, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

> **Note**
> This section discusses deploying applications using a non-legacy PHP container. If you are running an application using a legacy PHP container, then we recommend that you migrate to a non-legacy PHP container. For instructions on how to check if you are running a legacy container and to migrate to a non-legacy container, see Migrating Your Application from a Legacy Container Type (p. 91). If you require instructions to deploy an application using a legacy PHP container, see Getting Started with Eb (p. 445). (You can use eb 2.6.x to deploy an application using a legacy container, but not EB CLI 3.x.)

Develop, Test, and Deploy

The following diagram illustrates a typical software development life cycle including deploying your application to Elastic Beanstalk.



Typically, after developing and testing your application locally, you will deploy your application to Elastic Beanstalk. At this point, your application will be live at a URL such as http://myexampleapp-wpams3yrvj.elasticbeanstalk.com. Because your application will be live, you should consider setting up multiple environments, such as a testing environment and a production environment. You can point your domain name to the Amazon Route 53 (a highly available and scalable Domain Name System (DNS) web service) CNAME <*yourappname*>.elasticbeanstalk.com. Contact your DNS provider to set this up. For information about how to map your root domain to your Elastic Load Balancer, see Using Elastic Beanstalk with Amazon Route 53 to Map Your Domain to Your Load Balancer (p. 295). After you remotely test and debug your Elastic Beanstalk application, you can then make any updates and redeploy to Elastic Beanstalk. After you are satisfied with all of your changes, you can upload your latest version to your production environment. The following sections provide more details explaining each stage of the software development life cycle.

This section walks you through the steps to deploy a PHP application to Elastic Beanstalk using eb. If you want instructions on how to deploy a PHP application using the Elastic Beanstalk console, see Deploying Elastic Beanstalk Applications in PHP Using the Elastic Beanstalk Console (p. 727).

Get Set Up

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Develop Locally

After installing EB CLI on your local computer, you use the Git command line as you normally would to create your local repository and add and commit changes. You create your PHP application as you normally would with your favorite editor. If you don't already have a PHP application ready, you can use a simple "Hello World" application. Type the following program into your favorite editor, and save it as a PHP file.

```
<html>
 <head>
```

```
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

Next, create a new local repository, add your new program, and commit your change.

```
git add index.php
git commit -m "initial check-in"
```

Note
For information about Git commands, go to Git - Fast Version Control System.

Test Locally

Normally, at this point you would test your application locally before deploying to Elastic Beanstalk. Suppose you find a few issues you would like to fix. Using the above "Hello World" application, add a "!" after "Hello World" and check in your changes. Update your index.php file, and then type the following commands to check in your updated file.

```
git add index.php
git commit -m "my second check-in"
```

After you commit your changes, you should see a response similar to the following:

```
[master 0535814] my second check-in
1 files changed, 1 insertions(+), 1 deletions(-)
```

Note the commit ID that is generated. This ID is used to generate a version label for your application.

Deploy to AWS Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk. Deploying requires the following steps:

- Configure Elastic Beanstalk.
- Deploy a sample application.
- Update the sample application with your application.

When you update the sample application with your application, Elastic Beanstalk replaces the existing sample application version with your new application version in the existing environment.

Use the `init` command, and Elastic Beanstalk will prompt you to enter this information. If a default value is available, and you want to use it, press **Enter**.

If you get a "command not found" error when you run `eb --version`, ensure that the directory that contains the `eb` executable is in your PATH environment variable

Linux, OS X, or Unix

```
$ export PATH=/eb/executable/directory:$PATH
```

Windows

```
> set PATH=%PATH%;C:\eb\executable\directory
```

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application to use, type the number corresponding to the option **Create new Application**. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use HelloWorld.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "windows"): HelloWorld
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.
4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.
5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```

6. Create your running environment.

   ```
   eb create
   ```

7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   Note
   If you have a space in your application name, make sure you do not have a space in your environment name.
8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

```
Enter DNS CNAME prefix
(default is HelloWorld):
```

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key.

To update the sample application with your local application

1. Type the following command.

   ```
   eb deploy
   ```

2. If everything worked as expected, you should see something similar to the following:

   ```
   Counting objects: 5, done.
   Delta compression using up to 4 threads.
   Compressing objects:100% (2/2), done.
   Writing objects: 100% (3/3), 298 bytes, done.
   Total 3 (delta 1), reused 0 (delta 0)
   To https://<some long string>@git.elasticbeanstalk.us-west-
   2.amazon.com/helloworld/helloworldenv
       44c7066..b1f11a1    master -> master
   ```

3. Verify that your application has been updated by refreshing your web browser.

   Note
   The running version is updated and begins with the commit ID from your last commit.

Debug/View Logs

You can use the `eb logs` command to investigate any issues. If you do not specify any flags with the command, logs will be displayed in the command window. For other ways to retrieve logs using this command, see logs (p. 428).

Edit the Application and Redeploy

Now that you have tested your application, it is easy to edit your application, redeploy, and see the results in moments. First, make changes to your application and commit your changes. Then deploy a new application version to your existing Elastic Beanstalk environment.

```
git add index.php
git commit -m "my third check-in"
eb deploy
```

A new application version will be uploaded to your Elastic Beanstalk environment.

You can use the AWS Management Console, CLI, or APIs to manage your Elastic Beanstalk environment. For more information, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

You can also configure Git to push from a specific branch to a specific environment. For more information, see Deploying a Git Branch to a Specific Environment (p. 451).

Deploy to Production

When you are satisfied with all of the changes you want to make to your application, you can deploy it to your production environment. To deploy your application to a new environment, do the following:

1. Commit your changes
2. Create a branch
3. Create and launch your new environment
4. Deploy your application to your new production environment

When you update your application using EB CLI, Elastic Beanstalk will create a new application version. For information on how to deploy an already existing application version to a new environment, see Launching a New AWS Elastic Beanstalk Environment (p. 55). The following steps walk you through committing your new changes and then updating your environment with a new application version using EB CLI and Git.

To deploy to production using EB CLI

1. Commit your changes.

   ```
   git add .
   git commit -m "final checkin"
   ```

2. Create a branch and switch to it.

   ```
   git checkout -b prodenv
   eb use prod
   ```

3. When prompted, type your new environment name, and accept all settings from your previous environment.
4. When you are ready, deploy your new application version to Elastic Beanstalk.

   ```
   eb deploy
   ```

Deploy an Existing Application Version to an Existing Environment

If you need to deploy an existing application to an existing environment, you can do so using the AWS Management Console, CLI, or APIs. You may want to do this if, for instance, you need to roll back to a previous application version. For instructions on how to deploy an existing application version to an existing environment, see Deploying Applications to AWS Elastic Beanstalk Environments (p. 78).

Configuring PHP Containers with Elastic Beanstalk

You can fine-tune the behavior of your Amazon EC2 instances by using a configuration file to configure your container settings. For instructions on customizing and configuring a PHP container, see Customizing and Configuring a PHP Environment (p. 712). For a list of container options, see PHP Container Options (p. 125).

The PHP container settings also lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the AWS Management Console.

Customizing and Configuring a PHP Environment

When deploying your PHP application, you may want to customize and configure the behavior of your EC2 instances. You can easily customize your instances at the same time that you deploy your application version by including a configuration file with your source bundle. This section walks you through the process of creating a configuration file and bundling it with your source.

> **Note**
> Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

> **Note**
> This section does not apply to PHP containers. If you are running an application using a legacy PHP container, then we recommend that you migrate to a non-legacy PHP container. For instructions on how to check if you are running a legacy container and to migrate to a non-legacy container, see Migrating Your Application from a Legacy Container Type (p. 91). If you require instructions for managing your environment for a legacy container using the Elastic Beanstalk console, CLI, or API, see Managing Environments (p. 50).

To customize and configure your PHP environment

1. Create a configuration file with the extension **.config** (e.g., myapp.config) and place it in an **.ebextensions** top-level directory of your source bundle. You can have multiple configuration files in your **.ebextensions** directory. For information about the file format and contents of the configuration file, see Using Configuration Files (p. 99).

   The following is an example snippet of a configuration file.

```
# If you do not specify a namespace, the default used is aws:elasticbean
stalk:application:environment
option_settings:
  - option_name: PARAM1
    value: somevalue
  - option_name: PARAM2
    value: somevalue2
```

   > **Note**
   > You can specify any key-value pairs in the `aws:elasticbeanstalk:application:environment` namespace, and they will be passed in as environment variables on your EC2 instances.

2. Deploy your application version.

For an example walkthrough of deploying a Symfony2 application, see Deploying a Symfony2 Application to Elastic Beanstalk (p. 719). For an example walkthrough of deploying a CakePHP application, see Deploying a CakePHP Application to Elastic Beanstalk (p. 723).

Accessing Environment Configuration Settings

Inside the PHP environment running in Elastic Beanstalk, these values are written to /etc/php.d/environment.ini and are accessible using $_SERVER.

> **Note**
> The `get_cfg_var` function is also supported.

You might have a code snippet that looks similar to the following to access the keys and parameters:

```
echo $_SERVER['PARAM1'];
echo $_SERVER['PARAM2'];
…
echo $_SERVER['PARAM5'];
```

> **Note**
> Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

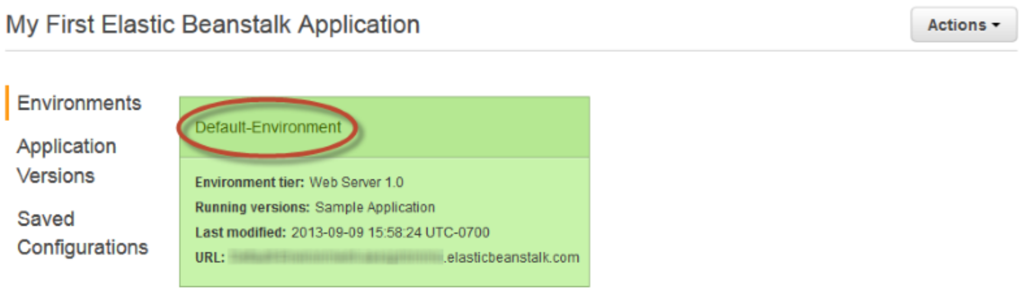
For a list of configuration settings, see PHP Container Options (p. 125).

AWS Management Console

The PHP container settings panel lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Container** tab using the AWS Management Console.

To access the PHP container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



My First Elastic Beanstalk Application

Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

Container Options

On the configuration page, specify the following:

- **Document root**–Lets you specify the child directory of your project that is treated as the public-facing web root. If your root document is stored in your project directory, leave this set to /. If your root document is inside a child directory (e.g., <*project*>/public), set this value to match the child directory. Values should begin with a "/" character, and may *not* begin with a "." character. (This value is written to the http-vhosts.conf file.)
- **Memory limit**–Specifies the amount of memory allocated to the PHP environment. (This value is written to the `php.ini` file.)
- **Allow URL fopen**–Specifies whether the PHP's file functions are allowed to retrieve data from remote locations, such as websites or FTP servers. (This value is written to the php.ini file.)
- **Display errors**–Specifies whether error messages should be part of the output. (This value is written to the `php.ini` file.)
- **Max execution time**–Sets the maximum time, in seconds, a script is allowed to run before the environment terminates it. This helps prevent poorly written scripts from tying up the server.

Log Options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3**–Specifies whether log files for your application's Amazon EC2 instances should be copied to your Amazon S3 bucket associated with your application.

Environment Properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Accessing Environment Configuration Settings

Inside the PHP environment running in Elastic Beanstalk, these values are written to /etc/php.d/environment.ini and are accessible using $_SERVER.

Note
The `get_cfg_var` function is also supported.

You might have a code snippet that looks similar to the following to access the keys and parameters:

```
echo $_SERVER['PARAM1'];
echo $_SERVER['PARAM2'];
…
echo $_SERVER['PARAM5'];
```

Note
Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Command Line Interface (CLI)

To edit an application's environment configuration settings

- Update an application's environment configuration settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "document_root",
        "Value": "/"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "memory_limit",
        "Value": "128M"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "zlib.output_compression",
        "Value": "false"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "allow_url_fopen",
        "Value": "true"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "display_errors",
        "Value": "Off"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "max_execution_time",
        "Value": "60"
    },
    {
        "Namespace": "aws:elasticbeanstalk:container:php:phpini",
        "OptionName": "composer_options",
        "Value": "vendor/package"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_ACCESS_KEY_ID",
        "Value": "AKIAIOSFODNN7EXAMPLE"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "AWS_SECRET_KEY",
        "Value": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    },
    {
        "Namespace": "aws:elasticbeanstalk:application:environment",
        "OptionName": "myvar",
        "Value": "somevalue"
    },
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To edit an application's environment configuration settings

- Call `UpdateEnvironment` with the following parameters:
  - *`EnvironmentName`* = `SampleAppEnv`
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.1.OptionName`* = `document_root`
  - *`OptionSettings.member.1.Value`* = `/`
  - *`OptionSettings.member.2.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.2.OptionName`* = `memory_limit`
  - *`OptionSettings.member.2.Value`* = `128M`
  - *`OptionSettings.member.3.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.3.OptionName`* = `zlib.output_compression`
  - *`OptionSettings.member.3.Value`* = `false`
  - *`OptionSettings.member.4.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.4.OptionName`* = `allow_url_fopen`
  - *`OptionSettings.member.4.Value`* = `true`
  - *`OptionSettings.member.5.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.5.OptionName`* = `display_errors`
  - *`OptionSettings.member.5.Value`* = `Off`
  - *`OptionSettings.member.6.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.6.OptionName`* = `max_execution_time`
  - *`OptionSettings.member.6.Value`* = `60`
  - *`OptionSettings.member.7.Namespace`* = `aws:elasticbeanstalk:container:php:phpini`
  - *`OptionSettings.member.7.OptionName`* = `composer_options`
  - *`OptionSettings.member.7.Value`* = `vendor/package`
  - *`OptionSettings.member.8.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.8.OptionName`* = `AWS_ACCESS_KEY_ID`
  - *`OptionSettings.member.8.Value`* = `AKIAIOSFODNN7EXAMPLE`
  - *`OptionSettings.member.9.Namespace`* = `aws:elasticbeanstalk:application:environment`
  - *`OptionSettings.member.9.OptionName`* = `AWS_SECRET_KEY`
  - *`OptionSettings.member.9.Value`* = `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

- *OptionSettings.member.10.Namespace* = `aws:elasticbeanstalk:application:environment`
- *OptionSettings.member.10.OptionName* = `myvar`
- *OptionSettings.member.10.Value* = `somevalue`
- *OptionSettings.member.11.Namespace* = `aws:elasticbeanstalk:hostmanager`
- *OptionSettings.member.11.OptionName* = `LogPublicationControl`
- *OptionSettings.member.11.Value* = `false`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.1.OptionName=document_root
&OptionSettings.member.1.Value=/
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.2.OptionName=memory_limit
&OptionSettings.member.2.Value=128M
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.3.OptionName=zlib.output_compression
&OptionSettings.member.3.Value=false
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.4.OptionName=allow_url_fopen
&OptionSettings.member.4.Value=true
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.5.OptionName=display_errors
&OptionSettings.member.5.Value=Off
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.6.OptionName=max_execution_time
&OptionSettings.member.6.Value=60
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Acontain
er%3Aphp%3Aphpini
&OptionSettings.member.7.OptionName=composer_options
&OptionSettings.member.7.Value=vendor/package
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.8.OptionName=AWS_ACCESS_KEY_ID
&OptionSettings.member.8.Value=AKIAIOSFODNN7EXAMPLE
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.9.OptionName=AWS_SECRET_KEY
&OptionSettings.member.9.Value=wJalrXUtnFEMI%2FK7MDENG%2FbPxRfiCYEXAMPLEKEY
&OptionSettings.member.10.Namespace=aws%3Aelasticbeanstalk%3Aapplication%3Aen
vironment
&OptionSettings.member.10.OptionName=myvar
&OptionSettings.member.10.Value=somevalue
&OptionSettings.member.11.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.11.OptionName=LogPublicationControl
&OptionSettings.member.11.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Deploying a Symfony2 Application to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using EB Command Line Interface (CLI) 3.x and Git, and then updating the application to use the Symfony2 framework.

Step 1: Set Up Your Git Repository

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Step 2: Set Up Your Symfony2 Development Environment

Set up Symfony2 and create the project structure. The following walks you through setting up Symfony2 on a Linux operating system. For more information, go to http://symfony.com/download.

To set up your PHP development environment on your local computer

1. Download and install composer from getcomposer.org. For more information, go to http://getcomposer.org/download/.

   ```
   curl -s https://getcomposer.org/installer | php
   ```

2. Install Symfony2 Standard Edition with Composer. Check http://symfony.com/download for the latest available version. Using the following command, composer will install the vendor libraries for you.

   ```
   php composer.phar create-project symfony/framework-standard-edition sym
   fony2_example/ <version number>
   cd symfony2_example
   ```

   Note
   You may need to set the date.timezone in the php.ini to successfully complete installation. Also provide parameters for Composer, as needed.

3. Initialize the Git repository.

   ```
   git init
   ```

4. Update the **.gitignore** file to ignore vendor, cache, logs, and composer.phar. These files do not need to get pushed to the remote server.

```
cat > .gitignore <<EOT
app/bootstrap.php.cache
app/cache/*
app/logs/*
vendor
composer.phar
EOT
```

5. Generate the hello bundle.

```
php app/console generate:bundle --namespace=Acme/HelloBundle --format=yml
```

   When prompted, accept all defaults. For more information, go to Creating Pages in Symfony2.

Next, configure Composer. Composer dependencies require that you set the HOME or COMPOSER_HOME environment variable. Also configure Composer to self-update so that you always use the latest version.

To configure Composer

1. Create a configuration file with the extension **.config** (e.g., `composer.config`) and place it in an `.ebextensions` directory at the top level of your source bundle. You can have multiple configuration files in your `.ebextensions` directory. For information about the file format of configuration files, see Using Configuration Files (p. 99).

   Note
   Configuration files should conform to YAML or JSON formatting standards. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively.

2. In the **.config** file, type the following.

```
commands:
  01updateComposer:
    command: export COMPOSER_HOME=/root && /usr/bin/composer.phar self-update

option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
    option_name: COMPOSER_HOME
    value: /root
```

Step 3: Configure Elastic Beanstalk

The following instructions use the Elastic Beanstalk command line interface (p. 384) (EB CLI) to configure an Elastic Beanstalk application repository in your local project directory.

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

```
eb init
```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application name, type the name of the application. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use symfony2app.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "windows"): symfony2app
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.
4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.
5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```
6. Create your running environment.

   ```
   eb create
   ```
7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   Note
   If you have a space in your application name, make sure you do not have a space in your environment name.
8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

   ```
   Enter DNS CNAME prefix
   (default is HelloWorld):
   ```

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key.

To deploy a sample application

- From the directory where you created your local repository, type the following command:

```
eb deploy
```

This process may take several minutes to complete. Elastic Beanstalk will provide status updates during the process. If at any time you want to stop polling for status updates, press **Ctrl+C**. Once the environment status is Green, Elastic Beanstalk will output a URL for the application. You can copy and paste the URL into your web browser to view the application.

Step 4: View the Application

To view the application

- To open your application in a browser window, type the following:

```
eb open
```

Step 5: Update the Application

After you have deployed a sample application, you can update it with your own application. In this step, we update the sample application with a simple "Hello World" Symfony2 application.

To update the sample application

1. Add your files to your local Git repository, and then commit your change.

```
git add -A && git commit -m "Initial commit"
```

> **Note**
> For information about Git commands, go to Git - Fast Version Control System.

2. Create an application version matching your local repository and deploy to the Elastic Beanstalk environment if specified.

```
eb deploy
```

   You can also configure Git to push from a specific branch to a specific environment. For more information, see "Using Git with EB CLI" in the topic Using the EB CLI (p. 394).

3. After your environment is Green and Ready, append **/web/hello/AWS** to the URL of your application. The application should write out "Hello AWS!"

You can access the logs for your EC2 instances running your application. For instructions on accessing your logs, see Instance Logs (p. 282).

Step 6: Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `terminate` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

- From the directory where you created your local repository, type the following command:

  ```
  eb terminate
  ```

  This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

  > **Note**
  > If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Deploying a CakePHP Application to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using EB Command Line Interface (CLI) 3.x and Git, and then updating the application to use the CakePHP framework.

> **Note**
> This example uses Amazon RDSs, and you may be charged for its usage. For more information about pricing, go to Amazon Relational Database Service (RDS) Pricing. If you are a new customer, you can make use of the AWS Free Usage Tier. For details, go to AWS Free Usage Tier.

Step 1: Set Up Your Git Repository

EB CLI is a command line interface that you can use with Git to deploy applications quickly and more easily. EB is available as part of the Elastic Beanstalk command line tools package. For instructions to install EB CLI, see Install the EB Command Line Interface (CLI) (p. 385).

Initialize your Git repository. After you run the following command, when you run `eb init`, EB CLI will recognize that your application is set up with Git.

```
git init .
```

Step 2: Set Up Your CakePHP Development Environment

Set up CakePHP and create the project structure. The following steps walk you through setting up CakePHP on a Linux operating system. For more information, go to http://book.cakephp.org/2.0/en/installation.html.

To set up your PHP development environment on your local computer

1. Download the latest release of CakePHP. This can be done using the .zip file available from GitHub.

   ```
   mkdir ~/cakephp_example
   cd ~/cakephp_example
   ```

```
wget https://github.com/cakephp/cakephp/archive/2.2.4.zip
unzip 2.2.4.zip && rm 2.2.4.zip
mv cakephp-2.2.4/* . && rm -rf cakephp-2.2.4
```

2. Initialize the Git repository.

   ```
   git init
   ```

3. Copy the database settings over.

   ```
   cp app/Config/database.php.default app/Config/database.php
   ```

4. CakePHP excludes `app/Config` by default from being committed to a repository. (The `.gitignore` file that comes with CakePHP also excludes `app/tmp` by default from being committed to a repository.) When deploying to Elastic Beanstalk, we need the database to be configured so that it reads database settings from environment variables. So we need to make sure we do not exclude our database settings. Update the `.gitignore` file so that it does not exclude our database settings.

   ```
   cat > .gitignore <<EOT
   /lib/Cake/Console/Templates/skel/tmp/
   /plugins
   /vendors
   /build
   /dist
   .DS_Store
   /tags
   .elasticbeanstalk/
   EOT
   ```

Step 3: Configure Elastic Beanstalk

The following instructions use the Elastic Beanstalk command line interface (p. 384) (EB CLI) to configure an Elastic Beanstalk application repository in your local project directory.

To configure Elastic Beanstalk

1. From the directory where you created your local repository, type the following command:

   ```
   eb init
   ```

2. When you are prompted for the Elastic Beanstalk region, type the number of the region. For information about this product's regions, go to Regions and Endpoints in the Amazon Web Services General Reference. For this example, we'll use **US West (Oregon)**.
3. When you are prompted for the Elastic Beanstalk application name, type the name of the application. Elastic Beanstalk generates an application name based on the current directory name if an application name has not been previously configured. In this example, we use cakephpapp.

   ```
   Enter an AWS Elastic Beanstalk application name (auto-generated value is
   "windows"): cakephpapp
   ```

   Note
   If you have a space in your application name, make sure you do not use quotation marks.

4. Type **y** if Elastic Beanstalk correctly detected the correct platform you are using. Type **n** if not, and then specify the correct platform.
5. When prompted, type **y** if you want to set up Secure Shell (SSH) to connect to your instances. Type **n** if you do not want to set up SSH. In this example, we will type **n**.

   ```
   Do you want to set up SSH for your instances?
   (y/n): n
   ```

6. Create your running environment.

   ```
   eb create
   ```

7. When you are prompted for the Elastic Beanstalk environment name, type the name of the environment. Elastic Beanstalk automatically creates an environment name based on your application name. If you want to accept the default, press **Enter**.

   ```
   Enter Environment Name
   (default is HelloWorld-env):
   ```

   Note
   If you have a space in your application name, make sure you do not have a space in your environment name.

8. When you are prompted to provide a CNAME prefix, type the CNAME prefix you want to use. Elastic Beanstalk automatically creates a CNAME prefix based on the environment name. If you want to accept the default, press **Enter**.

   ```
   Enter DNS CNAME prefix
   (default is HelloWorld):
   ```

After configuring Elastic Beanstalk, you are ready to deploy a sample application.

If you want to update your Elastic Beanstalk configuration, you can use the **`init`** command again. When prompted, you can update your configuration options. If you want to keep any previous settings, press the **Enter** key.

To deploy a sample application

- From the directory where you created your local repository, type the following command:

  ```
  eb deploy
  ```

  This process may take several minutes to complete. Elastic Beanstalk will provide status updates during the process. If at any time you want to stop polling for status updates, press **Ctrl+C**. Once the environment status is Green, Elastic Beanstalk will output a URL for the application. You can copy and paste the URL into your web browser to view the application.

Step 4: View the Application

To view the application

- To open your application in a browser window, type the following:

```
eb open
```

Step 5: Update the Application

After you have deployed a sample application, you can update the sample application with your own application. In this step, we'll update the sample application with a simple "Hello World" CakePHP application.

To update the sample application

1. On your local computer, modify the **app/Config/database.php** to include RDS database settings. You'll need to add a block of code above the `class DATABASE_CONFIG {` line that defines constant values for RDS configurations. You then need to modify the public $default value to use the constants.

```
if (!defined('RDS_HOSTNAME')) {
  define('RDS_HOSTNAME', $_SERVER['RDS_HOSTNAME']);
  define('RDS_USERNAME', $_SERVER['RDS_USERNAME']);
  define('RDS_PASSWORD', $_SERVER['RDS_PASSWORD']);
  define('RDS_DB_NAME', $_SERVER['RDS_DB_NAME']);
}

class DATABASE_CONFIG {

  public $default = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => RDS_HOSTNAME,
    'login' => RDS_USERNAME,
    'password' => RDS_PASSWORD,
    'database' => RDS_DB_NAME,
    'prefix' => '',
   //'encoding' => 'utf8',
  );

  public $test = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'login' => 'user',
    'password' => 'password',
    'database' => 'test_database_name',
    'prefix' => '',
    //'encoding' => 'utf8',
  );
}
```

2. Add your files to your local Git repository, and then commit your change.

```
git add -A && git commit -m "eb config"
```

> **Note**
> For information about Git commands, go to Git - Fast Version Control System.

3. Create an application version matching your local repository and deploy to the Elastic Beanstalk environment if specified.

```
eb deploy
```

   You can also configure Git to push from a specific branch to a specific environment. For more information, see "Using Git with EB CLI" in the topic Using the EB CLI (p. 394).

You can access the logs for your EC2 instances running your application. For instructions on accessing your logs, see Instance Logs (p. 282).

Step 6: Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `terminate` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

- From the directory where you created your local repository, type the following command:

```
eb terminate
```

  This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

> **Note**
> If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Deploying Elastic Beanstalk Applications in PHP Using the Elastic Beanstalk Console

If you prefer to use a graphical user interface to deploy your PHP application, you can use the Elastic Beanstalk console. When using the console, you need to do the following:

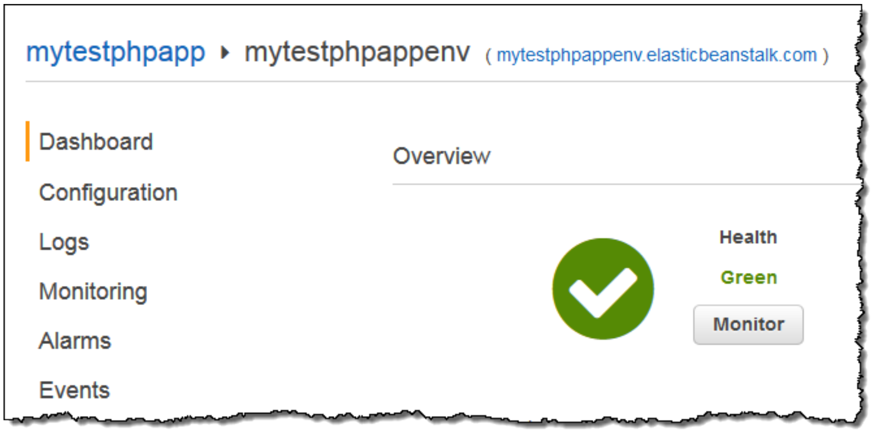
1. Create a .zip file containing your application files.
2. Upload your .zip file to Elastic Beanstalk.
3. Configure the path for your root document.

To deploy your PHP application using the Elastic Beanstalk console

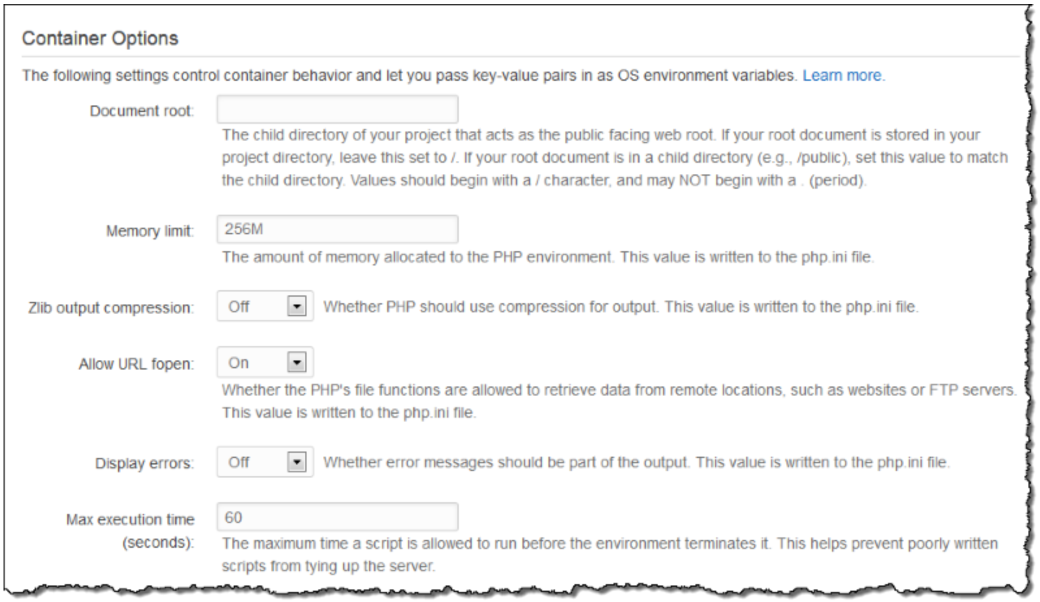
1. Create a .zip file containing your application files. By default, Elastic Beanstalk looks for your root document in top-level directory of your source bundle. If you place your root document in a child directory (e.g., `<yourproject>/public`), you will need to configure the path for the document root. In this example, we'll use the following directory structure for your .zip file.

```
myproject/public/index.php
```

2. Using the Elastic Beanstalk console, create a new application and upload your .zip file. For instructions, see Create an Application (p. 34).
3. Once your environment is green and ready, click the URL link on the environment dashboard to view your application.



4. Update the document root settings to point to the child directory. In the console, click **Configuration** and then click ⚙ for **Software Configuration** in order to edit the container settings. For more information, see Configuring PHP Containers with Elastic Beanstalk (p. 712).



Container Options

The following settings control container behavior and let you pass key-value pairs in as OS environment variables. Learn more.

Document root:
The child directory of your project that acts as the public facing web root. If your root document is stored in your project directory, leave this set to /. If your root document is in a child directory (e.g., /public), set this value to match the child directory. Values should begin with a / character, and may NOT begin with a . (period).

Memory limit: 256M
The amount of memory allocated to the PHP environment. This value is written to the php.ini file.

Zlib output compression: Off
Whether PHP should use compression for output. This value is written to the php.ini file.

Allow URL fopen: On
Whether the PHP's file functions are allowed to retrieve data from remote locations, such as websites or FTP servers. This value is written to the php.ini file.

Display errors: Off
Whether error messages should be part of the output. This value is written to the php.ini file.

Max execution time (seconds): 60
The maximum time a script is allowed to run before the environment terminates it. This helps prevent poorly written scripts from tying up the server.

It will take a few minutes to update your environment.

Using Amazon RDS with PHP

With Amazon Relational Database Service (Amazon RDS), you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. This topic explains how you can use Amazon RDS and PHP with your Elastic Beanstalk application. For more information about Amazon RDS, go to http://aws.amazon.com/rds/.

> **Note**
> The instructions in this topic are for non-legacy container types. If you have deployed an Elastic Beanstalk application using a legacy container type, we recommend that you migrate to a non-legacy container type to gain access to new features. For instructions on how to check the container type and migrate your application, see Migrating Your Application from a Legacy Container Type (p. 91). For instructions on using RDS with applications running on legacy container types, see Using Amazon RDS with PHP (Legacy Container Types) (p. 862).

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. If you plan to use PDO, install the PDO drivers. For more information, go to http://www.php.net/manual/pdo.installation.php.
3. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
4. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with PHP

This topic walks you through creating a new Amazon RDS DB Instance and using it with your PHP application.

To use a new Amazon RDS DB Instance and PHP from your Elastic Beanstalk application

1. Create an Amazon RDS DB Instance. You can create an RDS DB Instance in one of the following ways:

   - Create an Amazon RDS DB instance when you create a new application version. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of a CakePHP application deployment with Amazon RDS using eb, see Deploying a CakePHP Application to Elastic Beanstalk (p. 723).
   - Create an Amazon RDS DB instance when you launch a new environment with an existing application version. For instructions using the AWS Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
   - If you already deployed an application to Elastic Beanstalk, you can create an Amazon RDS DB instance and attach it to an existing environment. For instructions using the AWS Elastic Beanstalk console, see Configuring Databases with Elastic Beanstalk (p. 195). If you use the EB CLI, use the `--database` option when you run `eb create`.

2. If you plan to use PDO, install the PDO drivers. For more information, go to http://www.php.net/manual/pdo.installation.php.

3. Establish a database connection in your code using your Amazon RDS DB Instance's connectivity information. You can access your connectivity information using environment variables. The following shows how you would connect to the database on an RDS instance.

   Example 1. Example using PDO to connect to an RDS database

   ```
   <?php
   $dbhost = $_SERVER['RDS_HOSTNAME'];
   $dbport = $_SERVER['RDS_PORT'];
   $dbname = $_SERVER['RDS_DB_NAME'];

   $dsn = "mysql:host={$dbhost};port={$dbport};dbname={$dbname}";
   $username = $_SERVER['RDS_USERNAME'];
   $password = $_SERVER['RDS_PASSWORD'];

   $dbh = new PDO($dsn, $username, $password);
   ?>
   ```

   For more information about constructing a connection string using PDO, go to http://us2.php.net/manual/en/pdo.construct.php.

   Example 2. Example using mysqli_connect() to connect to an RDS database

   ```
   $link = mysqli_connect($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'],
    $_SERVER['RDS_PASSWORD'], $_SERVER['RDS_DB_NAME'], $_SERVER['RDS_PORT']);
   ```

   For more information on using `mysqli_connect()`, go to http://www.phpbuilder.com/manual/function.mysqli-connect.php.

4. Deploy the updated application to your Elastic Beanstalk environment.

Using an Existing Amazon RDS DB Instance with PHP

You can update your application to use an Amazon RDS DB Instance that you have previously created. This topic walks you through how to update your PHP application using an existing Amazon RDS DB Instance and deploy your application to Elastic Beanstalk.

To use an existing Amazon RDS DB Instance and PHP from your Elastic Beanstalk application

1. Create an Elastic Beanstalk environment in one of the following ways:

   - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of a CakePHP application deployment with Amazon RDS using eb, see Deploying a CakePHP Application to Elastic Beanstalk (p. 723). You do not need to create an RDS DB Instance with this environment because you already have an existing RDS DB Instance.
   - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.

2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. If you plan to use PDO, install the PDO drivers. For more information, go to http://www.php.net/manual/pdo.installation.php.
4. Establish a database connection in your code using your Amazon RDS DB instance's connectivity information. The following examples show how you could connect to the database on an RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

   Example 1. Example using PDO to connect to an RDS database

   ```
   <?php
   $dsn = 'mysql:host=mydbinstance.abcdefghijkl.us-east-1.rds.amazon
   aws.com;port=3306;dbname=mydb';
   $username = 'sa';
   $password = 'mypassword';

   $dbh = new PDO($dsn, $username, $password);
   ?>
   ```

   For more information about constructing a connection string using PDO, go to http://us2.php.net/manual/en/pdo.construct.php.

   Example 2. Example using mysqli_connect() to connect to an RDS database

   ```
   $link = mysqli_connect('mydbinstance.abcdefghijkl.us-east-1.rds.amazon
   aws.com', 'sa', 'mypassword', 'mydb', 3306);
   ```

   For more information on using `mysqli_connect()`, go to http://www.phpbuilder.com/manual/function.mysqli-connect.php.

5. Deploy the updated application to your Elastic Beanstalk environment.

Tools

AWS SDK for PHP

With the AWS SDK for PHP, you can get started in minutes with a single, downloadable package complete with the AWS PHP library, code samples, and documentation. You can build PHP applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides PHP developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in PHP for how to use the libraries to build applications. Online video tutorials and reference documentation are provided to help you learn how to use the libraries and code samples. For more information about the AWS SDK for PHP, go to http://aws.amazon.com/sdkforphp/.

Git Deployment Via Eb

Eb is an updated command line interface for AWS Elastic Beanstalk that enables you to deploy applications quickly and more easily. To learn how to get started deploying a PHP application to AWS Elastic Beanstalk using eb and Git, see Develop, Test, and Deploy (p. 707).

Resources

There are several places you can go to get additional help when developing your PHP applications:

Resource	Description
GitHub	Install the AWS SDK for PHP using GitHub.
PHP Development Forum	Post your questions and get feedback.
PHP Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.
AWS SDK for PHP FAQs	Get answers to commonly asked questions.

Working with Python

This section provides tutorials and information about deploying Python applications using AWS Elastic Beanstalk.

Topics

Installing Python

Depending upon what operating system you use and upon your development needs, you may need to install or upgrade Python on your system. This topic provides information about how to install and configure Python for development and deployment with Amazon Web Services.

The instructions in this topic are presented as a convenience to AWS developers. For complete information about installing Python, see the Python Beginner's Guide.

Do you need Python?

Your operating system may include Python by default. If you are running a Mac OS X, Linux or Unix system this is likely to be the case.

To check for a working Python installation on your system

- open a terminal (command-line) window and type:

```
python --version
```

If the command results in an error, or the version number is less than 2.7, you should install or upgrade your Python installation.

You should also check for the presence of `pip`, a useful package installer for Python that is often installed by default with modern Python installations. Many of the examples in the AWS documentation will use `pip` as an example when installing additional packages or utilities such as the AWS CLI or EB CLI.

To check for pip

- open a terminal window and type:

```
pip --version
```

If pip is not installed, then either:

- install a version of Python that includes `pip` by default, using the instructions in this topic appropriate for your operating system.

or

- see Installing Pip (p. 736) to install `pip` yourself.

Installing Python on Windows

Python is not usually installed on Windows by default; you will likely need to install it yourself.

To install Python on Windows

1. download the latest Python installer for your Windows version (32 or 64-bit) from the Python.org Windows download page.

   Important
   Take care to note where (the volume and directory path) Python was installed. You will need this information for the next step.
2. Install Python on your system using the downloaded Windows installer.
3. Add Python's install path and its `scripts` directory to your user environment:
   a. Open your system Properties dialog (on many versions of Windows, choose **Start** and right-click **Computer**. Then choose **Properties** and click **Change Settings** on the resulting dialog.)
   b. On the **Advanced** tab, choose **Environment Variables**.
   c. Create a new user environment variable by choosing **New...** under **User variables**. Call it `PYTHON_PATH`.
   d. Set the value of `PYTHON_PATH` to the volume and directory where Python was installed. For example: `C:\Python34` if you installed Python 3.4 in the default location.

      Note
      if you ever upgrade your Python installation, you will probably need only to update your `PYTHON_PATH` variable to change to the new installation.
   e. Under **User variables**, edit the `PATH` variable (choose **Edit...**) if it already exists, or create it (choose **New...**) if necessary.
   f. Add the following to your `PATH`:

```
%PYTHON_PATH%;%PYTHON_PATH%\Scripts
```

4. Test your Python installation by opening a new command line window and typing:

```
python --version
```

5. Verify that pip is installed by typing:

```
pip --version
```

If pip is not installed, see Installing Pip (p. 736).

Installing Python on Mac OS X

Mac OS X will typically have Python installed by default, but may not include pip.

To install or upgrade Python on OS X

1. Download the latest Python installer from the Python.org OS X download page. Be sure to pick the version that matches your OS X installation (32 or 64-bit).
2. Install Python using the downloaded OS X installer.
3. Open a Terminal window and test your Python installation by typing:

```
python --version
```

4. Verify that pip is installed by typing:

```
pip --version
```

If pip is not installed, see Installing Pip (p. 736).

Installing Python on Linux or Unix

Python is often installed by default on modern Linux distributions and Unix (BSD) variants, but may not include the -devel packages or pip. Installation instructions will vary depending on your system.

To install Python on RPM (RedHat-based) distributions

1. Open a terminal window and type:

```
sudo yum install python python-devel python-pip
```

2. Verify that python ins installed by typing:

```
python --version
```

3. Verify that pip is installed by typing:

```
pip --version
```

To install Python on APT (Debian-based) distributions

1. Open a terminal window and type:

   ```
   sudo apt-get install python python-dev python-pip
   ```

2. Verify that `python` ins installed by typing:

   ```
   python --version
   ```

3. Verify that `pip` is installed by typing:

   ```
   pip --version
   ```

To install Python on other Linux distributions or on Unix variants

- Use the package manager on your system to search for and install Python and `pip`.

or

- Visit the Python.org download page for "other platforms" to find an appropriate install package and guidance for your platform.

Installing Pip

The `pip` utility is installed by default with Python, beginning with versions 2.7.9 and 3.4.0. If your Python installation doesn't include `pip`, then you can install it yourself. Full instructions are provided on the pip install page.

Installing the AWS Elastic Beanstalk CLI

The *EB CLI 3.x* provides a number of useful commands that you can use to create, configure, and deploy your applications with Elastic Beanstalk. For full documentation, see EB Command Line Interface.

The simplest way to install the CLI is to use `pip`:

```
pip install awsebcli
```

Once installation of the CLI is complete, test your installation by typing:

```
eb --version
```

If successfully installed, EB CLI will report its version number. For example:

```
EB CLI 3.0.10 (Python 2.7.3)
```

Tip
For a full list of available commands provided by EB CLI, type `eb --help` at the command prompt.

Common Steps for Deploying Python Applications

AWS Elastic Beanstalk provides a consistent interface for deploying Python applications, so there are common procedures to follow regardless of the application framework you're using, or whether you're using one at all.

Topics

Common Prerequisites

Important
To use any Amazon Web Service (AWS), including AWS Elastic Beanstalk, you need to have an AWS account and credentials. To learn more and to sign up, visit https://aws.amazon.com/.

For all Python applications that you'll deploy with AWS Elastic Beanstalk, these prerequisites are common:

1. Python 2.7 or 3.x.
2. The `pip` utility, matching your Python version. This is used to install and list dependencies for your project, so that AWS Elastic Beanstalk knows how to set up your application's environment.
3. The `virtualenv` package. This is used to create an environment used to develop and test your application, so that the environment can be replicated by AWS Elastic Beanstalk without installing extra packages that aren't needed by your application.
4. The `awsebcli` package. This is used to initialize your application with the files necessary for deploying with AWS Elastic Beanstalk.
5. A working `ssh` installation. This is used to connect with your running instances when you need to examine or debug a deployment.

Some of these requirements rely upon prior ones (for example: `pip` cannot be installed without Python, and `virtualenv` and `awsebcli` are installed using `pip`); you should install them in the order in which they're listed here.

Installing Python and pip

For information about how to install Python and pip on your system, see *Installing Python* (p. 733).

Installing virtualenv

Once you have Python and `pip` installed, installing or updating virtualenv is simple. Open a command-line window and type:

```
pip install virtualenv
```

Note
On Unix-like systems (Linux or Mac OS X), you may need to prepend `sudo` to the above command if you receive permission errors when trying to install `virtualenv`.

Installing the AWS Elastic Beanstalk CLI

The *EB CLI 3.x* provides a number of useful commands that you can use to create, configure, and deploy your applications with Elastic Beanstalk. For full documentation, see EB Command Line Interface.

The simplest way to install the CLI is to use `pip`:

```
pip install awsebcli
```

Once installation of the CLI is complete, test your installation by typing:

```
eb --version
```

If successfully installed, EB CLI will report its version number. For example:

```
EB CLI 3.0.10 (Python 2.7.3)
```

Tip
For a full list of available commands provided by EB CLI, type `eb --help` at the command prompt.

Installing SSH

You will need a working SSH installation to log into your EC2 instance. If you run Linux or Mac OS X, then SSH is usually pre-installed. To test this, open a command-line window and type:

```
ssh -v
```

If the command doesn't exist, then you'll need to install it using an appropriate means for your operating system. The OpenSSH project is a good starting place to find an implementation for your system.

Tip
Windows users may wish to use a graphical SSH client such as PuTTY.

Setting up a virtual Python environment

Once you have the prerequisites installed, set up a virtual environment with `virtualenv` to install your application's dependencies. By using a virtual environment, you can discern exactly which packages are needed by your application so that the required packages are installed on the EC2 instances that are running your application.

To set up a virtual environment

1. Open a command-line window and type:

   ```
   virtualenv -p python2.7 /tmp/eb_python_app
   ```

   Replace *eb_python_app* with a name that makes sense for your application (using your application's name or directory name is a good idea). The `virtualenv` command creates a virtual environment for you and prints the results of its actions:

   ```
   Running virtualenv with interpreter /usr/bin/python2.7
   New python executable in /tmp/eb_python_app/bin/python2.7
   Also creating executable in /tmp/eb_python_app/bin/python
   Installing setuptools, pip...done.
   ```

2. Once your virtual environment is ready, start it by running the `activate` script located in the environment's `bin` directory. For example, to start the `eb_python_app` environment created in the previous step, you would type:

   ```
   . /tmp/eb_python_app/bin/activate
   ```

   The virtual environment prints its name (for example: `(eb_python_app)`) at the beginning of each command prompt, reminding you that you're in a virtual Python environment.

   Note
   Once created, you can restart the virtual environment at any time by running its `activate` script again.

Configuring a Python project for AWS Elastic Beanstalk

You can use the AWS Elastic Beanstalk CLI to prepare your Python applications for deployment with AWS Elastic Beanstalk.

To configure a Python application for deployment with AWS Elastic Beanstalk

1. From within your virtual environment (p. 738), return to the top of your project's directory tree (`python_eb_app`), and type:

   ```
   pip freeze >requirements.txt
   ```

   This command copies the names and versions of the packages that are installed in your virtual environment to requirements.txt, For example, if the *PyYAML* package, version *3.11* is installed in your virtual environment, the file will contain the line:

   ```
   PyYAML==3.11
   ```

   This allows AWS Elastic Beanstalk to replicate your application's Python environment using the same packages and same versions that you used to develop and test your application.
2. Create an AWS Elastic Beanstalk configuration using the `eb init` command, specifying the AWS region to use. For example:

```
eb init --region us-west-2
```

This command launches a command-line wizard that will prompt you with a number of questions to set up your configuration. Here are some example questions/responses:

Question	Recommended response
Enter Application Name	accept the default (your application directory name) or choose something easy to remember.
It appears you are using Python. Is this correct? (y/n)	y
Select a platform version.	*select your current Python version*
Do you want to set up SSH for your instances? (y/n)	y
Select a keypair.	choose an existing keypair, or create a new one.

If you choose to create a keypair now, `eb init` will prompt you for a keypair name and passphrase (which it will ask you to enter three times). It will then create an SSH keypair on your machine that you can use to access your EC2 instance. Remember your passphrase: you will need this to log into your EC2 instance if your application deployment goes awry, or if you simply want to connect to and examine your instance.

The `eb init` command adds a new directory to your project, named `.elasticbeanstalk`. This directory is hidden on Linux or Mac OS X, but you can see it by typing `ls -a` or `tree -a` on the command-line. For example, running `tree -a` on the files in the example *python_eb_app* application would reveal:

```
python_eb_app
|-- .elasticbeanstalk
|   `-- config.yml
|-- requirements.txt
```

The `config.yml` file within the `.elasticbeanstalk` directory is a YAML-formatted text file that contains the settings that you chose when running `eb init`. For example:

```
branch-defaults:
  default:
    environment: null
global:
  application_name: python_eb_app
  default_ec2_keyname: aws-eb
  default_platform: Python 2.7
  default_region: us-west-2
  profile: null
```

By default, AWS Elastic Beanstalk looks for a file called `application.py` to start your application. If this doesn't exist in the Python project that you've created, some adjustment of your application's environment is necessary. You will also need to set environment variables so that your application's modules can be loaded.

You can use additional configuration files to modify your application environment. These are placed in an `.ebextensions` subdirectory at the top level of your project's directory.

> **Note**
>
> A specific examples of this can be found in the *Deploying a Django Application* (p. 750) tutorial, which configures AWS Elastic Beanstalk to look for Django's `wsgi.py` startup script, instead.

For additional information, see the following topics:

- For more information about setting options in your applications `.ebextensions` directory, see Customizing and Configuring a Python Container.
- For a description of each of the available options, see Option Values, which also includes information about the available Python Container Options.

Deploying your project using AWS Elastic Beanstalk

To create an application environment and deploy your Python application

1. From within your application's top-level directory (for example: `python_eb_app`), deploy you application using the command:

   ```
   eb create
   ```

   This command, like `eb init`, is a configuration wizard that will prompt you for an *environment name* (the default is your project's directory name) and a *DNS CNAME prefix* used to create your final site URL (*dns_cname_prefix*.elasticbeanstalk.com). You can usually accept the default CNAME prefix, but if it has already been used, you will need to enter a unique name.

   `eb create` creates an AWS Elastic Beanstalk application environment, starts the necessary AWS services and launches an EC2 instance to run your application. There will be a significant amount of output, and the process may take more than a few minutes to complete. When finished, the output will end with:

   ```
   INFO: Successfully launched environment: python_eb_app
   ```

2. You can test your deployment using the EB CLI:

   ```
   eb open
   ```

   This command opens a browser window using the URL (CNAME) created for your application.

If you don't see your application running, or get an error message, see Troubleshooting deployments (p. 742) for help with how to determine the cause of the error.

Stopping and deleting your environment

To terminate a running environment, use the `eb terminate` command. When run without any switches, it will terminate the specified application environment. For example:

```
eb terminate python_eb_app
```

You'll be asked to confirm the environment termination by typing its name again. This will terminate the application environment, but will not remove the application itself from AWS Elastic Beanstalk.

To terminate everything associated with your application, use the `--all` switch with `eb terminate`:

```
eb terminate --all python_eb_app
```

This will completely clean up your environment.

Troubleshooting deployments

If your AWS Elastic Beanstalk deployment didn't go quite as smoothly as planned, you may get a 404 (if your application failed to launch) or 500 (if your application fails during runtime) response, instead of seeing your website. To troubleshoot many common issues, you can use the EB CLI to check the status of your deployment, view its logs, gain access to your EC2 instance with SSH, or to open the AWS Management Console page for your application environment.

> **Tip**
> For a complete guide to resolving common deployment problems, see the main Troubleshooting topic, which provides a list of common issues and workarounds.

To use the EB CLI to help troubleshoot your deployment

- Run `eb status` to see the status of your current deployment and health of your EC2 hosts. For example:

  ```
  eb status --verbose

  Environment details for: python_eb_app
    Application name: python_eb_app
    Region: us-west-2
    Deployed Version: app-150206_035343
    Environment ID: e-wa8u6rrmqy
    Platform: 64bit Amazon Linux 2014.09 v1.1.0 running Python 2.7
    Tier: WebServer-Standard-
    CNAME: python_eb_app.elasticbeanstalk.com
    Updated: 2015-02-06 12:00:08.557000+00:00
    Status: Ready
    Health: Green
    Running instances: 1
        i-8000528c: InService
  ```

  > **Note**
  > Using the `--verbose` switch provides information about the status of your running instances. Without it, `eb status` will print only general information about your environment.
- Run `eb logs` to download and view the logs associated with your application deployment. The logs will contain many lines of information, so it's recommended that you send the output to a local file:

  ```
  eb logs >logs.txt
  ```

  You can then use programs such as `grep` or any text editor to view and search through the output.
- Run `eb ssh` to connect with the EC2 instance that's running your application and examine it directly. On the instance, your deployed application can be found in the `/opt/python/current/app` directory, and your Python environment will be found in `/opt/python/run/venv/`.
- Run `eb console` to view your application environment on the AWS Management Console. You can use the web interface to easily examine various aspects of your deployment, including your application's configuration, status, events, logs. You can also download the current or past application versions that you've deployed to the server.

Deploying a Flask Application

This tutorial walks through the deployment of a simple Flask website using AWS Elastic Beanstalk. It shares many steps with those in *Common Steps for Deploying Python Applications* (p. 737). Common steps will often be *referenced* instead of reiterated; this topic will focus on the minimum setup necessary to deploy a Flask-based Python application on AWS Elastic Beanstalk.

> **Note**
> At the time of writing this topic, Flask 0.9 was found to be incompatible with Python 3.4. If you run into issues running Flask with Python 3.4, try using the Python 2.7.x release.

Topics

Prerequisites

> **Important**
> To use any Amazon Web Service (AWS), including AWS Elastic Beanstalk, you need to have an AWS account and credentials. To learn more and to sign up, visit https://aws.amazon.com/.

To follow this tutorial, you should have all of the Common Prerequisites (p. 737) for Python installed. In addition, the Flask framework will be installed as part of the tutorial.

Set up a virtual Python environment

Once you have the prerequisites installed, set up a virtual environment with `virtualenv` to install Flask and its dependencies. By using a virtual environment, you can discern exactly which packages are needed by your application so that the required packages are installed on the EC2 instances that are running your application.

> **Note**
> For an overview of setting up a virtual Python environment, see Setting up a virtual Python environment (p. 737) in *Common Steps for Deploying Python Applications* (p. 737).

To set up your virtual environment

1. Open a command-line window and type:

```
virtualenv -p python2.7 /tmp/eb_flask_app
```

2. Once your virtual environment is ready, start it by typing:

```
. /tmp/eb_flask_app/bin/activate
```

You will see (eb_flask_app) prepended to your command prompt, indicating that you're in a virtual environment.

Install Flask

Now that the virtual environment has been created and is active, you should install Flask in the environment.

Use *pip* to install Flask by typing:

```
pip install Flask
```

To verify that Flask has been installed, type:

```
pip freeze
```

This will list Flask and its dependencies. For example:

```
Flask==0.10.1
itsdangerous==0.24
Jinja2==2.7.3
MarkupSafe==0.23
Werkzeug==0.10.1
```

Create a Flask application

Next, create an application that you'll deploy using AWS Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

To create the Hello World Flask application

1. Create a directory for your project. Use eb_flask_app to match the environment:

   ```
   mkdir eb_flask_app
   ```

2. With your favorite text editor, create a new text file in this directory called application.py. Add the following contents:

   ```
   from flask import Flask

   # print a nice greeting.
   def say_hello(username = "World"):
       return '<p>Hello %s!</p>\n' % username

   # some bits of text for the page.
   header_text = '''
       <html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
   instructions = '''
       <p><em>Hint</em>: This is a RESTful web service! Append a username
       to the URL (for example: <code>/Thelonious</code>) to say hello to
       someone specific.</p>\n'''
   home_link = '<p><a href="/">Back</a></p>\n'
   footer_text = '</body>\n</html>'
   ```

```
# EB looks for an 'application' callable by default.
application = Flask(__name__)

# add a rule for the index page.
application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

# add a rule when the page is accessed with a name appended to the site
# URL.
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# run the app.
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

This example prints a customized greeting that depends on the URL used to access the service.

Note

By adding `application.debug = True` before running the application, debug output is enabled in case something goes wrong. It's a good practice for development, but you should remove debug statements in production code, since debug output can reveal internal aspects of your application.

Using `application.py` as the filename and providing a callable `application` object (the Flask object, in this case) allows AWS Elastic Beanstalk to easily find your application's code.

Run the application locally

You can test the application locally by running it with Python.

To test your application

1. Open a terminal window and navigate to your application directory:

```
cd eb_flask_app
```

2. *Optional*. Start your virtual environment if it isn't already started:

```
. /tmp/eb_flask_app/bin/activate
```

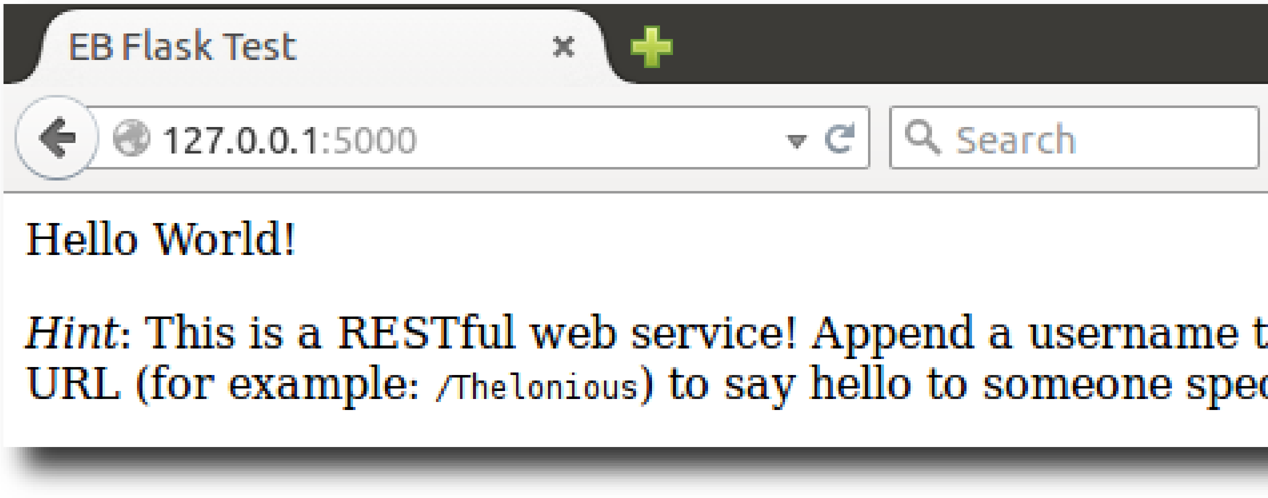
3. Run `application.py` with Python:

```
python application.py
```

Flask will start a web server and display the URL to access your application with. For example:

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
```

4. Open the URL in your Web browser. You should see the application running, showing the index page:



Hello World!

Hint: This is a RESTful web service! Append a username t
URL (for example: `/Thelonious`) to say hello to someone spe

If you got debug output instead, fix the errors and make sure the application is running locally before configuring it for AWS Elastic Beanstalk.

Configure your Flask application for AWS Elastic Beanstalk

With your application running locally, you're now ready to configure it to deploy with AWS Elastic Beanstalk.

To configure your Flask application for AWS Elastic Beanstalk

1. From within your virtual environment, return to the top of your project's directory tree (`eb_flask_app`), and type:

```
pip freeze >requirements.txt
```

   This command copies the names and versions of the packages that are installed in your virtual environment to requirements.txt, which is loaded by AWS Elastic Beanstalk to install the packages needed by your application.

2. Deactivate your virtual environment for now by typing the `deactivate` command:

```
deactivate
```

3. Create an AWS Elastic Beanstalk configuration using the `eb init` command, specifying the AWS region to use. For example:

```
eb init --region us-west-2
```

This command launches a command-line wizard that will prompt you with a number of questions to set up your configuration. You can accept the default values for most of the questions. Here is a short guide to help you fill out your responses:

Question	Recommended response
Select an application to use	Accept default: `[Create new Application]`
Enter Application Name	Accept default: `eb_flask_app`
It appears you are using Python. Is this correct? (y/n)	
Select a platform version	*Select your Python version*

Question	Recommended response
Do you want to set up SSH for your instances? (y/n)	
Select a key pair.	Accept default: *Create new keypair*

The `eb init` command adds a new directory to your project, named `.elasticbeanstalk`. This directory is hidden on Linux or Mac OS X, but you can see it by typing `ls -a` or `tree -a` on the command-line.

Your project directory should now look like this:

```
eb_flask_app
|-- .elasticbeanstalk
|   `-- config.yml
|-- .gitignore
|-- application.py
`-- requirements.txt
```

The `config.yml` file within the `.elasticbeanstalk` directory is a YAML-formatted text file that contains the settings that you chose when running `eb init`. If you followed the recommended responses, it will contain:

```
branch-defaults:
  default:
    environment: null
global:
  application_name: eb_flask_app
  default_ec2_keyname: aws-eb
  default_platform: Python 2.7
  default_region: us-west-2
  profile: null
```

Deploy your site using AWS Elastic Beanstalk

Next, you'll create your application environment and deploy your configured application with AWS Elastic Beanstalk.

To create an application environment and deploy your Flask application

1. From within your application's top-level directory (`eb_flask_app`), deploy you application using the command:

   ```
   eb create
   ```

   This command, like `eb init`, is a configuration wizard that will prompt you for an *environment name* (the default is your project's directory name) and a *DNS CNAME prefix* used to create your final site URL (*dns_cname_prefix*.elasticbeanstalk.com). You can usually accept the default CNAME prefix, but if it has already been used, you will need to enter a unique name.

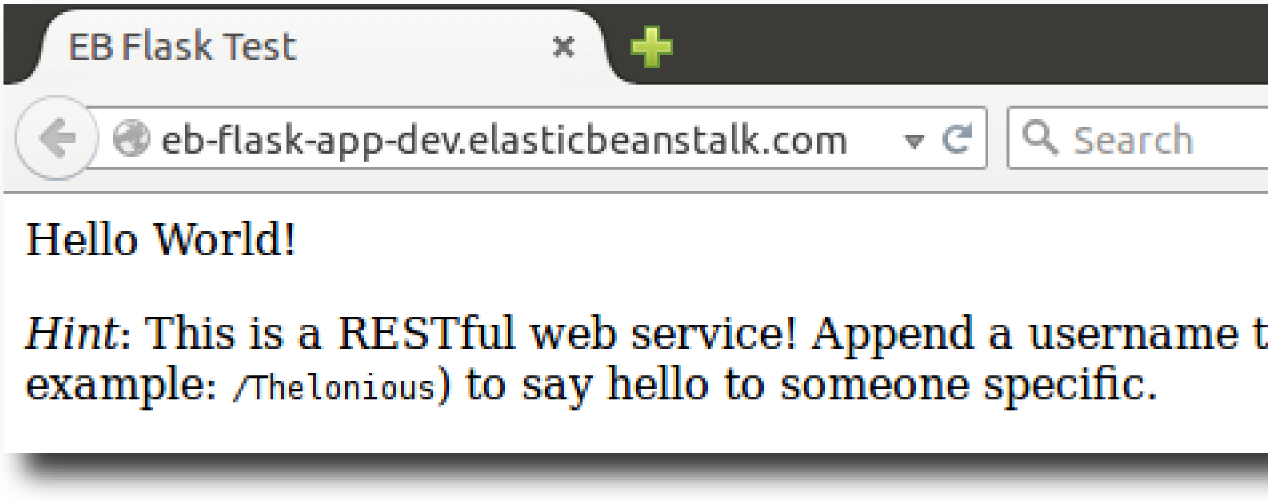
   `eb create` creates an AWS Elastic Beanstalk application environment, starts the necessary AWS services and launches an EC2 instance to run your application. There will be a significant amount of output, and the process may take more than a few minutes to complete. When finished, the output will end with:

   ```
   INFO: Successfully launched environment: eb-flask-app-dev
   ```

2. Test your deployment using the EB CLI:

   ```
   eb open
   ```

   This will open a browser window using the URL (CNAME) created for your application. You should see the same Flask website that you created and tested locally. Now, however, it's running on AWS Elastic Beanstalk!



Hello World!

Hint: This is a RESTful web service! Append a username t
example: `/Thelonious`) to say hello to someone specific.

If you don't see your application running, or get an error message, see Troubleshooting deployments (p. 737) for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Flask application with AWS Elastic Beanstalk!

Stopping and Deleting your Environment

Once you're finished experimenting with your Flask application and with AWS Elastic Beanstalk deployments, you should terminate your application environment to keep your EC2 costs low:

```
eb terminate eb-flask-app-dev
```

The termination process will ask you to confirm the environment name by re-typing it.

Where do I go from here?

- For more information about deploying, updating, terminating and troubleshooting Python applications with AWS Elastic Beanstalk, be sure to review the material in *Common Steps for Deploying Python Applications* (p. 737).
- If you want information about deploying a Django-based site using AWS Elastic Beanstalk, continue to *Deploying a Django Application* (p. 750).

Deploying a Django Application

This tutorial walks through the deployment of a simple Django website using AWS Elastic Beanstalk. It shares many steps with those in *Common Steps for Deploying Python Applications* (p. 737). Common steps will often be *referenced* instead of reiterated; this topic will focus on the minimum setup necessary to deploy a Django-based Python application on AWS Elastic Beanstalk.

Topics

Prerequisites

> **Important**
> To use any Amazon Web Service (AWS), including AWS Elastic Beanstalk, you need to have an AWS account and credentials. To learn more and to sign up, visit https://aws.amazon.com/.

To follow this tutorial, you should have all of the Common Prerequisites (p. 737) for Python installed. In addition, the Django framework will be installed as part of the tutorial.

Set up a virtual Python environment

Once you have the prerequisites installed, set up a virtual environment with `virtualenv` to install Django and its dependencies. By using a virtual environment, you can discern exactly which packages are needed by your application so that the required packages are installed on the EC2 instances that are running your application.

> **Note**
> For an overview of setting up a virtual Python environment, see Setting up a virtual Python environment (p. 737) in *Common Steps for Deploying Python Applications* (p. 737).

To set up your virtual environment

1. Open a command-line window and type:

```
virtualenv -p python2.7 /tmp/eb_django_app
```

2. Once your virtual environment is ready, start it by typing:

```
. /tmp/eb_django_app/bin/activate
```

You will see `(eb_django_app)` prepended to your command prompt, indicating that you're in a virtual environment.

Install Django

Now that the virtual environment has been created and is active, you should install Django in the environment.

Use *pip* to install Django by typing:

```
pip install django
```

To verify that Django has been installed, type:

```
pip freeze
```

This will list Django and its dependencies. For example:

```
Django==1.7.7
```

Create a Django project

Now you are ready to create a Django project and run it on your machine, using the virtual python environment you set up.

To create your Django project

1. Make sure that you are in your virtual Python environment as set up in Set up a virtual Python environment (p. 750). Restart it if necessary.
2. Create a new directory to hold your project files, such as `eb_django_app`, and enter it:

```
mkdir eb_django_app
cd eb_django_app
```

3. From within your `eb_django_app` directory, create a new Django project by typing:

```
django-admin startproject django_eb
```

This will create a Django site using the directory name you specify (here, `django_eb` was used as the directory name). You will now have a project directory structure similar to the following:

```
eb_django_app/
`-- django_eb
    |-- django_eb
    |   |-- __init__.py
    |   |-- settings.py
    |   |-- urls.py
    |   `-- wsgi.py
    `-- manage.py
```

4. Enter the `django_eb` directory you just created:

```
cd django_eb
```

5. Apply migrations to your project to set up your site's applications. Type:

```
python manage.py migrate
```

Run the application locally

You can test your django project locally by running it with Python.

To test your application

1. Open a terminal window and navigate to your application directory:

```
cd eb_django_app/django_eb
```

2. *Optional*. Start your virtual environment if it isn't already started:

```
. /tmp/eb_django_app/bin/activate
```

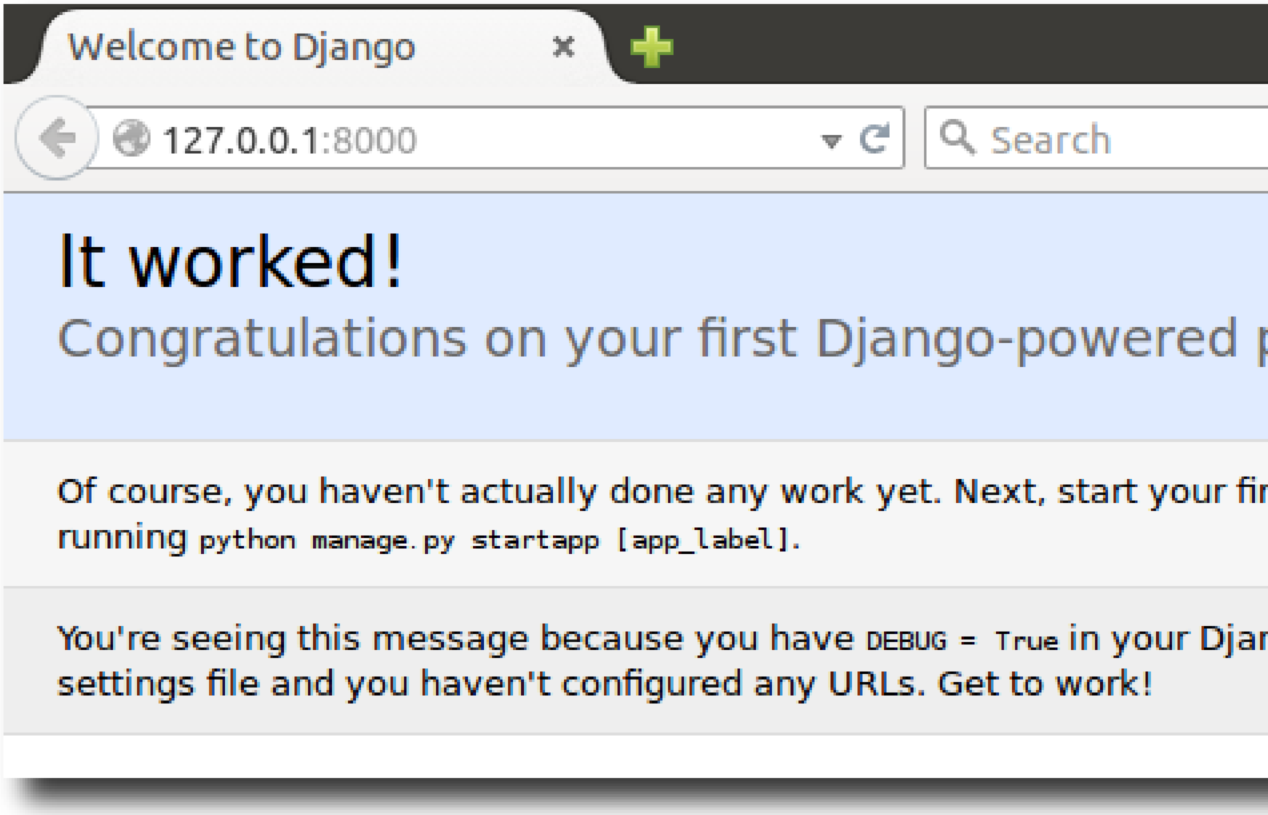
3. Test your site by running `manage.py`. Type:

```
python manage.py runserver
```

The `runserver` argument will start a local instance of the Django server, and will print the local URL that you can use to view the site. For example:

```
Performing system checks...

System check identified no issues (0 silenced).
February 05, 2015 - 09:48:02
Django version 1.7.4, using settings 'django_eb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

4. Copy the URL into your browser (in this example: `http://127.0.0.1:8000/`) to view the site:



It worked!
Congratulations on your first Django-powered p

Of course, you haven't actually done any work yet. Next, start your fir
running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Djan
settings file and you haven't configured any URLs. Get to work!

5. Quit the server by pressing **Ctrl+C**, which will return you to the command prompt. Once you've verified your site locally, there's no need to keep it running.

Configure your Django application for AWS Elastic Beanstalk

Now that you have a Django-powered site that you've created on your local system, you can configure it for deployment with AWS Elastic Beanstalk.

To configure your site for AWS Elastic Beanstalk

1. From within your virtual environment, return to the top of your project's directory tree (`eb_django_app`), and type:

```
pip freeze >requirements.txt
```

   This command copies the names and versions of the packages that are installed in your virtual environment to requirements.txt, which is loaded by AWS Elastic Beanstalk to install the packages needed by your application.
2. Deactivate your virtual environment for now by typing the `deactivate` command:

```
deactivate
```

3. Create an AWS Elastic Beanstalk configuration using the `eb init` command, specifying the AWS region to use. For example:

   ```
   eb init --region us-west-2
   ```

   This command launches a command-line wizard that will prompt you with a number of questions to set up your configuration. You can accept the default values for most of the questions. Here is a short guide to help you fill out your responses:

Question	**Recommended response**
Select an application to use	Accept default: `[Create new Application]`
Enter Application Name	Accept default: `eb_django_app`
It appears you are using Python. Is this correct? (y/n)	
Select a platform version	*Select your Python version*

Question	Recommended response
Do you want to set up SSH for your instances? (y/n)	
Select a keypair.	Accept default: *Create new keypair*

The `eb init` command adds a new directory to your project, named `.elasticbeanstalk`. This directory is hidden on Linux or Mac OS X, but you can see it by typing `ls -a` or `tree -a` on the command-line.

Your project directory should now look like this:

```
eb_django_app
|-- .elasticbeanstalk
|   `-- config.yml
|-- django_eb
|   |-- db.sqlite3
|   |-- django_eb
|   |   |-- __init__.py
|   |   |-- settings.py
|   |   |-- urls.py
|   |   |-- wsgi.py
|   `-- manage.py
`-- requirements.txt
```

The `config.yml` file within the `.elasticbeanstalk` directory is a YAML-formatted text file that contains the settings that you chose when running `eb init`. If you followed the recommended responses, it will contain:

```
branch-defaults:
  default:
    environment: null
global:
  application_name: django_eb
  default_ec2_keyname: aws-eb
  default_platform: Python 2.7
  default_region: us-west-2
  profile: null
```

By default, AWS Elastic Beanstalk looks for a file called `application.py` to start your application. Since this doesn't exist in the Django project that you've created, some adjustment of your application's environment is necessary. You will also need to set environment variables so that your application's modules can be loaded.

You can do this using a separate YAML configuration file that contains overrides and additional settings for your application environment.

To configure your application environment

1. On the command-line, go to the top-level of your project (`eb_django_app`) and create a new directory, called `.ebextensions`:

   ```
   mkdir .ebextensions
   ```

2. Using a text editor, create a new file called `01-django_eb.config` in the `.ebextensions` directory, and add the following lines to it:

   ```
   option_settings:
     "aws:elasticbeanstalk:application:environment":
       DJANGO_SETTINGS_MODULE: "django_eb.settings"
       PYTHONPATH: "/opt/python/current/app/django_eb:$PYTHONPATH"
     "aws:elasticbeanstalk:container:python":
       WSGIPath: "django_eb/django_eb/wsgi.py"
   ```

   These settings will be used in your application environment when it starts. Setting the *PYTHONPATH* allows the environment to find your application's modules, and the *DJANGO_SETTINGS_MODULE* specifies which module to load to configure your Django site.
3. Save the file and close your text editor.

Note
See these topics for additional information:

- For more information about setting options in your applications `.ebextensions` directory, see Customizing and Configuring a Python Container.
- For a description of each of the available options, see Option Values, which also includes information about the available Python Container Options.

Deploy your site using AWS Elastic Beanstalk

You've added everything you need to deploy your application on AWS Elastic Beanstalk. Your project directory should now look like this:

```
eb_django_app/
|-- .ebextensions
|   `-- 01-django_eb.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- django_eb
|   |-- db.sqlite3
|   |-- django_eb
|   |   |-- __init__.py
|   |   |-- settings.py
```

```
|   |   |-- urls.py
|   |   |-- wsgi.py
|   `-- manage.py
`-- requirements.txt
```

Next, you'll create your application environment and deploy your configured application with AWS Elastic Beanstalk.

To create an application environment and deploy your Django application

1. From within your application's top-level directory (`eb_django_app`), deploy you application using the command:

   ```
   eb create
   ```

   This command, like `eb init`, is a configuration wizard that will prompt you for an *environment name* (the default is your project's directory name) and a *DNS CNAME prefix* used to create your final site URL (*dns_cname_prefix*.elasticbeanstalk.com). You can usually accept the default CNAME prefix, but if it has already been used, you will need to enter a unique name.

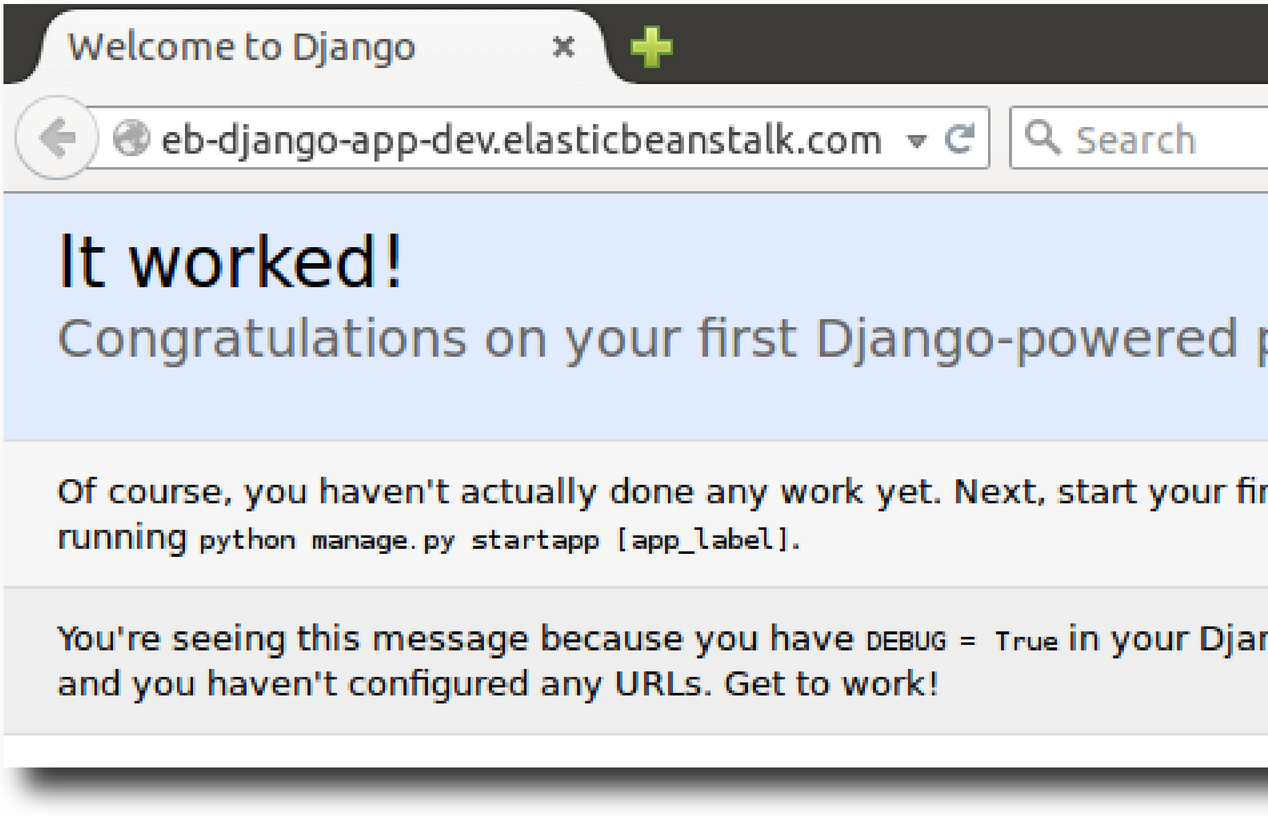
   `eb create` creates an AWS Elastic Beanstalk application environment, starts the necessary AWS services and launches an EC2 instance to run your application. There will be a significant amount of output, and the process may take more than a few minutes to complete. When finished, the output will end with:

   ```
   INFO: Successfully launched environment: eb-django-app-dev
   ```

2. Test your deployment using the EB CLI:

   ```
   eb open
   ```

   This will open a browser window using the URL (CNAME) created for your application. You should see the same Django website that you created and tested locally. Now, however, it's running on AWS Elastic Beanstalk!



It worked!
Congratulations on your first Django-powered

Of course, you haven't actually done any work yet. Next, start your fir
running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Djar
and you haven't configured any URLs. Get to work!

If you don't see your application running, or get an error message, see Troubleshooting deployments (p. 737) for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Django application with AWS Elastic Beanstalk!

Updating your application

Now that you have a running application on AWS Elastic Beanstalk, you can update and redeploy your application or its configuration and AWS Elastic Beanstalk will take care of the work of updating your instances and starting your new application version.

For this example, we'll enable Django's admin console and configure a few other settings.

Modify your site settings

Change the time zone that your site uses. By default, 'UTC' is used.

To change your site's time zone

1. Go to your application's top-level directory (`eb_django_app`).
2. Open your Django site's `settings.py` file using a text editor. You'll find it at `django_eb/django_eb/settings.py`.
3. In `settings.py`, edit the TIME_ZONE property to set the desired time zone. For example:

```
TIME_ZONE = 'US/Pacific'
```

4. Save the file and close your text editor.

Add a site migration to your startup script

You can add commands to your `.ebextensions` script that will be run when your site is updated. This allows you to automatically generate database migrations.

To add a migrate step when your site starts up

1. Go to your application's top-level directory (`eb_django_app`).
2. Open the `.ebextensions/01-django_eb.config` file that you created in Configure your Django application for AWS Elastic Beanstalk (p. 753), and add the following lines to the top of the file:

```
container_commands:
  01_migrate:
    command: "django-admin.py migrate"
    leader_only: true
```

   You can list a number of commands here; they will be run in alphabetic order when starting your instances. Specifying `leader_only: true` in the command ensures that it is run only once when you're deploying to multiple instances.

   > **Note**
   > For more information about using commands in your configuration scripts, see Customizing the Software on EC2 Instances Running Linux.

1. Save the file and close your text editor.

Create a site administrator

1. Enter the `eb_django_app/django_eb` directory, which contains `manage.py`, and set up your site administrator:

```
python manage.py createsuperuser
```

   You'll be asked a few questions, such as your admin's username, email address, and password (you'll be prompted for it twice). For example:

```
Username: admin
Email address: me@mydomain.com
Password: ********
Password (again): ********
Superuser created successfully.
```

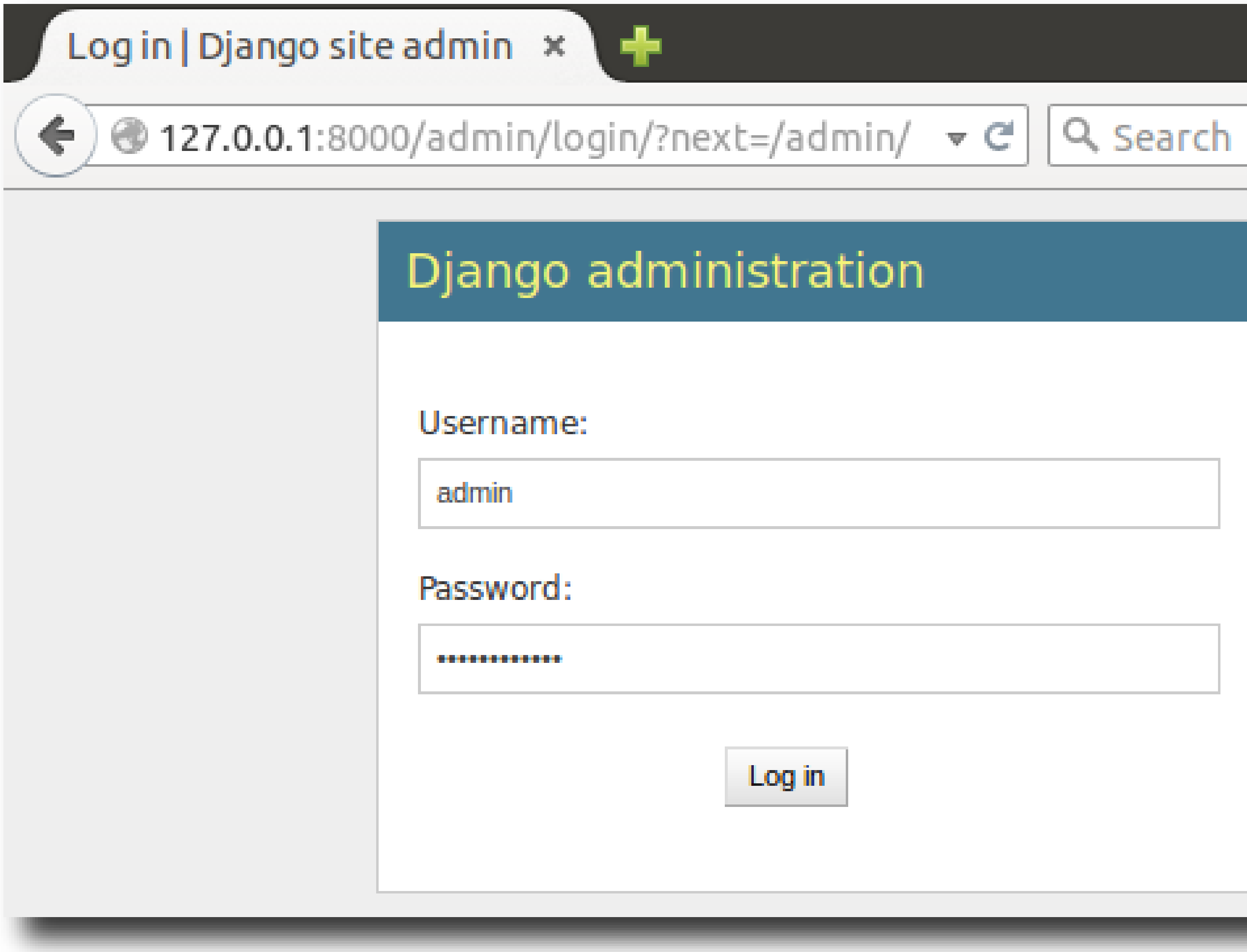
2. Test your changes locally by using Django's `runserver` command. From the `eb_django_app/django_eb` directory, run:

```
python manage.py runserver
```

3. View the admin console by opening the local site in your browser, appending `/admin/` to the site URL, such as:

   ```
   http://127.0.0.1:8000/admin/
   ```

   You should see your admin login screen:

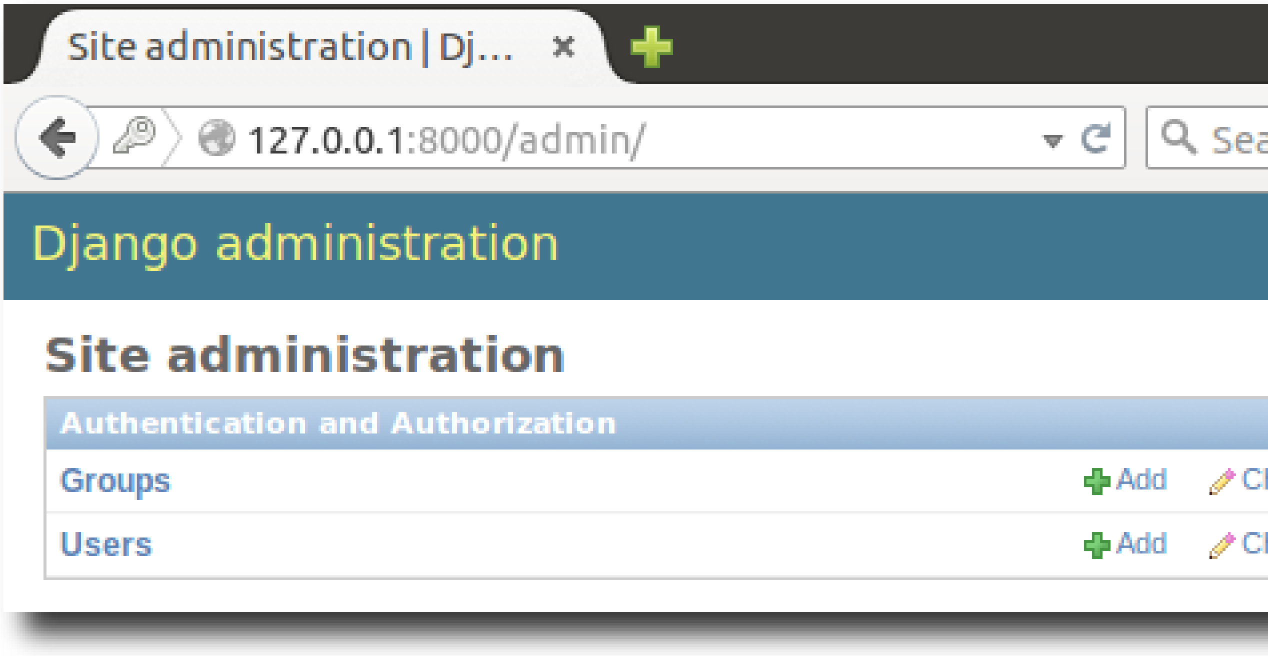


Username: admin

Password: •••••••••••••

Log in | Django site admin
127.0.0.1:8000/admin/login/?next=/admin/
Search
Django administration
Log in

4. Use the Username/Password information you created for the superuser and login to your built-in administration console:



Django administration

Site administration

Authentication and Authorization		
Groups	Add	Cl
Users	Add	Cl

When you're finished viewing your changes, press **Ctrl+C** to stop the server.

5. Return to the top-level directory in your project (`eb_django_app`) and use the EB CLI to re-deploy your application:

```
eb deploy
```

6. Once deployment is complete, view the changes on your AWS Elastic Beanstalk-hosted site:

```
eb open
```

   You can access the admin console by adding `/admin/` to the end of the URL in your browser, such as:

```
http://eb-django-app-dev.elasticbeanstalk.com/admin/
```

You can use a similar procedure of local updating/testing followed by `eb deploy`. AWS Elastic Beanstalk takes care of the work of updating your live servers, so you can focus on application development instead of server administration!

Stopping and Deleting your Environment

Once you're finished experimenting with your Django application and with AWS Elastic Beanstalk deployments, you should terminate your application environment to keep your EC2 costs low:

```
eb terminate eb-django-app-dev
```

The termination process will ask you to confirm the environment name by re-typing it.

Where do I go from here?

- For more information about deploying, updating, terminating and troubleshooting Python applications with AWS Elastic Beanstalk, be sure to review the material in *Common Steps for Deploying Python Applications* (p. 737).
- If you want information about deploying a Flask-based site using AWS Elastic Beanstalk, continue to *Deploying a Flask Application* (p. 743).

Using Amazon RDS with Python

With Amazon Relational Database Service (Amazon RDS), you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. This topic explains how you can use Amazon RDS and Python with your Elastic Beanstalk application. For more information about Amazon RDS, go to http://aws.amazon.com/rds/.

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
3. Update your **requirements.txt** file.
4. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with Python

This topic walks you through creating a new Amazon RDS DB Instance and using it with your Python application.

To use a new Amazon RDS DB Instance and Python from your Elastic Beanstalk application

1. Create an Amazon RDS DB Instance. You can create an RDS DB Instance in one of the following ways:

   - Create an Amazon RDS DB instance when you create a new application version. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of a Django application deployment with Amazon RDS using the EB CLI, see Deploying a Django Application (p. 750).
   - Create an Amazon RDS DB instance when you launch a new environment with an existing application version. For instructions using the AWS Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
   - If you already deployed an application to Elastic Beanstalk, you can create an Amazon RDS DB instance and attach it to an existing environment. For instructions using the AWS Elastic Beanstalk console, see Configuring Databases with Elastic Beanstalk (p. 195). If you use the EB CLI, use the `--database` option when you run `eb create`.

2. Access the RDS connection information in the `os.environ` object.

```
import os

if 'RDS_HOSTNAME' in os.environ:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': os.environ['RDS_DB_NAME'],
            'USER': os.environ['RDS_USERNAME'],
            'PASSWORD': os.environ['RDS_PASSWORD'],
            'HOST': os.environ['RDS_HOSTNAME'],
            'PORT': os.environ['RDS_PORT'],
        }
    }
```

   Note
   This information is only accessible by application code and will not appear in environment variables listed from a terminal.

3. Create a **requirements.txt** file and place it in the top-level directory of your source bundle. A typical python application will have dependencies on other third-party Python packages. In Python, pip is the standard way of installing packages. Pip has a feature that allows you to specify all the packages you need (as well as their versions) in a single requirements file. For more information about the requirements file, go to Requirements File Format. The following is an example requirements.txt file for Django.

```
Django==1.4.1
MySQL-python==1.2.3
```

4. Deploy the updated application to your Elastic Beanstalk environment.

Using an Existing Amazon RDS DB Instance with Python

You can update your Python application to use an Amazon RDS DB Instance that you have previously created. This topic walks you through how to update your Python application using an existing Amazon RDS DB Instance and deploy your application to Elastic Beanstalk.

To use an existing Amazon RDS DB Instance and Python from your Elastic Beanstalk application

1. Create an Elastic Beanstalk environment in one of the following ways:

   - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of an Django application deployment with Amazon RDS using the EB CLI, see Deploying a Django Application (p. 750). You do not need to create an RDS DB Instance with this environment because you already have an existing RDS DB Instance.
   - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.

2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Establish a database connection in your code using your Amazon RDS DB instance's connectivity information. The following examples show how you could connect to the database on an RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': 'mydb',
            'USER': 'sa',
            'PASSWORD': 'mypwd',
           'HOST': 'mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com',

            'PORT': '3306',
        }
    }
```

4. Create a **requirements.txt** file, add the package you need to communicate to the database, and place it in the top-level directory of your source bundle. For more information about the requirements file, go to Requirements File Format. The following is an example requirements.txt file.

```
MySQL-python==1.2.3
```

5. Deploy the updated application to your Elastic Beanstalk environment.

Configuring Python Containers with Elastic Beanstalk

You can fine-tune the behavior of your Amazon EC2 instances using a configuration file to configure your container settings. For instructions on customizing and configuring a Python container, see Customizing and Configuring a Python Container (p. 764). For a list of container options, see Python Container Options (p. 125).

If you want to enable or disable Amazon S3 log rotation, you can use an instance configuration file, or you can use the AWS Management Console, CLI, or the API. The topic explains how to configure this setting using the AWS Management Console, CLI, and the API.

Customizing and Configuring a Python Container

When deploying your Python application, you may want to customize and configure the behavior of your Amazon EC2 instances. You can easily customize your instances at the same time that you deploy your application version by including a configuration file with your source bundle. This section walks you through the process of creating a configuration file and bundling it with your source.

To customize and configure your Python container

1. Create a configuration file with the extension **.config** (e.g., myapp.config) and place it in an **.ebextensions** top-level directory of your source bundle. You can have multiple configuration files in your **.ebextensions** directory. These files are executed in alphabetical order. For example, `.ebextensions/01run.config` is executed before `.ebextensions/2do.config`.

   Note
   Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

   The following is an example snippet of a configuration file.

```
packages:
  yum:
    libmemcached-devel: '0.31'

container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  03wsgipass:
    command: 'echo "WSGIPassAuthorization On" >> ../wsgi.conf'
  99customize:
    command: "scripts/customize.sh"

# You can specify any key-value pairs in the aws:elasticbeanstalk:applica
tion:environment namespace and it will be
# passed in as environment variables on your EC2 instances
option_settings:
  "aws:elasticbeanstalk:application:environment":
    DJANGO_SETTINGS_MODULE: "djproject.settings"
    "application_stage": "staging"
  "aws:elasticbeanstalk:container:python":
    WSGIPath: djproject/wsgi.py
    NumProcesses: 3
    NumThreads: 20
  "aws:elasticbeanstalk:container:python:staticfiles":
    "/static/": "static/"
```

   The following is an example of specifying the **option_settings** in a list format:

```
# If you do not specify a namespace, the default used is aws:elasticbean
stalk:application:environment
option_settings:
  - option_name: PARAM1
    value: somevalue
```

> **Note**
> If you want to set environment variables that will be available to your application, you do not need to provide a "namespace" key in the option_settings section.

2. Create a **requirements.txt** file and place it in the top-level directory of your source bundle. A typical python application will have dependencies on other third-party Python packages. In Python, pip is the standard way of installing packages. Pip has a feature that allows you to specify all the packages you need (as well as their versions) in a single requirements file. For more information about the requirements file, go to Requirements File Format. The following is an example requirements.txt file for Django.

   ```


   Django==1.4.1
   MySQL-python==1.2.3
   ```

   From your working environment, you can also type the following command to generate the requirements file.

   ```
   pip install django
   pip install MySQL-python==1.2.3
   pip freeze > requirements.txt
   ```

3. Deploy your application version.

For an example walkthrough of deploying a Django application using an instance configuration file, see Deploying a Django Application (p. 750). For an example walkthrough of deploying a Flask application, see Deploying a Flask Application (p. 743).

Accessing Environment Variables

Inside the Python environment running in Elastic Beanstalk, these values are accessible using Python's `os.environ` dictionary. For more information, go to http://docs.python.org/library/os.html. For a list of option settings, see Python Container Options (p. 125).

You might have a code snippet that looks similar to the following to access the keys and parameters:

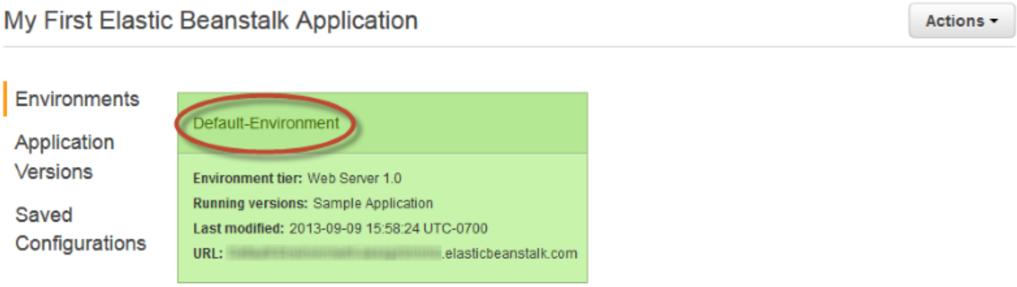
```
import os

param1 = os.environ['PARAM1']
django_settings_module = os.environ['DJANGO_SETTINGS_MODULE']
```

AWS Management Console

The Python container settings lets you enable or disable Amazon S3 log rotation.

To access the Python container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



Default-Environment

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

Log Options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3**–Specifies whether log files for your application's Amazon EC2 instances should be copied to your Amazon S3 bucket associated with your application.

Command Line Interface (CLI)

To edit an application's environment configuration settings

- Update an application's environment configuration settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To edit an application's environment configuration settings

- Call `UpdateEnvironment` with the following parameters:

- *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:hostmanager`
- *`OptionSettings.member.1.OptionName`* = `LogPublicationControl`
- *`OptionSettings.member.1.Value`* = `false`

Example

```
https://elasticbeanstalk.us-east-1.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.1.OptionName=LogPublicationControl
&OptionSettings.member.1.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Deploying a Python Application to Elastic Beanstalk Using the Elastic Beanstalk Console

If you prefer to use a graphical user interface to deploy your Python application, you can use the Elastic Beanstalk console. When using the console, you need to do the following:

1. Create a .zip file containing your application files.
2. Upload your .zip file to Elastic Beanstalk.

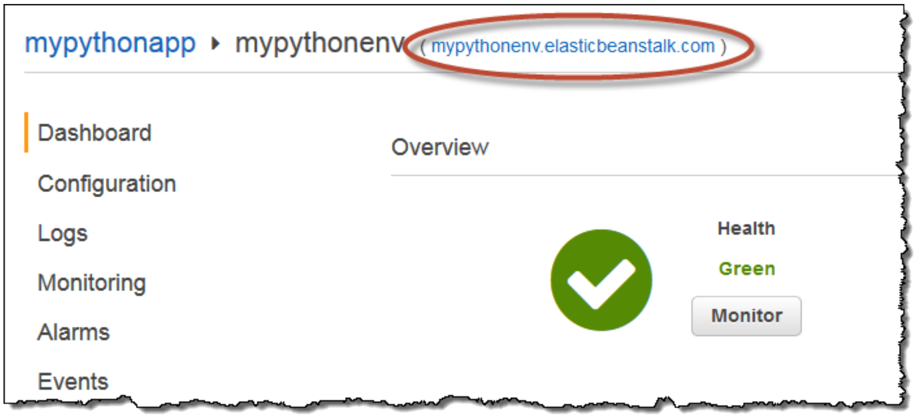
To deploy your python application using the Elastic Beanstalk console

1. Create a `requirements.txt` file and place it in the top-level directory of your source bundle. The following is an example requirements.txt file for Django.

   ```
   Django==1.4.1
   MySQL-python==1.2.3
   ```

   Note
   For more information about the requirements file, go to Requirements File Format.

2. Create a .zip file containing your application files. By default, Elastic Beanstalk looks for your application (application.py) in top-level directory of your source bundle.
3. Using the Elastic Beanstalk console, create a new application and upload your .zip file. For instructions, see Create an Application (p. 34).
4. Once your environment is green and ready, click the URL link on the environment dashboard to view your application.



Python Tools and Resources

This section provides information about additional tools and resources for developing Python applications with Elastic Beanstalk.

Topics

- Tools (p. 769)
- Resources (p. 769)

Tools

Boto (open source AWS SDK for Python)

With Boto, you can get started in minutes with a single, downloadable package complete with the AWS Python library, code samples, and documentation. You can build Python applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides Python developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in Python for how to use the libraries to build applications. For information about Boto, sample code, documentation, tools, and additional resources, go to http://aws.amazon.com/python/.

Git Deployment Via Eb

Eb is an updated command line interface for AWS Elastic Beanstalk that enables you to deploy applications quickly and more easily. To learn how to get started deploying a Python application to Elastic Beanstalk using eb and Git, see Working with Python (p. 733).

Resources

There are several places you can go to get additional help when developing your Python applications:

Resource	Description
Boto (open source AWS SDK for Python)	Install Boto using GitHub.
Python Development Forum	Post your questions and get feedback.

Resource	Description
Python Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Deploying Elastic Beanstalk Applications in Ruby Using EB CLI and Git

Topics

Elastic Beanstalk for Ruby makes it easy to deploy, manage, and scale your Ruby web applications using Amazon Web Services. Elastic Beanstalk is available to anyone developing or hosting a web application using Ruby. This section provides step-by-step instructions for deploying a sample application to Elastic Beanstalk using the Elastic Beanstalk command line interface (EB CLI), and then updating the application to use the Rails and Sinatra web application frameworks. To complete this walkthrough, you will need to install and configure the EB CLI (p. 384).

After you deploy your Elastic Beanstalk application, you can use the AWS Management Console, CLIs, or the APIs to manage your Elastic Beanstalk environment. For more information, see Managing and Configuring Applications and Environments Using the Console, CLI, and APIs (p. 34).

Deploying a Rails Application to Elastic Beanstalk

You can use the EB Command Line Interface (CLI) and Git to deploy a Rails sample application to Elastic Beanstalk. This walkthrough shows you how. You'll also learn how to set up a Rails installation from scratch in case you don't already have a development environment and application.

Software Versions

Many of the technologies presented here are under active development. For the best results, use the same versions of each tool when possible. The versions used during the development of this tutorial are listed below.

Ubuntu	14.04
RVM	1.26.3
Ruby	2.1.5p273
Rails	4.1.8
Python	2.7.6

For the typographic conventions used in this tutorial, see Document Conventions in the General Reference.

Topics

Rails Development Environment Setup

Read this section if you are setting up a Rails development environment from scratch. If you have a development environment configured with Rails, Git and a working app, you can skip this section (p. 773).

Getting an Ubuntu EC2 Instance

The following instructions were developed and tested using an Amazon EC2 instance running Ubuntu 14.04. For instructions on configuring and connecting to an EC2 instance using the AWS Management Console, read the Getting Started section of the *Amazon EC2 User Guide for Linux*.

If you don't have access to the AWS Management Console or prefer to use the command line, check out the AWS CLI User Guide for instructions on installing the AWS CLI and using it to configure security groups, create a key pair, and launch instances with the same credentials that you will use with the EB CLI.

Install Rails

RVM, a popular version manager for Ruby, provides an option to install RVM, Ruby, and Rails with just a few commands:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
$ curl -sSL https://get.rvm.io | bash -s stable --rails
$ source /home/ubuntu/.rvm/scripts/rvm
```

Install nodejs to allow the Rails server to run locally:

```
$ sudo apt-get install nodejs
```

Note
For help installing rails on other operating systems, try http://installrails.com/.

Create a New Rails Project

Use `rails new` with the name of the application to create a new Rails project.

```
$ rails new rails-beanstalk
```

Rails creates a directory with the name specified, generates all of the files needed to run a sample project locally, and then runs bundler to install all of the dependencies (Gems) defined in the project's Gemfile.

Run the Project Locally

Test your Rails installation by running the default project locally.

```
$ cd rails-beanstalk
rails-beanstalk $ rails server -d
=> Booting WEBrick
=> Rails 4.2.0 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
rails-beanstalk $ curl http://localhost:3000
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails: Welcome aboard</title>
...
```

Note
Elastic Beanstalk precompiles Rails assets by default. For Ruby 2.1 container types, note the following:

- The Nginx web server is preconfigured to serve assets from the `/public` and `/public/assets` folders.
- The Puma application requires that you add **`gem "puma"`** to your `Gemfile` for `bundle exec` to run correctly.

Install the EB CLI

In this section you'll install the EB CLI, a few dependencies, and Git.

Note
Using Git or another form of revision control is recommended but also entirely optional when using the EB CLI. Any of the steps in this tutorial that use Git can be skipped.

Install Git, Python Development Libraries and Pip

This tutorial uses Git for revision control and Pip to manage the EB CLI installation. In your Ubuntu development environment, you can install all of them with the following sequence of commands:

```
$ sudo apt-get install git
$ sudo apt-get install python-dev
$ curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
$ sudo python get-pip.py
```

Windows Users
Install Python 3.4, which includes pip.

Install the EB CLI

With Pip you can install the EB CLI with a single command:

Linux, OS X, or Unix

```
$ sudo pip install awsebcli
```

Windows

```
> pip install awsebcli
```

Set Up Your Git Repository

If your Rails project is already in a local Git repository, continue to Configure the EB CLI (p. 774).

First, initiate the repository. From within the Rails project directory, type `git init`.

```
$ git init
Initialized empty Git repository in /home/ubuntu/rails-beanstalk/.git/
```

Next, add all of the project's files to the staging area and commit the change.

```
rails-beanstalk $ git add .
rails-beanstalk $ git commit -m "default rails project"
 56 files changed, 896 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Gemfile
...
```

Configure the EB CLI

With the Git repository configured and all necessary tools installed, configuring the EB CLI project is as simple as running `eb init` from within the project directory and following the prompts.

```
$ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
...
```

The following values work for this tutorial, but feel free to use values that make sense for your requirements. If you don't have access keys, see How Do I Get Security Credentials? in the AWS General Reference.

Eb Init Values

Region	**Enter** (keep default)
AWS Access Key ID	Your access key
AWS Secret Access Key	Your secret key
Application Name	**Enter** (keep default)
Using Ruby?	**y** (yes)
Platform Version	**Enter** (keep default)
Set up SSH?	**n** (no)

In addition to configuring the environment for deployment, `eb init` sets up some Git extensions and adds an entry to the `.gitignore` file in the project directory. Commit the change to `.gitignore` before moving on.

```
rails-beanstalk $ git commit -am "updated .gitignore"
```

Deploy the Project

Next, you'll deploy the default Rails project to an Elastic Beanstalk environment.

```
rails-beanstalk $ eb create rails-beanstalk-env
Creating application version archive "app-150219_215138".
Uploading rails-beanstalk/app-150219_215138.zip to S3. This may take a while.
Upload Complete.
Environment details for: rails-beanstalk-env
  Application name: rails-beanstalk
  Region: us-west-2
  Deployed Version: app-150219_215138
  Environment ID: e-pi3immkys7
  Platform: 64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Puma)
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2015-02-19 21:51:40.686000+00:00
Printing Status:
INFO: createEnvironment is starting.
...
```

With just one command, the EB CLI sets up all of the resources our application needs to run in AWS, including the following:

- An Amazon S3 bucket to store environment data
- A load balancer to distribute traffic to the web server(s)
- A security group to allow incoming web traffic
- An Auto Scaling group to adjust the number of servers in response to load changes
- Amazon CloudWatch alarms that notify the Auto Scaling group when load is low or high
- An Amazon EC2 instance hosting our application

When the process is complete, the EB CLI outputs the public DNS name of the application server. Use **eb open** to open the website in the default browser. In our Ubuntu environment the default browser is a text based browser called W3M.

```
$ eb open
A really lowlevel plumbing error occured. Please contact your local Maytag(tm)
 repair man.
```

This is Puma's way of telling us that something went wrong. When an error like this occurs, you can check out the logs using the `eb logs` command.

```
rails-beanstalk $ eb logs
INFO: requestEnvironmentInfo is starting.
INFO: [Instance: i-8cdc6480] Successfully finished tailing 5 log(s)
================ i-8cdc6480 =================
-------------------------------------
/var/log/eb-version-deployment.log
-------------------------------------
...
```

The error you're looking for is in the web container log, `/var/log/puma/puma.log`.

```
...
-------------------------------------
/var/log/puma/puma.log
-------------------------------------
=== puma startup: 2014-12-15 18:37:51 +0000 ===
=== puma startup: 2014-12-15 18:37:51 +0000 ===
[1982] + Gemfile in context: /var/app/current/Gemfile
[1979] - Worker 0 (pid: 1982) booted, phase: 0
2014-12-15 18:41:42 +0000: Rack app error: #<RuntimeError: Missing
`secret_key_base` for 'production' environment, set this value in `con
fig/secrets.yml`>
/opt/rubies/ruby-2.1.4/lib/ruby/gems/2.1.0/gems/railties-4.1.8/lib/rails/applic
ation.rb:462:in `validate_secret_key_config!'
/opt/rubies/ruby-2.1.4/lib/ruby/gems/2.1.0/gems/railties-4.1.8/lib/rails/applic
ation.rb:195:in `env_config'
...
```

To get the application working, you need to configure a few environment variables. First is SECRET_KEY_BASE, which is referred to by `secrets.yml` in the `config` folder of our project.

This variable is used to create keys and should be a secret, as the name suggests. This is why you don't want it stored in source control where other people might see it. Set this to any value using `eb setenv`:

```
rails-beanstalk $ eb setenv SECRET_KEY_BASE=23098520lkjsdlkjfsdf
INFO: Environment update is starting.
INFO: Updating environment rails-beanstalk-env's configuration settings.
INFO: Successfully deployed new configuration to environment.
INFO: Environment update completed successfully.
```

The EB CLI automatically restarts the web server whenever you update configuration or deploy new code. Try loading the site again.

```
$ eb open
The page you were looking for doesn't exist (404)
```

A 404 error may not look like much of an improvement, but it shows that the web container is working and couldn't find a route to the page you're looking for.

So what happened to the welcome page you saw earlier? In this case the environment variable you need is RACK_ENV. Right now it's set to production, suppressing the display of debug features as well as the Welcome to Rails page.

View the current value of all environment variables using the `eb printenv` command.

```
rails-beanstalk $ eb printenv
 Environment Variables:
     AWS_SECRET_KEY = None
     RAILS_SKIP_ASSET_COMPILATION = false
     SECRET_KEY_BASE = 23098520lkjsdlkjfsdf
     RACK_ENV = production
     PARAM5 = None
     PARAM4 = None
     PARAM3 = None
     PARAM2 = None
     PARAM1 = None
     BUNDLE_WITHOUT = test:development
     RAILS_SKIP_MIGRATIONS = false
     AWS_ACCESS_KEY_ID = None
```

The proper way to fix this is to add content and routes to the project. For the moment, though, we just want to see our project working, so we'll set RACK_ENV to development.

```
rails-beanstalk $ eb setenv RACK_ENV=development
```

The next time you load the site it should succeed.

```
$ eb open
Ruby on Rails: Welcome aboard
...
```

Now that you know it works, you can set RACK_ENV back to production and see about adding that content.

```
rails-beanstalk $ eb setenv RACK_ENV=production
```

Update the Application

Now it's time to add some content to the front page to avoid the 404 error you saw in production mode.

First you'll use `rails generate` to create a controller, route, and view for your welcome page.

```
$ rails generate controller WelcomePage welcome
      create  app/controllers/welcome_page_controller.rb
       route  get 'welcome_page/welcome'
      invoke  erb
      create    app/views/welcome_page
      create    app/views/welcome_page/welcome.html.erb
...
```

This gives you all you need to access the page at *rails-beanstalk-env-kpvmmmqpbr*.elasticbeanstalk.com/welcome_page/welcome. Before you publish the changes, however, change the content in the view and add a route to make this page appear at the top level of the site.

Use your favorite text editor to edit the content in `app/views/welcome_page/welcome.html.erb`. Nano and Vim are popular command line editors. For this example, you'll use `cat` to simply overwrite the content of the existing file.

```
rails-beanstalk $ cat > app/views/welcome_page/welcome.html.erb
> <h1>Welcome!</h1>
> <p>This is the front page of my first Rails application on Elastic Bean
stalk.</p>
Ctrl+D
```

Finally, add the following route to `config/routes.rb`:

```
Rails.application.routes.draw do
  get 'welcome_page/welcome'
  root 'welcome_page#welcome'
end
```

This tells Rails to route requests to the root of the website to the welcome page controller's welcome method, which renders the content in the welcome view (`welcome.html.erb`). Now we're ready to commit the changes and update our environment using `eb deploy`.

```
rails-beanstalk $ git add .
rails-beanstalk $ git commit -m "welcome page controller, view and route"
rails-beanstalk $ eb deploy
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instances.
INFO: Environment update completed successfully.
```

The update process is fairly quick. Read the front page at the command line using Curl or navigate to the type `eb open` to open the site in a web browser to see the results.

```
$ eb open
Welcome

This is the front page of my first Rails application on Elastic Beanstalk.
```

Now you're ready to continue work on your Rails site. Whenever you have new commits to push, use `eb deploy` to update your environment.

Clean Up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `terminate` command to terminate your environment and the `delete` command to delete your application.

To terminate your environment and delete the application

- From the directory where you created your local repository, type the following command:

  ```
  eb terminate
  ```

  This process may take a few minutes. Elastic Beanstalk displays a message once the environment has been successfully terminated.

  > **Note**
  > If you attached an Amazon RDS DB instance to your environment, your Amazon RDS DB will be deleted, and you will lose your data. To save your data, create a snapshot before you delete the application. For instructions on how to create a snapshot, go to Creating a DB Snapshot in the *Amazon Relational Database Service User Guide*.

Don't hesitate to terminate an environment to save on resources while you continue to develop your site. You can always recreate your Beanstalk environment using `eb create`.

Deploying a Sinatra Application to AWS Elastic Beanstalk

This walkthrough shows how to deploy a simple Sinatra web application to AWS Elastic Beanstalk (Elastic Beanstalk) using the EB Command Line Interface (EB CLI).

Topics

Prerequisites

This walkthrough requires a Linux, Windows or OS X workstation. Performing the walkthrough will modify your workstation's Git and EB CLI configuration. If you do not want to modify your workstation's configuration for the walkthrough, you can use one of the following:

- An instance running in a virtual machine on your workstation.

  This walkthrough was prepared using Vagrant to run a Ubuntu 14.04 LTS instance in VirtualBox.

- An Amazon Elastic Compute Cloud (Amazon EC2) instance.

  Use SSH to log in to the instance. You can perform the entire walkthrough from the command line. When you have finished, you can terminate the instance.

The following tools are required to complete this walkthrough:

- The EB CLI, installed as described in Install the EB Command Line Interface (CLI) (p. 385).

  This topic also describes how to sign up for an AWS account, if you do not have one.
- AWS credentials that have permissions to create the AWS resources that make up the application's environment on your system.

  These credentials allow the EB CLI to act on your behalf to create the environment's resources. If you do not have stored credentials, the EB CLI prompts you for credentials when you create the application. For more information on how to store credentials and how the EB CLI handles stored credentials, see Access Key Provider Chain (p. 392). For more information on the required permissions, see Using Elastic Beanstalk with AWS Identity and Access Management (IAM) (p. 326).
- Git

  For Linux systems, you can use the package manager to install Git. For example, the following command installs Git on Debian-family Linux systems, such as Ubuntu.

  ```
  $ sudo apt-get install git
  ```

  For Red Hat-family systems, you can use the same command, but you use the package manager name `yum`. For more information, including directions for installing Git on OS X systems, see git.

Step 1: Set Up Your Project

With the EB CLI, you can quickly create an Elastic Beanstalk environment (p. 50) and deploy applications to that environment from a Git repository. Before starting your first project, set up the example project for the walkthrough.

To set up the example project

1. Open a terminal window and create a directory for your project in a convenient location on your system. This walkthrough assumes that the directory is named `sinatraapp`.

   ```
   ~$ mkdir sinatraapp
   ```

2. Move to the `sinatraapp` directory and initialize a Git repository.

   ```
   ~$ cd sinatraapp
   ~sinatraapp$ git init .
   ```

   Note
   You do not need to have access to a remote repository, such as GitHub, for this walkthough. The walkthrough uses a local Git repository.
3. If this is your first time using Git, add your user name and email address to the Git configuration so you can commit changes.

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Username"
```

Step 2: Create an Application

Now create an application and its associated environment (p. 50).

To create an application

1. In the sinatraapp directory, create the application by running the following command.

```
~/sinatraapp$ eb init
```

2. Choose the default AWS region. For this walkthrough, choose **US-West-2**.

```
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-southeast-1 : Asia Pacific (Singapore)
7) ap-southeast-2 : Asia Pacific (Sydney)
8) ap-northeast-1 : Asia Pacific (Tokyo)
9) sa-east-1 : South America (Sao Paulo)
(default is 3): 3
```

Tip
If you have previously configured a default region with the EB CLI, the AWS CLI or an SDK, the EB CLI skips this step and creates the application in the default region unless you explicitly specify a region with the `--region` (p. 399) option.

3. Provide a set of AWS credentials with appropriate permissions (p. 326). If you have an appropriate set of stored credentials (p. 392), the EB CLI uses them automatically and skips this step.

Important
We strongly recommend that you do not provide your account's root credentials to Elastic Beanstalk. Instead, create an AWS Identity and Access Management (IAM) user with appropriate permissions and provide those credentials. For more information on managing AWS credentials, see Best Practices for Managing AWS Access Keys.

If you do not have stored credentials, or your stored credentials do not grant the correct permissions, `eb init` prompts you for credentials, as follows:

```
You have not yet set up your credentials or your credentials are incorrect
You must provide your credentials.
(aws-access-id): AKIAIOSFODNN7EXAMPLE
(aws-secret-key): wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Elastic Beanstalk then stores these credentials in the AWS CLI `config` file with a profile name of `eb-cli`.

4. Enter your application name. For this walkthrough, use the default value, which is the application's root directory name.

```
Enter Application Name
(default is "sinatraapp"): sinatraapp
```

5. Specify your platform. For this walkthrough, use **Ruby**.

```
Select a platform.
1) PHP
2) Node.js
3) IIS
4) Tomcat
5) Python
6) Ruby
7) Docker
8) GlassFish
9) Go
(default is 1): 6
```

6. Specify the platform version. For this example, use **Ruby 2.1 (Puma)**.

```
Select a platform version.1
1) Ruby 2.1 (Puma)
2) Ruby 2.1 (Passenger Standalone)
3) Ruby 2.0 (Puma)
4) Ruby 2.0 (Passenger Standalone)
5) Ruby 1.9.3
(default is 1): 1
```

7. Specify whether you want to use SSH to log in to your instances. You won't need to log in to your instances for this walkthrough, so enter n.

```
Do you want to set up SSH for your instances?
(y/n): n
```

Step 3: Create an Environment

You can now create the application environment, which is a set of AWS resources that support your application. For this example, the environment includes the following.

- An Amazon Simple Storage Service (Amazon S3) bucket
- A security group.
- An Elastic Load Balancing (ELB) load balancer
- An Auto Scaling group
- An Amazon CloudWatch (Cloud Watch) alarm
- An Amazon EC2 instance, which is a member of the Auto Scaling group

To create the environment

1. Run the following command to create the environment.

   ```
   ~/sinatraapp$ eb create
   ```

   The command will prompt you for configuration settings, as described in the following steps.
2. Specify an environment name. For this walkthrough, accept the default value, which is `-dev` appended to the application name.

   ```
   Enter Environment Name
   (default is sinatraapp-dev): sinatraapp-dev
   ```

3. Specify a DNS CNAME prefix. For this walkthrough, accept the default value, which is the environment name.

   ```
   Enter DNS CNAME prefix
   (default is sinatraapp-dev): sinatraapp-dev
   ```

After you specify the DNS CNAME prefix, the EB CLI creates the environment's AWS resources and automatically deploys the application. Because you haven't yet created an application, the EB CLI deploys a default sample application. The initial deployment process creates the environment's resources in addition to deploying the application, so it usually takes several minutes. After deployment has finished, you can view the default application by running the `eb open` command, which opens the application in your default browser.

Step 4: Deploy a Simple Sinatra Application

You can now create and deploy a Sinatra application. This step describes how to implement a simple Sinatra application and deploy it to the environment that you created in the preceding step. Our example implements a classic application that prints a simple text string, `Hello World!`. You can easily extend this example to implement more complex classic applications or modular applications.

> **Note**
> Create all of the application files in the following procedure in the application's root directory, `sinatraapp`.

To create and deploy a Sinatra application

1. Create a configuration file named **config.ru** with the following contents.

   ```
   require './helloworld'
   run Sinatra::Application
   ```

2. Create a Ruby code file named **helloworld.rb** with the following contents.

   ```
   require 'sinatra'
   get '/' do
     "Hello World!"
   end
   ```

3. Create a **Gemfile** with the following contents.

```
source 'http://rubygems.org'
gem 'sinatra'
```

4. Add your files to the Git repository and then commit your changes, as follows:

```
~/sinatraapp$ git add .
~/sinatraapp$ git commit -m "Add a simple Sinatra application"
```

   You should see output similar to the following indicating that your files were successfully committed.

```
[master (root-commit) dcdfe6c] Add a simple Sinatra application
 4 files changed, 13 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Gemfile
 create mode 100644 config.ru
 create mode 100644 helloworld.rb
```

   The files are committed to the current Git branch. Because you have not yet explicitly created any branches, the files are committed to the default branch, which is named `master`. If the repository has multiple branches, you can configure Git to push each branch to a different environment. For more information, see Using the EB CLI (p. 394).

5. Deploy the new Sinatra application to the environment.

```
~/sinatraapp$ eb deploy
```

   The `eb deploy` command creates a bundle (p. 43) of the application code in the master branch and deploys it to the environment, replacing the default application. The second deployment should be much faster than the first because you have already created the environment's AWS resources.

6. Run the `eb status --verbose` command to check your environment status. You should see output similar to the following.

```
~/sinatraapp$ eb status --verbose
Environment details for: sinatraapp-dev
  Application name: sinatraapp
  Region: us-west-2
  Deployed Version: dcdf
  Environment ID: e-kn7feaqre2
  Platform: 64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Puma)
  Tier: WebServer-Standard
  CNAME: sinatraapp-dev.elasticbeanstalk.com
  Updated: 2015-03-03 23:15:19.183000+00:00
  Status: Ready
  Health: Green
  Running instances: 1
      i-c2e712cf: InService
```

   Repeat the command until **Status** is **Ready** and **Health** is **Green**. Then, refresh your browser or run `eb open` again to view the updated application, which should display `Hello World!`.

Tip

For a detailed description of the deployment, you can display the deployment log by running `eb logs`.

Step 5: Clean Up

When you have finished, you can terminate the application's environment by running the following command from the root directory, `sinatraapp`.

```
~/sinatraapp$ eb terminate
```

This command shuts down all of the environment's AWS resources, so you do not incur further charges. It typically takes a few minutes. When the process is complete, Elastic Beanstalk displays the following message.

```
INFO: terminateEnvironment completed successfully.
```

Tip

If you have attached an Amazon Relational Database Service (Amazon RDS) database instance to your environment, termination deletes the instance. If you want to save your data, create a snapshot before terminating the environment. For more information, see Creating a DB Snapshot. For more information about using Amazon RDS with Elastic Beanstalk, see Using Amazon RDS with Ruby.

Related Resources

For more information about Git commands, see Git - Fast Version Control System.

This walkthrough uses only a few of the EB CLI commands. For a complete list run `eb --help` or see EB CLI Command Reference (p. 399).

Configuring Ruby Environments with Elastic Beanstalk

Customizing and Configuring a Ruby Environment

When deploying your Ruby application, you may want to customize and configure the behavior of your Amazon EC2 instances. You can easily customize your instances at the same time that you deploy your application version by including a configuration file with your source bundle. This section walks you through the process of creating a configuration file and bundling it with your source.

Note

Note

Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

To customize and configure your Ruby environment

1. Create a configuration file with the extension **.config** (e.g., myapp.config) and place it in an **.ebextensions** top-level directory of your source bundle. You can have multiple configuration files in your **.ebextensions** directory. For information about file format and contents of the configuration file, see Using Configuration Files (p. 99).

   The following is an example snippet of a configuration file.

   ```
   # Configure third-party service credentials
   # in environment variables:
   option_settings:
     - option_name: AIRBRAKE_API_KEY
       value: MYAPIKEY

   # Run rake tasks before an application deployment
   container_commands:
     01deploy:
       command: rake my_deployment_tasks
   ```

   Note
   If you want to set environment variables that will be available to your application, you do not need to provide a "namespace" key in the option_settings section.

   You can also pass in your access credentials. For example, you could specify the following:

   ```
   # If you do not specify a namespace, the default used is aws:elasticbean
   stalk:application:environment
   option_settings:
     - option_name: PARAM1
       value: somevalue
   ```

2. Deploy your application version.

For an example walkthrough of deploying a Rails application, see Deploying a Rails Application to Elastic Beanstalk (p. 771). For an example walkthrough of deploying a Sinatra application, see Deploying a Sinatra Application to AWS Elastic Beanstalk (p. 779).

Accessing Environment Variables

Inside the Ruby environment running in Elastic Beanstalk, environment variables are accessible using ENV['VARIABLE_NAME'].

You might have a code snippet that looks similar to the following:

```
param1 = ENV['MYPARAM']
param2 = ENV['MYPARAM2']
```

Note
Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

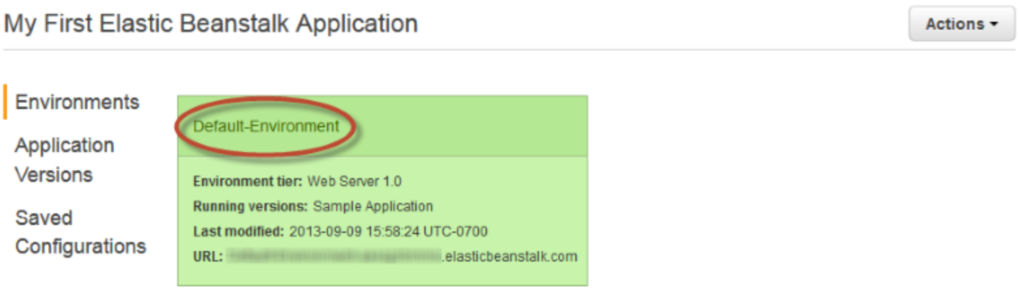
For a list of configuration options, see Ruby Container Options (p. 126).

AWS Management Console

The Ruby settings lets you enable or disable Amazon S3 log rotation.

To access the Ruby container configurations for your Elastic Beanstalk application

1. Open the Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. From the Elastic Beanstalk console applications page, click the environment that you want to configure.



My First Elastic Beanstalk Application

Environment tier: Web Server 1.0
Running versions: Sample Application
Last modified: 2013-09-09 15:58:24 UTC-0700
URL: [redacted].elasticbeanstalk.com

3. In the **Overview** section of the environment dashboard, click **Edit**.
4. On the **Configuration** page, click ⚙ for **Software Configuration** in order to edit the container settings.

Log Options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3**–Specifies whether log files for your application's Amazon EC2 instances should be copied to your Amazon S3 bucket associated with your application.

Environment Properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

You can configure the following environment settings:

- **BUNDLE_WITHOUT**–A colon-separated list of groups to ignore when installing dependencies from a Gemfile. For more information, go to http://gembundler.com/groups.html.
- Specify additional environment configuration settings by entering them in the **PARAM** boxes.

  Note

  Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.
- **RACK_ENV**–Specifies the environment stage (e.g., development, production, test) in which an application can be run.
- **RAILS_SKIP_ASSET_COMPILIATION**–Specifies whether to run `rake assets:precompile` on behalf of the users applications, or simply skip it. This is only applicable to Rails 3.x applications.

- **RAILS_SKIP_MIGRATIONS**–Specifies whether the container should run `rake db:migrate` on behalf of users' applications; or simply skip it. This is only applicable to Rails 3.x applications. For non-Rails migrations, you should use a file with the extension `.config` (e.g., `myconfig.config`) file to specify the container command to manually run their migrations. If you set **RAILS_SKIP_MIGRATIONS** to **true**, you should run migrations using a configuration file. For more information on using configuration files, see Customizing and Configuring a Ruby Environment (p. 785).

Accessing Environment Variables

Inside the Ruby environment running in Elastic Beanstalk, environment variables are accessible using ENV['VARIABLE_NAME'].

You might have a code snippet that looks similar to the following:

```
param1 = ENV['MYPARAM']
param2 = ENV['MYPARAM2']
```

> **Note**
> Environment configuration settings can contain any printable ASCII character except the grave accent (`, ASCII 96) and cannot exceed 200 characters in length.

Command Line Interface (CLI)

To edit an application's environment configuration settings

- Update an application's environment configuration settings.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings file://options.txt
```

options.txt

```
[
    {
        "Namespace": "aws:elasticbeanstalk:hostmanager",
        "OptionName": "LogPublicationControl",
        "Value": "false"
    }
]
```

API

To edit an application's environment configuration settings

- Call `UpdateEnvironment` with the following parameters:
  - *`OptionSettings.member.1.Namespace`* = `aws:elasticbeanstalk:hostmanager`
  - *`OptionSettings.member.1.OptionName`* = `LogPublicationControl`
  - *`OptionSettings.member.1.Value`* = `false`

Example

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&OptionSettings.member.1.Namespace=aws%3Aelasticbeanstalk%3Ahostmanager
&OptionSettings.member.1.OptionName=LogPublicationControl
&OptionSettings.member.1.Value=false
&Operation=UpdateEnvironment
&AuthParams
```

Using Amazon RDS with Ruby

With Amazon Relational Database Service (Amazon RDS), you can quickly and easily provision and maintain a MySQL, Oracle, or Microsoft SQL Server instance in the cloud. This topic explains how you can use Amazon RDS and Ruby with your Elastic Beanstalk application. For more information about Amazon RDS, go to http://aws.amazon.com/rds/.

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

1. Create an Amazon RDS DB instance.
2. Establish a database connection in your code by using the connectivity information for your Amazon RDS DB instance.
3. Deploy your application to Elastic Beanstalk.

This topic walks you through the following:

- Using a new Amazon RDS DB instance with your application
- Using an existing Amazon RDS DB instance with your application

Using a New Amazon RDS DB Instance with Ruby

This topic walks you through creating a new Amazon RDS DB Instance and using it with your Ruby application.

To use a new Amazon RDS DB Instance and Ruby from your Elastic Beanstalk application

1. Create an Amazon RDS DB instance. You can create an RDS DB instance in one of the following ways:

   - Create an Amazon RDS DB instance when you create a new application version. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of a Rails application deployment with Amazon RDS using eb, see Deploying a Rails Application to Elastic Beanstalk (p. 771).
   - Create an Amazon RDS DB instance when you launch a new environment with an existing application version. For instructions using the AWS Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55).
   - If you already deployed an application to Elastic Beanstalk, you can create an Amazon RDS DB instance and attach it to an existing environment. For instructions using the AWS Elastic Beanstalk console, see Configuring Databases with Elastic Beanstalk (p. 195). If you use the EB CLI, use the `--database` option when you run `eb create`.

2. Establish a database connection in your code using your Amazon RDS DB Instance's connectivity information. You can access your connectivity information using environment variables. The following shows how you would connect to the database on an RDS instance.

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= ENV['RDS_DB_NAME'] %>
  username: <%= ENV['RDS_USERNAME'] %>
  password: <%= ENV['RDS_PASSWORD'] %>
  host: <%= ENV['RDS_HOSTNAME'] %>
  port: <%= ENV['RDS_PORT'] %>
```

3. Deploy the updated application to your Elastic Beanstalk environment.

Using an Existing Amazon RDS DB Instance with Ruby

You can update your Ruby application to use an Amazon RDS DB Instance that you have previously created. This topic walks you through how to update your Ruby application using an existing Amazon RDS DB Instance and deploy your application to Elastic Beanstalk.

To use an existing Amazon RDS DB Instance and Ruby from your Elastic Beanstalk application

1. Create an Elastic Beanstalk environment in one of the following ways:

   - Create a new application with a new environment. For instructions using the Elastic Beanstalk console, see Create an Application (p. 34). For a sample walkthrough of a Rails application deployment with Amazon RDS using eb, see Deploying a Rails Application to Elastic Beanstalk (p. 771). You do not need to create an RDS DB Instance with this environment because you already have an existing RDS DB Instance.
   - Launch a new environment with an existing application version. For instructions using the Elastic Beanstalk console, see Launching a New AWS Elastic Beanstalk Environment (p. 55). You do not need to create an Amazon RDS DB instance with this environment because you already have an existing Amazon RDS DB instance.

2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Establish a database connection in your code using your Amazon RDS DB instance's connectivity information. The following examples show how you could connect to the database on an RDS instance at mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

```
production:
  adapter: mysql2
  encoding: utf8
  database: exampledb
  username: sa
  password: mypassword
```

```
host: mydbinstance.abcdefghijkl.us-west-2.rds.amazonaws.com
port: 3306
```

4. Deploy the updated application to your Elastic Beanstalk environment.

Tools

AWS SDK for Ruby

You can get started in minutes with a single, downloadable package complete with the AWS Ruby library, code samples, and documentation. You can build Ruby applications on top of APIs that take the complexity out of coding directly against web services interfaces. The all-in-one library provides Ruby developer-friendly APIs that hide much of the lower-level tasks associated with programming for the AWS cloud, including authentication, request retries, and error handling. Practical examples are provided in Ruby for how to use the libraries to build applications. For information about the SDK, sample code, documentation, tools, and additional resources, go to http://aws.amazon.com/ruby/.

Git Deployment Via EB CLI

EB CLI is an AWS Elastic Beanstalk that enables you to deploy applications quickly and more easily from the command line. To learn how to get started deploying a Ruby application to Elastic Beanstalk using eb and Git, see Deploying Elastic Beanstalk Applications in Ruby Using EB CLI and Git (p. 771).

Resources

There are several places you can go to get additional help when developing your Ruby applications:

Resource	Description
AWS SDK for Ruby	Install AWS SDK for Ruby.
Ruby Development Forum	Post your questions and get feedback.
Ruby Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Elastic Beanstalk Resources

The following related resources can help you as you work with this service.

- **Elastic Beanstalk API Reference** s A comprehensive description of all SOAP and Query APIs. Additionally, it contains a list of all SOAP data types.
- **Elastic Beanstalk Sample Code and Libraries** – A link to the command line tool as well as a sample Java web application. See the links below for additional sample applications.
- **Elastic Beanstalk Technical FAQ** – The top questions developers have asked about this product.
- **Elastic Beanstalk Release Notes** – A high-level overview of the current release. This document specifically notes any new features, corrections, and known issues.

- **AWS Training and Courses** – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- **AWS Developer Tools** – Links to developer tools and resources that provide documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
- **AWS Support Center** – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- **AWS Support** – The primary web page for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- **Contact Us** – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- **AWS Site Terms** – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Sample Applications

The following are download links to the sample applications that are deployed as part of Getting Started Using Elastic Beanstalk (p. 4).

- **Java with Tomcat**– elasticbeanstalk-sampleapp.war
- **.NET on Windows Server with IIS**– FirstSample.zip
- **Node.js**– nodejs-sample.zip
- **PHP**– php-newsample-app.zip

- **Python**– python-sample-20150402.zip
- **Ruby (Passenger Standalone)**– ruby-sample.zip
- **Ruby (Puma)**– ruby2PumaSampleApp.zip
- **Single Container Docker** – docker-sample-v3.zip
- **Preconfigured Docker (Glassfish)**– glassfish-sample.war
- **Preconfigured Docker (Python 3.x)**– python3-sample.zip
- **Preconfigured Docker (Go)**–golang-sample.zip

Document History

The following table describes the important changes to the documentation since the last release of *Elastic Beanstalk*.

API version: 2010-12-01*

*The service's API version is only updated when breaking changes are made to the API.

Latest documentation update: June 16, 2015

Change	Description	Date Changed
New content	New topic on Bundling Multiple WAR Files for Java with Tomcat Environments (p. 48).	16 June 2015
Updated content	Updated Supported Platforms (p. 24) with detailed information related to new container types released on May 28, 2015.	28 May 2015
Updated content	Updated Supported Platforms (p. 24) with detailed information related to new container types released on May 26, 2015.	26 May 2015
New and updated content	Updated autoscaling documentation, including adding new content about time-based scaling that enables you to schedule scaling actions.	14 May 2015
Updated content	Updated Supported Platforms (p. 24) with detailed information about new container types released on May 7, 2015 that address ALAS-2015-522.	7 May 2015
Updated content	Updated Supported Platforms (p. 24) with detailed information related to new container types released on April 30, 2015.	30 April 2015
Updated content	Updated Supported Platforms (p. 24) with detailed information related to new container types released on April 21, 2015.	21 April 2015
Updated content	Updated Supported Platforms (p. 24) Windows and .NET table with detailed information related to new container types released on April 16, 2015.	16 April 2015
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on April 7, 2015.	7 April 2015

Change	Description	Date Changed
New pages for Multicontainer Docker platform	Added new topics describing the Multicontainer Docker platform, version 2 of `Dockerrun.aws.json`file format, and a tutorial describing their use under Deploying Elastic Beanstalk Applications from Docker Containers (p. 540)	24 March 2015
New and updated content	Added information about: • version 1.2 of worker environment tiers, including new support for periodic tasks • new and updated commands for version 3.1 of EB CLI • cloning an environment to use a newer solution stack version	17 February 2015
New content	Added content about using preconfigured Docker containers for Go applications.	6 February 2015
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on January 28, 2015 that address CVE-2015-0235 Advisory (Ghost).	28 January 2015
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on December 12, 2014 that address ALAS-2014-461.	12 December 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on November 25, 2014.	25 November 2014
New and updated content	Added content about using preconfigured Docker containers to deploy Elastic Beanstalk applications. Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on November 5, 2014, including preconfigured Docker containers.	5 November 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on October 30, 2014.	30 October 2014
New and updated content	Added and updated content about using EB Command Line Interface (CLI) 3.x to deploy and manage Elastic Beanstalk applications.	29 October 2014
New and updated content	• Added content about deploying application versions in batches of instances. • Updated content about creating Launch Now URLs.	28 October 2014
Updated content	Updated table for `aws:autoscaling:launchconfiguration` namespace in General Options for all Environments (p. 103) to include valid instance types for newly supported eu-central-1 region.	23 October 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on October 15, 2014 that address CVE-2014-3566 Advisory.	16 October 2014

Change	Description	Date Changed
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on October 9, 2014.	9 October 2014
Updated content	Added procedures for editing Amazon RDS database instance settings for your environment to Configuring Databases with Elastic Beanstalk (p. 195).	8 October 2014
New content	Added Eb Operations (p. 453) reference for eb command line interface.	7 October 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new 32-bit container types released on September 26, 2014 that address ALAS-2014-419.	26 September 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on September 25, 2014 that address ALAS-2014-419.	26 September 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to new container types released on September 24, 2014 that address CVE-2014-6271 Advisory.	25 September 2014
Updated content	• Added information about new support for Elastic Load Balancing cross-zone availability and connection draining. • Updated Instance Logs (p. 282) to describe new bundle logs option.	27 August 2014
Updated content	• Updated Supported Platforms (p. 24) Ruby section with new versions of Ruby container types released on 14 August 2014. • Updated Deploying a Rails Application to Elastic Beanstalk (p. 771) with information pertaining to new Ruby container types.	14 August 2014
Updated content	Updated Supported Platforms (p. 24) Windows and .NET section with new versions of .NET container types released on 6 August 2014.	6 August 2014
New content	Added content to support new integration with Amazon CloudWatch Logs.	10 July 2014
Updated content	Updated Supported Platforms (p. 24) tables with new versions of container types released on 30 June 2014.	3 July 2014
Updated content	New support for `t2` instance types for all regions.	30 June 2014
Updated content	Added information about Docker 1.0 containers to Supported Platforms (p. 24).	16 June 2014

Change	Description	Date Changed
Updated content	- Updated Supported Platforms (p. 24) tables with detailed information related to new container types that address the June 5, 2014 OpenSSL Security Advisory. - New support for g2 instance types in the ap-northeast-1 region. - Support for c3 instance types in all regions.	5 June 2014
New content	Added information about version 1.1 of worker environment tiers.	27 May 2014
New content	Added information about support for Docker containers as a new solution stack.	23 April 2014
New content	Added information about support for tagging environments.	22 April 2014
New content	Added information about support for a new VPC configuration that does not require a NAT instance, including a new setting that associates Amazon EC2 instances with public IP addresses.	9 April 2014
New and updated content	- Updated Supported Platforms (p. 24) tables with detailed information related to new support for Ruby 2.0 - Added information about new support for Launch Now URLs	2 April 2014
Updated content	Updated Supported Platforms (p. 24) tables with detailed information related to: - new support for Amazon Linux 2014.02 container types - new support for Tomcat 7.0.47 and Apache 2.2.26 for Amazon Linux 2014.02 container types running Java 7 and Java 6 with Tomcat 7 platforms - new support for Nginx 1.4.3 for Amazon Linux 2013.09 container types running Node.js - new support for g2, i2, and m3 instance types	18 March 2014
New content	Added new content to support the concept of environment tiers, particularly a new type of environment tier called a worker tier that performs background-processing tasks.	11 December 2013
New and updated content	Updated Supported Platforms (p. 24) tables with detailed information related to: - new support for Amazon Linux AMI version 2013.09 - new support for Python 2.7 and Java 7 with Tomcat 7 platforms - changes to container names Added new content to support using rolling updates to manage changes in your environment.	8 November 2013

Change	Description	Date Changed
New content	Added tables with detailed information about supported platforms, including new container names and AMI versions.	25 October 2013
New content	Added content about new setting that supports customizing timeout for commands.	23 October 2013
New content	Added content about configuring SSL on single-instance environments.	20 September 2013
New and updated content	• Added content about mapping instance store volumes. • Updated content about RDS password-restricted characters. • Updated information about most recent supported version of Node.js.	21 August 2013
New and updated content	• Added content about environment types (load balanced, autoscaled and single instance). • Added content about creating environments in a VPC by using the AWS management console. • Updated content to support the new Elastic Beanstalk console.	17 July 2013
Updated content	Updated content to support instance profiles.	17 April 2013
Updated content	Updated .NET section for support for VPC, RDS, and Configuration Files.	8 April 2013
Updated content	Updated content for Node.js container support.	11 March 2013
Updated content	Updated PHP section for non-legacy container support. Added content for customizing environment resources.	18 December 2012
New container type	Added support for Windows Server 2012 running IIS 8.	19 November 2012
New content	Added content to support Ruby and Amazon VPC.	01 November 2012
New content	Added content for customizing container types. Added content for migrating applications using legacy container types to non-legacy container types.	02 October 2012
New content	Added content to support Asia Pacific (Singapore) Region.	04 September 2012
New content	Added content for deploying Python applications and Amazon RDS integration.	19 August 2012
New content	Added content to support US West (Oregon)Region and US West (N. California) Region.	10 July 2012
New content	Added Get Started section for command line interface.	27 June 2012
New content	Added content for deploying .NET applications using the standalone deployment tool.	29 May 2012

Change	Description	Date Changed
New content	Added content to support the EU (Ireland) Region.	16 May 2012
New content	Added new section for deploying .NET applications.	08 May 2012
New content	Added content to support the Asia Pacific (Tokyo) Region.	23 April 2012
New content	Added new section for deploying PHP applications using Git.	20 March 2012
New content	You can create a policy that allows or denies permissions to perform specific AWS Elastic Beanstalk actions on specific AWS Elastic Beanstalk resources.	06 March 2012
New content	Added new section for managing and configuration application and environments using the AWS Toolkit for Eclipse.	10 August 2011
New content	Combined *AWS Elastic Beanstalk Getting Started Guide* and *AWS Elastic Beanstalk User Guide* into one guide: *AWS Elastic Beanstalk Developer Guide*. Added new content for saving and editing environment configuration settings as well as swapping environment URLs. Added new section for CLI reference.	29 June 2011
New container type	This is the added support of Java Tomcat 7 for Elastic Beanstalk.	21 April 2011
New content	Added new topic Using Amazon RDS and MySQL Connector/J (p. 592).	04 March 2011
New content	Updated some topics for new console user interface. Added new topics Customizing Environments and Using Custom Environment Properties with Elastic Beanstalk (p. 589).	17 February 2011
New service	This is the initial release of Elastic Beanstalk.	18 January 2011

Appendix

Topics

Customizing AWS Resources

This section details the supported resources and type names available for customizing Elastic Beanstalk environments.

For information on updating AWS CloudFormation stacks, see AWS CloudFormation Stack Updates in the AWS CloudFormation User Guide.

Topics

AWS Resource Types Reference

The following table lists the supported AWS resources and API versions used with Elastic Beanstalk applications.

As we add support for more resources, resource type identifiers will take the following form:

```
AWS::aws-product-name::data-type-name
```

AWS Resource	Resource Type Identifier	Supported API Version
Amazon CloudWatch	AWS::CloudWatch::Alarm	2010-05-15

AWS Resource	Resource Type Identifier	Supported API Version
DynamoDB Table	AWS::DynamoDB::Table	2010-05-15
Amazon ElastiCache Cache Cluster	AWS::ElastiCache::CacheCluster	2011-07-15
Amazon ElastiCache Security Group	AWS::ElastiCache::SecurityGroup	2011-07-15
Amazon ElastiCache Security Group Ingress	AWS::ElastiCache::SecurityGroupIngress	2011-07-15
Amazon SNS Subscription	AWS::SNS::Subscription	2010-03-31
Amazon SNS Topic	AWS::SNS::Topic	2010-03-31
Amazon SQS Queue	AWS::SQS::Queue	2009-02-01

Use `AWS::Region` to get the value of the region for the Elastic Beanstalk environment. Use this with the Ref function to retrieve the value.

Resource Property Types Reference

See the following links for information about resource types supported by Elastic Beanstalk Extensions.

- CloudWatch Metric Dimension Property Type
- DynamoDB Primary Key
- DynamoDB Provisioned Throughput
- SNS Subscription Property Type

Intrinsic Function Reference

The following are several built-in functions that help you manage your environment resources.

- Fn::GetAtt
- Fn::Join
- Fn::GetOptionSetting (p. 801)
- Ref

Fn::GetOptionSetting

The intrinsic function `Fn::GetOptionSetting` returns the value of the specified option name. If no value is set, then the specified default value is used. To specify the value of the custom option, you create a separate configuration file with the namespace `aws:elasticbeanstalk:customoption`.

Note

Note
Configuration files must conform to YAML or JSON formatting standards. For example, indentation is critical to the proper interpretation of YAML. For more information, go to http://www.yaml.org/start.html or http://www.json.org, respectively. For more information

about using configuration files to deploy an application to Elastic Beanstalk, see Using Configuration Files (p. 99).

Declaration

```
Fn::GetOptionSetting:
  OptionName: name of option
  DefaultValue: default value to be used
```

Example

This example returns the value of the ELBAlarmEmail. If no value is set, then it uses "someone@example.com".

```
Subscription:
  - Endpoint:
      Fn::GetOptionSetting:
        OptionName: ELBAlarmEmail
        DefaultValue: "someone@example.com"
```

You can find more example snippets using `Fn::GetOptionSetting` at Example Snippets (p. 147).

Create a separate configuration file (e.g., options.config) that sets the value of the custom option. To set the value for above example, your configuration file would look like the following:

```
option_settings:
  "aws:elasticbeanstalk:customoption":
  ELBAlarmEmail : "someone@example.com"
```

For an example walkthrough using configuration files and `Fn::GetOptionSetting`, see Example: DynamoDB, CloudWatch, and SNS (p. 138).

Platform History

Topics

Docker Platform History

This page lists the previous versions of AWS Elastic Beanstalk's Docker platform configurations and the dates that each version was current. Configurations that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with Docker containers between August 3, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	**AMI**	**Docker Version**	**Web Server**
Single Container Docker 1.6 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Docker 1.6.2*	2015.03	1.6.2	Nginx 1.6.2
Multicontainer Docker 1.6 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Multi-container Docker 1.6.2 (Generic)*	2015.03	1.6.2	none

Elastic Beanstalk supports the following container types for environments created with Docker containers between July 23, 2015 and August 3, 2015:

Configuration and *Solution Stack Name*	**AMI**	**Docker Version**	**Web Server**
Single Container Docker 1.6 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2*	2015.03	1.6.2	Nginx 1.6.2
Multicontainer Docker 1.6 version 1.4.5 *64bit Amazon Linux 2015.03 v1.4.5 running Multi-container Docker 1.6.2 (Generic)*	2015.03	1.6.2	none

Elastic Beanstalk supports the following container types for environments created with Docker containers between June 15, 2015 and July 23, 2015:

Configuration and *Solution Stack Name*	**AMI**	**Docker Version**
Single Container Docker 1.6 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2*	2015.03	1.6.2

Configuration and *Solution Stack Name*	AMI	Docker Version
Multicontainer Docker 1.6 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Multi-container Docker 1.6.2 (Generic)*	2015.03	1.6.2

Elastic Beanstalk supports the following container types for environments created with Docker containers between May 27, 2015 and June 15, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2015.03 v1.4.1 running Single Container Docker 1.6.0	2015.03	1.6.0
64bit Amazon Linux 2015.03 v1.4.1 running Multi-container Docker 1.6.0 (Generic)	2015.03	1.6.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between May 8, 2015 and May 26, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2015.03 v1.4.0 running Docker 1.6.0[1]	2015.03	1.6.0
64bit Amazon Linux 2015.03 v1.4.0 running Multicontainer Docker 1.6.0 (Generic)[1]	2015.03	1.6.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between April 22, 2015 and May 7, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2015.03 v1.3.1 running Docker 1.5.0	2015.03	1.5.0
64bit Amazon Linux 2014.09 v1.2.1 running Multicontainer Docker 1.3.3 (Generic)	2014.09	1.3.3
64bit Amazon Linux 2014.09 v1.2.0 running Multicontainer Docker 1.3.3 (Generic)	2014.09	1.3.3

Elastic Beanstalk supports the following container types for environments created with Docker containers between March 24, 2015 and April 21, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09 v1.2.1 running Docker 1.5.0	2014.09	1.5.0
64bit Amazon Linux 2014.09 v1.2.0 running Multicontainer Docker 1.3.3 (Generic)	2014.09	1.3.3

Elastic Beanstalk supports the following container types for environments created with Docker containers between February 17, 2015 and March 23, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09 v1.2.0 running Docker 1.3.3	2014.09	1.3.3

Elastic Beanstalk supports the following container types for environments created with Docker containers between January 28, 2015 and February 16, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09[1] v1.1.0 running Docker 1.3.3	2014.09	1.3.3

Elastic Beanstalk supports the following container types for environments created with Docker containers between December 13, 2014 and January 27, 2015:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09[1] v1.0.11 running Docker 1.3.3	2014.09	1.3.3

Elastic Beanstalk supports the following container types for environments created with Docker containers between November 26, 2014 and December 12, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09 v1.0.10 running Docker 1.3.2	2014.09	1.3.2

Elastic Beanstalk supports the following container types for environments created with Docker containers between October 16, 2014 and November 25, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09 v1.0.9[1] running Docker 1.2.0	2014.09	1.2.0
64bit Amazon Linux 2014.03 v1.0.9[1] running Docker 1.0.0	2014.03	1.0.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between October 9, 2014 and October 15, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.09 v1.0.8 running Docker 1.2.0	2014.09	1.2.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between September 24, 2014 and October 8, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.7[1] running Docker 1.0.0	2014.03	1.0.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between June 30, 2014 and September 23, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.1 running Docker 1.0.0	2014.03	1.0.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between June 16, 2014 and June 29, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.0 running Docker 1.0.0	2014.03	1.0.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between June 5, 2014 and June 15, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.5[1] running Docker 0.9.0	2014.03	0.9.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between May 5, 2014 and June 4, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.4 running Docker 0.9.0	2014.03	0.9.0

Elastic Beanstalk supports the following container types for environments created with Docker containers between April 29, 2014 and May 4, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.3 running Docker 0.9.0	2014.03	0.9.0

Elastic Beanstalk supports the following container types for environments created with Docker containers prior to April 28, 2014:

Docker Configurations		
Name	**AMI**	**Docker Version**
64bit Amazon Linux 2014.03 v1.0.2 running Docker 0.9.0	2014.03	0.9.0

Preconfigured Docker Platform History

This page lists the previous versions of AWS Elastic Beanstalk's preconfigured Docker platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between August 3, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	AMI	Platform	Container OS	[illegible]
Glassfish 4.1 (Docker) version 1.4.6 *64bit Debian jessie v1.4.6 running GlassFish 4.1 Java 8 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Glassfish 4.0 (Docker) version 1.4.6 *64bit Debian jessie v1.4.6 running GlassFish 4.0 Java 7 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.4 (Docker) version 1.4.6 *64bit Debian jessie v1.4.6 running Go 1.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.3 (Docker) version 1.4.6 *64bit Debian jessie v1.4.6 running Go 1.3 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Python 3.4 with uWSGI 2 (Docker) version 1.4.6 *64bit Debian jessie v1.4.6 running Python 3.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between June 15, 2015 and August 3, 2015:

Configuration and *Solution Stack Name*	AMI	Platform	Container OS	[illegible]
Glassfish 4.1 (Docker) version 1.4.3 *64bit Debian jessie v1.4.3 running GlassFish 4.1 Java 8 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Glassfish 4.0 (Docker) version 1.4.3 *64bit Debian jessie v1.4.3 running GlassFish 4.0 Java 7 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.4 (Docker) version 1.4.3 *64bit Debian jessie v1.4.3 running Go 1.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Go 1.3 (Docker) version 1.4.3 *64bit Debian jessie v1.4.3 running Go 1.3 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]
Python 3.4 with uWSGI 2 (Docker) version 1.4.3 *64bit Debian jessie v1.4.3 running Python 3.4 (Preconfigured - Docker)*	2015.03	Docker 1.6.2	Debian Jessie	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between May 27, 2015 and June 15, 2015:

Preconfigured Docker Configurations				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]

Preconfigured Docker Configurations				
64bit Debian Jessie v1.4.1 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2015.03	Debian Jessie	1.6.0	[illegible] - no [illegible] 1.5.3
64bit Debian Jessie v1.4.1 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2015.03	Debian Jessie	1.6.0	[illegible] - no [illegible] 1.5.3
64bit Debian Jessie v1.4.1 running Go 1.4 (Preconfigured – Docker)	2015.03	Debian Jessie	1.6.0	[illegible]
64bit Debian Jessie v1.4.1 running Go 1.3 (Preconfigured – Docker)	2015.03	Debian Jessie	1.6.0	[illegible]
64bit Debian Jessie v1.4.1 running Python 3.4 (Preconfigured – Docker)	2015.03	Debian Jessie	1.6.0	[illegible] - no [illegible] 1.5.3

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between May 8, 2015 and May 26, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.4.0 running Glassfish 4.1 Java 8 (Preconfigured – Docker)[1]	2015.03	Debian Jessie	1.6.0	[illegible] - no [illegible] 1.5.3

Preconfigured Docker Container Types				
64bit Debian Jessie v1.4.0 running Glassfish 4.0 Java 7 (Preconfigured – Docker)[1]	2015.03	Debian Jessie	1.6.0	[illegible]
64bit Debian Jessie v1.4.0 running Go 1.4 (Preconfigured – Docker)[1]	2015.03	Debian Jessie	1.6.0	[illegible]
64bit Debian Jessie v1.4.0 running Go 1.3 (Preconfigured – Docker)[1]	2015.03	Debian Jessie	1.6.0	[illegible]
64bit Debian Jessie v1.3.1 running Python 3.4 (Preconfigured – Docker)[1]	2015.03	Debian Jessie	1.6.0	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between April 22, 2015 and May 7, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.3.1 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2015.03	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.3.1 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2015.03	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.3.1 running Go 1.4 (Preconfigured – Docker)	2015.03	Debian Jessie	1.5.0	[illegible]

Preconfigured Docker Container Types				
64bit Debian Jessie v1.3.1 running Go 1.3 (Preconfigured – Docker)	2015.03	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.3.1 running Python 3.4 (Preconfigured – Docker)	2015.03	Debian Jessie	1.5.0	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between March 24, 2015 and April 21, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.2.1 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.2.1 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.2.1 running Go 1.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.2.1 running Go 1.3 (Preconfigured – Docker)	2014.09	Debian Jessie	1.5.0	[illegible]
64bit Debian Jessie v1.2.1 running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.5.0	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between February 17, 2015 and March 23, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.2.0 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.2.0 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.2.0 running Go 1.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.2.0 running Go 1.3 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.2.0 running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between February 6, 2015 and February 16, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.1.0 running Go 1.3 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]

Preconfigured Docker Container Types				
64bit Debian Jessie v1.1.0 running Go 1.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between January 28, 2015 and February 5, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.1.0[1] running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.1.0[1] running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]
64bit Debian Jessie v1.1.0[1] running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between December 13, 2014 and January 27, 2015:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]

Preconfigured Docker Container Types				
64bit Debian Jessie v1.0.2 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3[1]	[illegible]
64bit Debian Jessie v1.0.2 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3[1]	[illegible]
64bit Debian Jessie v1.0.2 running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.3[1]	[illegible]

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers between November 26, 2014 and December 12, 2014:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	[illegible]
64bit Debian Jessie v1.0.1 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.2	[illegible]
64bit Debian Jessie v1.0.1 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.2	[illegible]

Preconfigured Docker Container Types				
64bit Debian Jessie v1.0.1 running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.3.2	[illegible] 1.5.3

Elastic Beanstalk supports the following container types for environments created with preconfigured Docker containers prior to November 25, 2014:

Preconfigured Docker Container Types				
Name	**AMI**	**Container Operating System**	**Docker Version**	**[illegible]**
64bit Debian Jessie v1.0.0 running Glassfish 4.1 Java 8 (Preconfigured – Docker)	2014.09	Debian Jessie	1.2.0	[illegible] 1.5.3
64bit Debian Jessie v1.0.0 running Glassfish 4.0 Java 7 (Preconfigured – Docker)	2014.09	Debian Jessie	1.2.0	[illegible] 1.5.3
64bit Debian Jessie v1.0.0 running Python 3.4 (Preconfigured – Docker)	2014.09	Debian Jessie	1.2.0	[illegible] 1.5.3

Java Platform History

This page lists the previous versions of AWS Elastic Beanstalk's Java platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following configurations for environments created with Java containers between July 31, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	**AMI**	**Language**	**Application Server**	**Web Server**
Java 8 with Tomcat 8 version 1.4.5 *64bit Amazon Linux 2015.03 v1.4.5 running Tomcat 8 Java 8*	2015.03	Java 1.8.0_51	Tomcat 8.0.20	Apache 2.2.29
Java 7 with Tomcat 7 version 1.4.5 *64bit Amazon Linux 2015.03 v1.4.5 running Tomcat 7 Java 7*	2015.03	Java 1.7.0_85	Tomcat 7.0.62	Apache 2.2.29
Java 6 with Tomcat 7 version 1.4.5 *64bit Amazon Linux 2015.03 v1.4.5 running Tomcat 7 Java 6*	2015.03	Java 1.6.0_35	Tomcat 7.0.62	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between July 7, 2015 and July 31, 2015:

Configuration and *Solution Stack Name*	**AMI**	**Language**	**Application Server**	**Web Server**
Java 8 with Tomcat 8 version 1.4.4 *64bit Amazon Linux 2015.03 v1.4.4 running Tomcat 8 Java 8*	2015.03	Java 1.8.0_45	Tomcat 8.0.20	Apache 2.2.29
Java 7 with Tomcat 7 version 1.4.4 *64bit Amazon Linux 2015.03 v1.4.4 running Tomcat 7 Java 7*	2015.03	Java 1.7.0_79	Tomcat 7.0.62	Apache 2.2.29
Java 6 with Tomcat 7 version 1.4.4 *64bit Amazon Linux 2015.03 v1.4.4 running Tomcat 7 Java 6*	2015.03	Java 1.6.0_35	Tomcat 7.0.62	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between June 15, 2015 and July 7, 2015:

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Java 8 with Tomcat 8 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Tomcat 8 Java 8*	2015.03	Java 1.8.0_45	Tomcat 8.0.20	Apache 2.2.29
Java 7 with Tomcat 7 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Tomcat 7 Java 7*	2015.03	Java 1.7.0_79	Tomcat 7.0.62	Apache 2.2.29
Java 6 with Tomcat 7 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Tomcat 7 Java 6*	2015.03	Java 1.6.0_35	Tomcat 7.0.62	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between May 27, 2015 and June 15, 2015:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2015.03 v1.4.1 running Tomcat 8 Java 8	2015.03	Java 1.8.0_31	Tomcat 8.0.20	Apache 2.2.29
64bit Amazon Linux 2015.03 v1.4.1 running Tomcat 7 Java 7	2015.03	Java 1.7.0_75	Tomcat 7.0.59	Apache 2.2.29
64bit Amazon Linux 2015.03 v1.4.1 running Tomcat 7 Java 6	2015.03	Java 1.6.0_34	Tomcat 7.0.59	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between April 22, 2015 and May 26, 2015:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.1 running Tomcat 8 Java 8	2015.03	Java 1.8.0_31	Tomcat 8.0.20	Apache 2.2.29

Java Configurations				
64bit Amazon Linux 2015.03 v1.3.1 running Tomcat 7 Java 7	2015.03	Java 1.7.0_75	Tomcat 7.0.59	Apache 2.2.29
64bit Amazon Linux 2015.03 v1.3.1 running Tomcat 7 Java 6	2015.03	Java 1.6.0_34	Tomcat 7.0.59	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between April 8, 2015 and April 21, 2015:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.0 running Tomcat 8 Java 8	2015.03	Java 1.8.0_31	Tomcat 8.0.20	Apache 2.2.29
64bit Amazon Linux 2015.03 v1.3.0 running Tomcat 7 Java 7	2015.03	Java 1.7.0_75	Tomcat 7.0.59	Apache 2.2.29
64bit Amazon Linux 2015.03 v1.3.0 running Tomcat 7 Java 6	2015.03	Java 1.6.0_34	Tomcat 7.0.59	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between February 17, 2015 and April 7, 2015:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2014.09 v1.1.0[1] running Tomcat 8 Java 8	2014.09	Java 1.8.0_31	Tomcat 8	Apache 2.2.29
64bit Amazon Linux 2014.09 v1.1.0[1] running Tomcat 7 Java 7	2014.09	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.09 v1.1.0[1] running Tomcat 7 Java 6	2014.09	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.1.0[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.71	Tomcat 7.0.55	Apache 2.2.29
64bit Amazon Linux 2014.03 v1.1.0[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.71	Tomcat 7.0.55	Apache 2.2.29
32bit Amazon Linux 2014.03 v1.1.0[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_33	Tomcat 7.0.55	Apache 2.2.29
64bit Amazon Linux 2014.03 v1.1.0[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_33	Tomcat 7.0.55	Apache 2.2.29

Elastic Beanstalk supports the following configurations for environments created with Java containers between January 28, 2015 and February 16, 2015:

Java Configurations

Name	AMI	Language	Application Server	Web Server
64bit Amazon Linux 2014.09 v1.2.0 running Tomcat 8 Java 8	2014.09	Java 1.8.0_31	Tomcat 8.0.15	Apache 2.2.29
64bit Amazon Linux 2014.09 v1.2.0 running Tomcat 7 Java 7	2014.09	Java 1.7.0.75	Tomcat 7.0.57	Apache 2.2.29
64bit Amazon Linux 2014.09 v1.2.0 running Tomcat 7 Java 6	2014.09	Java 1.6.0_33	Tomcat 7.0.57	Apache 2.2.29
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between November 6, 2014 and January 27, 2015:

Java Configurations

Name	AMI	Language	Application Server	Web Server
64bit Amazon Linux 2014.09 v1.0.0 running Tomcat 8 Java 8	2014.09	Java 1.8.0_25	Tomcat 8	Apache 2.2.29
64bit Amazon Linux 2014.09 v1.0.9[1] running Tomcat 7 Java 7	2014.09	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.09 v1.0.9[1] running Tomcat 7 Java 6	2014.09	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between October 16, 2014 and November 5, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2014.09 v1.0.9[1] running Tomcat 7 Java 7	2014.09	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.09 v1.0.9[1] running Tomcat 7 Java 6	2014.09	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.55	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.9[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.55	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between October 9, 2014 and October 15, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2014.09 v1.0.8 running Tomcat 7 Java 7	2014.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.09 v1.0.8 running Tomcat 7 Java 7	2014.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.09 v1.0.8 running Tomcat 7 Java 6	2014.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.09 v1.0.8 running Tomcat 7 Java 6	2014.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between September 24, 2014 and October 8, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.7[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26

Java Configurations				
64bit Amazon Linux 2014.03 v1.0.7[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.7[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.7[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between June 30, 2014 and September 23, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
64bit Amazon Linux 2014.03 v1.0.4 running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.4 running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between June 5, 2014 and June 29, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.3[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.3[1] running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.3[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.3[1] running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between May 5, 2014 and June 4, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**

Java Configurations				
32bit Amazon Linux 2014.03 v1.0.2 running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.2 running Tomcat 7 Java 7	2014.03	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.03 v1.0.2 running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.03 v1.0.2 running Tomcat 7 Java 6	2014.03	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26

Elastic Beanstalk supports the following configurations for environments created with Java containers between April 7, 2014 and May 4, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2014.02 v1.0.1[1] running Tomcat 7 Java 7	2013.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.02 v1.0.1[1] running Tomcat 7 Java 7	2013.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.02 v1.0.1[1] running Tomcat 7 Java 6	2013.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.02 v1.0.1[1] running Tomcat 7 Java 6	2013.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2013.09 v1.0.1[1] running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 v1.0.1[1] running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2013.09 v1.0.1[1] running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 v1.0.1[1] running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22
64bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22

Elastic Beanstalk supports the following configurations for environments created with Java containers between March 18, 2014 and April 6, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2014.02 running Tomcat 7 Java 7	2013.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.02 running Tomcat 7 Java 7	2013.09	Java 1.7.0.51	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2014.02 running Tomcat 7 Java 6	2013.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2014.02 running Tomcat 7 Java 6	2013.09	Java 1.6.0_24	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2013.09 running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2013.09 running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22
64bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22

Elastic Beanstalk supports the following configurations for environments created with Java containers between November 7, 2013 and March 17, 2014:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux 2013.09 running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 running Tomcat 7 Java 7	2013.09	Java 1.7.0_25	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux 2013.09 running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
64bit Amazon Linux 2013.09 running Tomcat 7 Java 6	2013.09	Java 1.6.0_62	Tomcat 7.0.47	Apache 2.2.26
32bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22

Java Configurations				
64bit Amazon Linux running Tomcat 6	2012.09	Java 1.6.0_24	Tomcat 6.0.35	Apache 2.2.22

Elastic Beanstalk supports the following configurations for environments created with Java containers prior to November 6, 2013:

Java Configurations				
Name	**AMI**	**Language**	**Application Server**	**Web Server**
32bit Amazon Linux running Tomcat 7	2012.09	Java 1.6.0_24	Tomcat 7.0.27	Apache 2.2.22
64bit Amazon Linux running Tomcat 7	2012.09	Java 1.6.0_24	Tomcat 7.0.27	Apache 2.2.22

.NET on Windows Server with IIS Platform History

This page lists the previous versions of AWS Elastic Beanstalk's .NET platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with .NET containers between July 21st, 2015 and August 20th, 2015:

Configuration and *Solution Stack Name*	Framework	Web Server
Windows Server 2012 R2[1] with IIS 8.5 *64bit Windows Server 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012 R2[1] Server Core with IIS 8.5 *64bit Windows Server Core 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012[1] with IIS 8 *64bit Windows Server 2012 running IIS 8*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8
Windows Server 2008 R2[1] with IIS 7.5 *64bit Windows Server 2008 R2 running IIS 7.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin Summary for July 2015

Elastic Beanstalk supports the following container types for environments created with .NET containers between June 12, 2015 and July 21st, 2015:

Configuration and *Solution Stack Name*	Framework	Web Server
Windows Server 2012 R2[1] with IIS 8.5 *64bit Windows Server 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012 R2[1] Server Core with IIS 8.5 *64bit Windows Server Core 2012 R2 running IIS 8.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
Windows Server 2012[1] with IIS 8 *64bit Windows Server 2012 running IIS 8*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8
Windows Server 2008 R2[1] with IIS 7.5 *64bit Windows Server 2008 R2 running IIS 7.5*	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin Summary for June 2015

Elastic Beanstalk supports the following container types for environments created with .NET containers between April 16, 2015 and June 12, 2015:

IIS Configurations			
Name	**AMI**	**Language**	**Web Server**
64bit Windows Server 2012 R2[1] running IIS 8.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
64bit Windows Server Core 2012 R2[1] running IIS 8.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
64bit Windows Server 2012[1] running IIS 8	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8
64bit Windows Server 2008 R2[1] running IIS 7.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin Summary for May 2015

Elastic Beanstalk supports the following container types for environments created with .NET containers between August 6, 2014 and April 16, 2015:

IIS Configurations			
Name	**AMI**	**Language**	**Web Server**
64bit Windows Server 2012 R2[1] running IIS 8.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
64bit Windows Server Core 2012 R2[1] running IIS 8.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8.5
64bit Windows Server 2012[1] running IIS 8	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8
64bit Windows Server 2008 R2[1] running IIS 7.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin MS14-066 - Critical

Elastic Beanstalk supports the following container types for environments created with .NET containers prior to August 6, 2014:

IIS Configurations			
Name	**AMI**	**Language**	**Web Server**
64bit Windows Server 2012[1] running IIS 8	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 8
64bit Windows Server 2008 R2[1] running IIS 7.5	Custom	.NET v4.5 Also supports 4.0, 3.5, 3.0, 2.0, 1.1 and 1.0	IIS 7.5

[1] Microsoft Security Bulletin MS14-066 - Critical

Node.js Platform History

This page lists the previous versions of AWS Elastic Beanstalk's Node.js platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between August 3, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	AMI	Platform	Web Server	Git
Node.js version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Node.js*	2015.03	Node.js 0.12.6 Also supports 0.10.39, 0.10.38, 0.10.31, 0.8.28	Nginx 1.6.2 or Apache 2.4.12	Git 2.1.0

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between July 7, 2015 and August 3, 2015:

Configuration and *Solution Stack Name*	AMI	Platform	Web Server
Node.js version 1.4.4 *64bit Amazon Linux 2015.03 v1.4.4 running Node.js*	2015.03	Node.js 0.12.6 Also supports 0.10.39, 0.10.38, 0.10.31, 0.8.28	Nginx 1.6.2 or Apache 2.4.12

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between June 15, 2015 and July 7, 2015:

Configuration and *Solution Stack Name*	AMI	Platform	Web Server
Node.js version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Node.js*	2015.03	Node.js 0.12.4 Also supports 0.12.2, 0.12.0, 0.10.38, 0.10.31, 0.8.28	Nginx 1.6.2 or Apache 2.4.12

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between May 27, 2015 and June 15, 2015:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2015.03 v1.4.1 running Node.js	2015.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31 0.10.38 0.12.0 0.12.2	Nginx 1.6.2 or Apache 2.4.12

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between April 22, 2015 and May 26, 2015:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.1 running Node.js	2015.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31 0.10.38 0.12.0 0.12.2	Nginx 1.6.2 or Apache 2.4.12

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between April 8, 2015 and April 21, 2015:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.1 running Node.js	2015.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31 0.10.38 0.12.0 0.12.2	Nginx 1.6.2 or Apache 2.4.10

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between March 24, 2015 and April 7, 2015:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**

Node.js Configurations

64bit Amazon Linux 2014.09 v1.2.1 running Node.js	2014.09	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31 0.12.0	Nginx 1.6.2 or Apache 2.4.10

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between February 17, 2015 and March 23, 2015:

Node.js Configurations

Name	AMI	Language	Node.js Version	Web Server
64bit Amazon Linux 2014.09 v1.2.0 running Node.js	2014.09	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.6.2 or Apache 2.4.10

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between January 28, 2015 and February 16, 2015:

Node.js Configurations

Name	AMI	Language	Node.js Version	Web Server
64bit Amazon Linux 2014.09 v1.1.0[1] running Node.js	2014.09	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.6.2 or Apache 2.4.6

Node.js Configurations				
32bit Amazon Linux 2014.03 v1.1.0[1] running Node.js	2014.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.4.7 or Apache 2.4.10
64bit Amazon Linux 2014.03 v1.1.0[1] running Node.js	2014.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.4.7 or Apache 2.4.10

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between October 16, 2014 and January 27, 2015:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2014.09 v1.0.9[1] running Node.js	2014.09	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.6.2 or Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.9[1] running Node.js	2014.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.6.2 or Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.9[1] running Node.js	2014.03	JavaScript	0.8.26 0.8.28 0.10.21 0.10.26 0.10.31	Nginx 1.6.2 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between October 9, 2014 and October 15, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2014.09 v1.0.8 running Node.js	2014.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between September 24, 2014 and October 8, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.7[1] running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.7[1] running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between June 30, 2014 and September 23, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
64bit Amazon Linux 2014.03 v1.0.4 running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between June 5, 2014 and June 29, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.3[1] running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.3[1] running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between May 5, 2014 and June 4, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web-Server**
32bit Amazon Linux 2014.03 v1.0.2 running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.2 running Node.js	2014.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.7 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between April 7, 2014 and May 4, 2014:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web-Server**
32bit Amazon Linux 2014.02 v1.0.1[1] running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.3 or Apache 2.4.6

Node.js Configurations				
64bit Amazon Linux 2014.02 v1.0.1[1] running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.3 or Apache 2.4.6
32bit Amazon Linux 2013.09 v1.0.1[1] running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	Nginx 1.4.3 or Apache 2.4.6
64bit Amazon Linux 2013.09 v1.0.1[1] running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	Nginx 1.4.3 or Apache 2.4.6

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between March 18, 2014 and April 6, 2014:

Node.js Configurations

Name	AMI	Language	Node.js Version	Web Server
32bit Amazon Linux 2014.02 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	Nginx 1.4.3 or Apache 2.4.6

Node.js Configurations

64bit Amazon Linux 2014.02 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.8.26 0.10.10 0.10.21 0.10.26	xnigN 3.4.1 r o ehcapA 6.4.2
32bit Amazon Linux 2013.09 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	xnigN 3.4.1 r o ehcapA 6.4.2
64bit Amazon Linux 2013.09 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	xnigN 3.4.1 r o ehcapA 6.4.2

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between October 29, 2013 and March 17, 2014:

Node.js Configurations

Name	AMI	Language	Node.js Version	beW -vS re
32bit Amazon Linux 2013.09 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	xnigN 3.4.1 r o ehcapA 6.4.2
64bit Amazon Linux 2013.09 running Node.js	2013.09	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10 0.10.21	xnigN 3.4.1 r o ehcapA 6.4.2

Elastic Beanstalk supports the following configurations for environments created with Node.js containers between August 15, 2013 and October 28, 2013:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
32bit Amazon Linux running Node.js	2013.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10	Nginx 1.2.9 or Apache 2.4.4
64bit Amazon Linux running Node.js	2013.03	JavaScript	0.8.6 through 0.8.21 0.8.24 0.10.10	Nginx 1.2.9 or Apache 2.4.4

Elastic Beanstalk supports the following configurations for environments created with Node.js containers prior to August 15, 2013:

Node.js Configurations				
Name	**AMI**	**Language**	**Node.js Version**	**Web Server**
32bit Amazon Linux 2012.09 running Node.js	2012.09	JavaScript	0.8.6 through 0.8.21 0.8.24	Nginx 1.2.6 or Apache 2.4.3
64bit Amazon Linux 2012.09 running Node.js	2012.09	JavaScript	0.8.6 through 0.8.21 0.8.24	Nginx 1.2.6 or Apache 2.4.3

PHP Platform History

This page lists the previous versions of AWS Elastic Beanstalk's PHP platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with PHP containers between August 3, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	AMI	Language	Composer	Web Server
PHP 5.6 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running PHP 5.6*	2015.03	PHP 5.6.9	1.0.0-alpha10	Apache 2.4.12
PHP 5.5 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running PHP 5.5*	2015.03	PHP 5.5.25	1.0.0-alpha10	Apache 2.4.12
PHP 5.4 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running PHP 5.4*	2015.03	PHP 5.4.41	1.0.0-alpha10	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between June 15, 2015 and August 3, 2015:

Configuration and *Solution Stack Name*	AMI	Language	Composer	Web Server
PHP 5.6 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running PHP 5.6*	2015.03	PHP 5.6.9	1.0.0-alpha10	Apache 2.4.12
PHP 5.5 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running PHP 5.5*	2015.03	PHP 5.5.25	1.0.0-alpha10	Apache 2.4.12
PHP 5.4 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running PHP 5.4*	2015.03	PHP 5.4.41	1.0.0-alpha10	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between May 29, 2015 and June 15, 2015:

PHP Configurations				
Name	**AMI**	**Language**	**Web Server**	**Composer**
64bit Amazon Linux 2015.03 v1.4.2 running PHP 5.6	2015.03	PHP 5.6.8	Apache 2.4.12	1.0.0-alpha10
64bit Amazon Linux 2015.03 v1.4.2 running PHP 5.5	2015.03	PHP 5.5.24	Apache 2.4.12	1.0.0-alpha10

PHP Configurations				
64bit Amazon Linux 2015.03 v1.4.2 running PHP 5.4	2015.03	PHP 5.4.40	Apache 2.4.12	-001 - la 01ahp

Elastic Beanstalk supports the following container types for environments created with PHP containers between May 27, 2015 and May 28, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.4.1 running PHP 5.6	2015.03	PHP 5.6.8	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.4.1 running PHP 5.5	2015.03	PHP 5.5.24	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.4.1 running PHP 5.4	2015.03	PHP 5.4.40	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between May 1, 2015 and May 26, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.2 running PHP 5.6	2015.03	PHP 5.6.8	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.3.2 running PHP 5.5	2015.03	PHP 5.5.24	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.3.2 running PHP 5.4	2015.03	PHP 5.4.40	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between April 22, 2015 and April 30, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.1 running PHP 5.5	2015.03	PHP 5.5.22	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.3.1 running PHP 5.4	2015.03	PHP 5.4.38	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between April 8, 2015 and April 21, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.0 running PHP 5.5	2015.03	PHP 5.5.22	Apache 2.4.12
64bit Amazon Linux 2015.03 v1.3.0 running PHP 5.4	2015.03	PHP 5.4.38	Apache 2.4.12

Elastic Beanstalk supports the following container types for environments created with PHP containers between February 17, 2015 and April 7, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2014.09 v1.2.0 running PHP 5.5	2014.09	PHP 5.5.20	Apache 2.4.10
64bit Amazon Linux 2014.09 v1.2.0 running PHP 5.4	2014.09	PHP 5.4.36	Apache 2.4.10

Elastic Beanstalk supports the following container types for environments created with PHP containers between January 28, 2015 and February 16, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2014.09 v1.1.0[1] running PHP 5.5	2014.09	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.09 v1.1.0[1] running PHP 5.4	2014.09	PHP 5.4.20	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.1.0[1] running PHP 5.5	2014.03	PHP 5.5.14	Apache 2.4.10
64bit Amazon Linux 2014.03 v1.1.0[1] running PHP 5.5	2014.03	PHP 5.5.14	Apache 2.4.10
32bit Amazon Linux 2014.03 v1.1.0[1] running PHP 5.4	2014.03	PHP 5.4.30	Apache 2.4.10
64bit Amazon Linux 2014.03 v1.1.0[1] running PHP 5.4	2014.03	PHP 5.4.30	Apache 2.4.10

Elastic Beanstalk supports the following container types for environments created with PHP containers between October 16, 2014 and January 27, 2015:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2014.09 v1.0.9[1] running PHP 5.5	2014.09	PHP 5.5.7	Apache 2.4.6

PHP Container Types			
64bit Amazon Linux 2014.09 v1.0.9[1] running PHP 5.4	2014.09	PHP 5.4.20	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.9[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.9[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.9[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.9[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between October 9, 2014 and October 15, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.09 v1.0.8 running PHP 5.5	2014.09	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.09 v1.0.8 running PHP 5.5	2014.09	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.09 v1.0.8 running PHP 5.4	2014.09	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.09 v1.0.8 running PHP 5.4	2014.09	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between September 24, 2014 and October 8, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.7[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.7[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.7[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.7[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between June 30, 2014 and September 23, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2014.03 v1.0.4 running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.4 running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between June 5, 2014 and June 29, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.3[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.3[1] running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.3[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.3[1] running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between May 5, 2014 and June 4, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.2 running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.2 running PHP 5.5	2014.03	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.03 v1.0.2 running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.03 v1.0.2 running PHP 5.4	2014.03	PHP 5.4.20	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between April 7, 2014 and May 4, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.02 v1.0.1[1] running PHP 5.5	2013.09	PHP 5.5.7	Apache 2.4.6

PHP Container Types			
64bit Amazon Linux 2014.02 v1.0.1[1] running PHP 5.5	2013.09	PHP 5.5.7	Apache 2.4.6
32bit Amazon Linux 2014.02 v1.0.1[1] running PHP 5.4	2013.09	PHP 5.4.20	Apache 2.4.6
64bit Amazon Linux 2014.02 v1.0.1[1] running PHP 5.4	2013.09	PHP 5.4.20	Apache 2.4.6
32bit Amazon Linux 2013.09 v1.0.1[1] running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
64bit Amazon Linux 2013.09 v1.0.1[1] running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
32bit Amazon Linux 2013.09 v1.0.1[1] running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6
64bit Amazon Linux 2013.09 v1.0.1[1] running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6
32bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6
64bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between March 18, 2014 and April 6, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.02 running PHP 5.5	2013.09	PHP 5.5.7	Apache 2.2.26
64bit Amazon Linux 2014.02 running PHP 5.5	2013.09	PHP 5.5.7	Apache 2.2.26
32bit Amazon Linux 2014.02 running PHP 5.4	2013.09	PHP 5.4.20	Apache 2.2.26
64bit Amazon Linux 2014.02 running PHP 5.4	2013.09	PHP 5.4.20	Apache 2.2.26
32bit Amazon Linux 2013.09 running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
64bit Amazon Linux 2013.09 running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
32bit Amazon Linux 2013.09 running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6
64bit Amazon Linux 2013.09 running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6

PHP Container Types			
32bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6
64bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between October 30, 2013 and March 17, 2014:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2013.09 running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
64bit Amazon Linux 2013.09 running PHP 5.5	2013.09	PHP 5.5	Apache 2.4.6
32bit Amazon Linux 2013.09 running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6
64bit Amazon Linux 2013.09 running PHP 5.4	2013.09	PHP 5.4	Apache 2.4.6
32bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6
64bit Amazon Linux running PHP 5.3	2013.03	PHP 5.3	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers between August 29, 2013 and October 29, 2013:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux running PHP 5.4	2013.03	PHP 5.4	Apache 2.4.6
64bit Amazon Linux running PHP 5.4	2013.03	PHP 5.4	Apache 2.4.6

Elastic Beanstalk supports the following container types for environments created with PHP containers prior to August 29, 2013:

PHP Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2012.09 running PHP 5.4	2012.09	PHP 5.4	Apache 2.4.3

PHP Container Types			
64bit Amazon Linux 2012.09 running PHP 5.4	2012.09	PHP 5.4	Apache 2.4.3
32bit Amazon Linux 2012.09 running PHP 5.3	2012.09	PHP 5.3	Apache 2.4.3
64bit Amazon Linux 2012.09 running PHP 5.3	2012.09	PHP 5.3	Apache 2.4.3

Python Platform History

This page lists the previous versions of AWS Elastic Beanstalk's Python platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with Python containers between August 3, 2015 and August 11, 2015:

Configuration and ***Solution Stack Name***	**AMI**	**Language**	**Web Server**
Python 3.4 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Python 3.4*	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
Python 2.7 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Python 2.7*	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5
Python 2.6 version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Python*	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between June 15, 2015 and August 3, 2015:

Configuration and ***Solution Stack Name***	**AMI**	**Language**	**Web Server**
Python 3.4 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Python 3.4*	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
Python 2.7 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Python 2.7*	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5

Configuration and *Solution Stack Name*	AMI	Language	Web Server
Python 2.6 version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Python*	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between May 27, 2015 and June 15, 2015:

Python Configurations			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.4.1 running Python 3.4	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.4.1 running Python 2.7	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.4.1 running Python 2.6	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between April 22, 2015 and May 26, 2015:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.1 running Python 3.4	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.3.1 running Python 2.7	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.3.1 running Python 2.6	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between April 8, 2015 and April 21, 2015:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**
64bit Amazon Linux 2015.03 v1.3.0 running Python 3.4	2015.03	Python 3.4.3	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.3.0 running Python 2.7	2015.03	Python 2.7.9	Apache 2.4.12 with mod_wsgi 3.5
64bit Amazon Linux 2015.03 v1.3.0 running Python 2.6	2015.03	Python 2.6.9	Apache 2.4.12 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between February 17, 2015 and April 7, 2015:

Python Container Types

Name	AMI	Language	Web Server
64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7	2014.09	Python 2.7.8	Apache 2.4.10 with mod_wsgi 3.5
64bit Amazon Linux 2014.09 v1.2.0 running Python 2.6	2014.09	Python 2.6.9	Apache 2.4.10 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between January 28, 2015 and February 16, 2015:

Python Container Types

Name	AMI	Language	Web Server
64bit Amazon Linux 2014.09 v1.1.0[1] running Python 2.7	2014.09	Python 2.7.5	Apache 2.4.10 with mod_wsgi 3.5
64bit Amazon Linux 2014.09 v1.1.0[1] running Python	2014.09	Python 2.6.9	Apache 2.4.10 with mod_wsgi 3.5
32bit Amazon Linux 2014.03 v1.1.0[1] running Python 2.7	2014.03	Python 2.7.5	Apache 2.4.10 with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.1.0[1] running Python 2.7	2014.03	Python 2.7.5	Apache 2.4.10 with mod_wsgi 3.2
32bit Amazon Linux 2014.03 v1.1.0[1] running Python	2014.03	Python 2.6.9	Apache 2.4.10 with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.1.0[1] running Python	2014.03	Python 2.6.9	Apache 2.4.10 with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between October 31, 2014 and January 27, 2015:

Python Container Types

Name	AMI	Language	Web Server
64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7	2014.09	Python 2.7.5	Apache 2.4.10 with mod_wsgi 3.5
64bit Amazon Linux 2014.09 v1.0.9 running Python	2014.09	Python 2.6.9	Apache 2.4.10 with mod_wsgi 3.5

Elastic Beanstalk supports the following container types for environments created with Python containers between October 16, 2014 and October 30, 2014:

Python Container Types

Name	AMI	Language	Web Server

Python Container Types

32bit Amazon Linux 2014.03 v1.0.9[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.9[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2014.03 v1.0.9[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.9[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between September 24, 2014 and October 15, 2014:

Python Container Types

Name	AMI	Language	Web Server
32bit Amazon Linux 2014.03 v1.0.7[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.7[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2014.03 v1.0.7[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.7[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between June 30, 2014 and September 23, 2014:

Python Container Types

Name	AMI	Language	Web Server
64bit Amazon Linux 2014.03 v1.0.4 running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.4 running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between June 5, 2014 and June 29, 2014:

Python Container Types

Name	AMI	Language	Web Server
32bit Amazon Linux 2014.03 v1.0.3[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2

Python Container Types			
64bit Amazon Linux 2014.03 v1.0.3[1] running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2014.03 v1.0.3[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.3[1] running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between May 5, 2014 and June 4, 2014:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2014.03 v1.0.2 running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.2 running Python 2.7	2014.03	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2014.03 v1.0.2 running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2014.03 v1.0.2 running Python	2014.03	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between April 7, 2014 and May 4, 2014:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux 2013.09 v1.0.1[1] running Python 2.7	2013.09	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2013.09 v1.0.1[1] running Python 2.7	2013.09	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2013.09 v1.0.1[1] running Python	2013.09	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2013.09 v1.0.1[1] running Python	2013.09	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers between November 7, 2013 and April 6, 2014:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**

Python Container Types			
32bit Amazon Linux 2013.09 running Python 2.7	2013.09	Python 2.7	Apache with mod_wsgi 3.2
64bit Amazon Linux 2013.09 running Python 2.7	2013.09	Python 2.7	Apache with mod_wsgi 3.2
32bit Amazon Linux 2013.09 running Python	2013.09	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux 2013.09 running Python	2013.09	Python 2.6	Apache with mod_wsgi 3.2

Elastic Beanstalk supports the following container types for environments created with Python containers prior to November 7, 2013:

Python Container Types			
Name	**AMI**	**Language**	**Web Server**
32bit Amazon Linux running Python	2012.09	Python 2.6	Apache with mod_wsgi 3.2
64bit Amazon Linux running Python	2012.09	Python 2.6	Apache with mod_wsgi 3.2

Ruby Platform History

This page lists the previous versions of AWS Elastic Beanstalk's Ruby platforms and the dates that each version was current. Platform versions that you have used to launch an environment remain available even after they are no longer current.

See the Supported Platforms (p. 24) page for information on the latest version of each platform supported by Elastic Beanstalk.

Elastic Beanstalk supports the following container types for environments created with Ruby containers between August 3, 2015 and August 11, 2015:

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Ruby 2.2 with Puma version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.2 (Puma)*	2015.03	Ruby 2.2.2	Puma 2.10.2	Nginx 1.6.2
Ruby 2.2 with Passenger version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.2 (Passenger Standalone)*	2015.03	Ruby 2.2.2	Passenger 4.0.59	Nginx 1.6.2

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Ruby 2.1 with Puma version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.1 (Puma)*	2015.03	Ruby 2.1.5-p273	Puma 2.10.2	Nginx 1.6.2
Ruby 2.1 with Passenger version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.1 (Passenger Standalone)*	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59	Nginx 1.6.2
Ruby 2.0 with Puma version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.0 (Puma)*	2015.03	Ruby 2.0.0-p598	Puma 2.10.2	Nginx 1.6.2
Ruby 2.0 with Passenger version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 2.0 (Passenger Standalone)*	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59	Nginx 1.6.2
Ruby 1.9 with Passenger version 1.4.6 *64bit Amazon Linux 2015.03 v1.4.6 running Ruby 1.9.3*	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59	Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between June 15, 2015 and August 3, 2015:

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Ruby 2.2 with Puma version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.2 (Puma)*	2015.03	Ruby 2.2.2	Puma 2.10.2	Nginx 1.6.2

Configuration and *Solution Stack Name*	AMI	Language	Application Server	Web Server
Ruby 2.2 with Passenger version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.2 (Passenger Standalone)*	2015.03	Ruby 2.2.2	Passenger 4.0.59	Nginx 1.6.2
Ruby 2.1 with Puma version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.1 (Puma)*	2015.03	Ruby 2.1.5-p273	Puma 2.10.2	Nginx 1.6.2
Ruby 2.1 with Passenger version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.1 (Passenger Standalone)*	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59	Nginx 1.6.2
Ruby 2.0 with Puma version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.0 (Puma)*	2015.03	Ruby 2.0.0-p598	Puma 2.10.2	Nginx 1.6.2
Ruby 2.0 with Passenger version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 2.0 (Passenger Standalone)*	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59	Nginx 1.6.2
Ruby 1.9 with Passenger version 1.4.3 *64bit Amazon Linux 2015.03 v1.4.3 running Ruby 1.9.3*	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59	Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between May 27, 2015 and June 15, 2015:

Ruby Configurations			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.2 (Puma)	2015.03	Ruby 2.2.2	Puma 2.10.2 and Nginx 1.6.2

Ruby Configurations

64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.2 (Passenger Standalone)	2015.03	Ruby 2.2.2	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.1 (Puma)	2015.03	Ruby 2.1.5-p273	Puma 2.10.2 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.1 (Passenger Standalone)	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.0 (Puma)	2015.03	Ruby 2.0.0-p598	Puma 2.10.2 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 2.0 (Passenger Standalone)	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.4.1 running Ruby 1.9.3	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59 and Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between April 22, 2015 and May 26, 2015:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.2 (Puma)	2015.03	Ruby 2.2.2	Puma 2.10.2 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.2 (Passenger Standalone)	2015.03	Ruby 2.2.2	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.1 (Puma)	2015.03	Ruby 2.1.5-p273	Puma 2.10.2 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.1 (Passenger Standalone)	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.0 (Puma)	2015.03	Ruby 2.0.0-p598	Puma 2.10.2 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.0 (Passenger Standalone)	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 1.9.3	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59 and Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between April 8, 2015 and April 21, 2015:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.2 (Puma)	2015.03	Ruby 2.2.2	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.2 (Passenger Standalone)	2015.03	Ruby 2.2.2	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.1 (Puma)	2015.03	Ruby 2.1.5-p273	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.1 (Passenger Standalone)	2015.03	Ruby 2.1.5-p273	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.0 (Puma)	2015.03	Ruby 2.0.0-p598	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 2.0 (Passenger Standalone)	2015.03	Ruby 2.0.0-p598	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2015.03 v1.3.1 running Ruby 1.9.3	2015.03	Ruby 1.9.3-p551	Passenger 4.0.59 and Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between March 24, 2015 and April 7, 2015:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.09 v1.2.1 running Ruby 2.2 (Puma)	2014.09	Ruby 2.2.0	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.1 running Ruby 2.2 (Passenger Standalone)	2014.09	Ruby 2.2.0	Passenger 4.0.59 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Puma)	2014.09	Ruby 2.1.5-p273	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Passenger Standalone)	2014.09	Ruby 2.1.5-p273	Passenger 4.0.53 and Nginx 1.6.2

Ruby Container Types			
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.0 (Puma)	2014.09	Ruby 2.0.0-p598	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.0 (Passenger Standalone)	2014.09	Ruby 2.0.0-p598	Passenger 4.0.53 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 1.9.3	2014.09	Ruby 1.9.3-p551	Passenger 4.0.53 and Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between February 17, 2015 and March 23, 2015:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Puma)	2014.09	Ruby 2.1.5-p273	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.1 (Passenger Standalone)	2014.09	Ruby 2.1.5-p273	Passenger 4.0.53 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.0 (Puma)	2014.09	Ruby 2.0.0-p598	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 2.0 (Passenger Standalone)	2014.09	Ruby 2.0.0-p598	Passenger 4.0.53 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.2.0 running Ruby 1.9.3	2014.09	Ruby 1.9.3-p551	Passenger 4.0.53 and Nginx 1.6.2

Elastic Beanstalk supports the following container types for environments created with Ruby containers between January 28, 2015 and February 16, 2015:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.09 v1.1.0[1] running Ruby 2.1 (Puma)	2014.09	Ruby 2.1.4	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.1.0[1] running Ruby 2.1 (Passenger Standalone)	2014.09	Ruby 2.1.4	Passenger 4.0.53
64bit Amazon Linux 2014.09 v1.1.0[1] running Ruby 2.0 (Puma)	2014.09	Ruby 2.0.0-p594	Puma 2.9.1 and Nginx 1.6.2

Ruby Container Types			
64bit Amazon Linux 2014.09 v1.1.0[1] running Ruby 2.0 (Passenger Standalone)	2014.09	Ruby 2.0.0-p594	Passenger 4.0.53
64bit Amazon Linux 2014.09 v1.1.0[1] running Ruby 1.9.3	2014.09	Ruby 1.9.3-p550	Passenger 4.0.53
64bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 2.1 (Puma)	2014.03	Ruby 2.1.2-p95	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 2.1 (Passenger Standalone)	2014.03	Ruby 2.1.2	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0-p481	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
32bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.1.0[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between October 31, 2014 and January 27, 2015:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.09 v1.0.9 running Ruby 2.1 (Puma)	2014.09	Ruby 2.1.4	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.0.9 running Ruby 2.1 (Passenger Standalone)	2014.09	Ruby 2.1.4	Passenger 4.0.53
64bit Amazon Linux 2014.09 v1.0.9 running Ruby 2.0 (Puma)	2014.09	Ruby 2.0.0-p594	Puma 2.9.1 and Nginx 1.6.2
64bit Amazon Linux 2014.09 v1.0.9 running Ruby 2.0 (Passenger Standalone)	2014.09	Ruby 2.0.0-p594	Passenger 4.0.53
64bit Amazon Linux 2014.09 v1.0.9 running Ruby 1.9.3	2014.09	Ruby 1.9.3-p550	Passenger 4.0.53

Elastic Beanstalk supports the following container types for environments created with Ruby containers between October 16, 2014 and October 30, 2014:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 2.1 (Puma)	2014.03	Ruby 2.1.2	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 2.1 (Passenger Standalone)	2014.03	Ruby 2.1.2	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
32bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.9[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between September 24, 2014 and October 15, 2014:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 2.1 (Puma)	2014.03	Ruby 2.1.2	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 2.1 (Passenger Standalone)	2014.03	Ruby 2.1.2	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
32bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.7[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between August 14, 2014 and September 23, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.0 running Ruby 2.1 (Puma)	2014.03	Ruby 2.1.2	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.0 running Ruby 2.1 (Passenger Standalone)	2014.03	Ruby 2.1.2	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.5 running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.4 running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.4 running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between June 30, 2014 and August 13, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.5 running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.4 running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.4 running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between June 5, 2014 and June 29, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.4[1] running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.3[1] running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37

Ruby Container Types

32bit Amazon Linux 2014.03 v1.0.3[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.3[1] running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between May 14, 2014 and June 4, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.3 running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7

Elastic Beanstalk supports the following container types for environments created with Ruby containers between May 5, 2014 and May 13, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.2 running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.2 running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
32bit Amazon Linux 2014.03 v1.0.2 running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.03 v1.0.2 running Ruby 1.9.3	2014.03	Ruby 1.9.3	Passenger 4.0.37

Elastic Beanstalk supports the following container types for environments created with Ruby containers between April 7, 2014 and May 4, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
64bit Amazon Linux 2014.03 v1.0.1[2] running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 v1.0.1[2] running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37

Ruby Container Types			
32bit Amazon Linux 2014.02 v1.0.1[1] running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.02 v1.0.1[1] running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
32bit Amazon Linux 2014.02 v1.0.1[1] running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
64bit Amazon Linux 2014.02 v1.0.1[1] running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
32bit Amazon Linux 2013.09 v1.0.1[1] running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
64bit Amazon Linux 2013.09 v1.0.1[1] running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
32bit Amazon Linux 2013.09 v1.0.1[1] running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20
64bit Amazon Linux 2013.09 v1.0.1[1] running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20

Elastic Beanstalk supports the following container types for environments created with Ruby containers between April 2, 2014 and April 6, 2014:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
64bit Amazon Linux 2014.03 running Ruby 2.0 (Puma)	2014.03	Ruby 2.0.0	Puma 2.8.1 and Nginx 1.4.7
64bit Amazon Linux 2014.03 running Ruby 2.0 (Passenger Standalone)	2014.03	Ruby 2.0.0	Passenger 4.0.37
32bit Amazon Linux 2014.02 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.02 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
32bit Amazon Linux 2014.02 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
64bit Amazon Linux 2014.02 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
32bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
64bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
32bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20

Ruby Container Types

64bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20

Elastic Beanstalk supports the following container types for environments created with Ruby containers between March 18, 2014 and April 1, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
32bit Amazon Linux 2014.02 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
64bit Amazon Linux 2014.02 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.37
32bit Amazon Linux 2014.02 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
64bit Amazon Linux 2014.02 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.37
32bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
64bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
32bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20
64bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20

Elastic Beanstalk supports the following container types for environments created with Ruby containers between November 9, 2013 and March 17, 2014:

Ruby Container Types

Name	AMI	Language	Application/Web Server
32bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
64bit Amazon Linux 2013.09 running Ruby 1.9.3	2013.09	Ruby 1.9.3	Passenger 4.0.20
32bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20
64bit Amazon Linux 2013.09 running Ruby 1.8.7	2013.09	Ruby 1.8.7	Passenger 4.0.20

Elastic Beanstalk supports the following container types for environments created with Ruby containers prior to November 9, 2013:

Ruby Container Types			
Name	**AMI**	**Language**	**Application/Web Server**
32bit Amazon Linux running Ruby 1.9.3	2012.09	Ruby 1.9.3	Passenger 3.0.17
64bit Amazon Linux running Ruby 1.9.3	2012.09	Ruby 1.9.3	Passenger 3.0.17
32bit Amazon Linux running Ruby 1.8.7	2012.09	Ruby 1.8.7	Passenger 3.0.17
64bit Amazon Linux running Ruby 1.8.7	2012.09	Ruby 1.8.7	Passenger 3.0.17

Using Amazon RDS with PHP (Legacy Container Types)

Amazon Relational Database Service (Amazon RDS) lets you quickly and easily provision and maintain a MySQL Server instance in the cloud. This topic discusses how you can use Amazon RDS and PHP with your Elastic Beanstalk application.

> **Note**
> For more information on AWS storage options, go to Storage Options in the AWS Cloud.

To use Amazon RDS and PHP from your Elastic Beanstalk application

1. Create an Amazon RDS DB Instance. For instructions on how to do this, go to the Amazon RDS User Guide.
2. Configure your Amazon RDS DB Security Group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. If you plan to use PDO, install the PDO drivers. For more information, go to http://www.php.net/manual/pdo.installation.php.
4. Establish a database connection in your PHP code using your Amazon RDS DB Instance's public DNS name, port number, and (optionally) database name and login credentials. The following examples show examples that would connect to the employee database on an RDS Instance at mydbinstance.abcdefghijkl.us-east-1.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

Example 1. Example using PDO to connect to an RDS database

```
<?php
$dsn = 'mysql:host=mydbinstance.abcdefghijkl.us-east-1.rds.amazon
aws.com;port=3306;dbname=mydb';
$username = 'sa';
$password = 'mypassword';

$dbh = new PDO($dsn, $username, $password);
?>
```

For more information about constructing a connection string using PDO, go to http://us2.php.net/manual/en/pdo.construct.php.

Example 2. Example using mysql_connect() to connect to an RDS database

```
<?php
$dbhost = 'mydbinstance.abcdefghijkl.us-east-1.rds.amazonaws.com:3306';
$username = 'sa';
$password = 'mypassword';
$dbname = 'mydb';

$link = mysql_connect($dbhost, $username, $password, $dbname);
mysql_select_db($dbname);
?>
```

For more information on using `mysql_connect()`, go to http://www.phpbuilder.com/manual/function.mysql-connect.php.

Example 3. Example using mysqli_connect() to connect to an RDS database

```
$link = mysqli_connect('mydbinstance.abcdefghijkl.us-east-1.rds.amazon
aws.com', 'sa', 'mypassword', 'mydb', 3306);
```

For more information on using `mysqli_connect()`, go to http://www.phpbuilder.com/manual/function.mysqli-connect.php.

5. Deploy your application to Elastic Beanstalk. For information on how to deploy your application using Elastic Beanstalk and the AWS Management Console, see Getting Started Using Elastic Beanstalk (p. 4). For information on how to deploy your application using AWS DevTools, see Deploying Elastic Beanstalk Applications in PHP (p. 706).

Using Amazon RDS and MySQL Connector/J (Legacy Container Types)

Amazon Relational Database Service (Amazon RDS) lets you quickly and easily provision and maintain a MySQL Server instance in the cloud. This topic discusses how you can use Amazon RDS and the MySQL Connector/J with your Elastic Beanstalk application.

To use Amazon RDS from your Elastic Beanstalk application, you need to do the following:

- Create an Amazon RDS DB Instance.

- Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application.
- Create a JDBC connection string using your Amazon RDS DB instance's public DNS name and configure your Elastic Beanstalk environment to pass the string to your Elastic Beanstalk application as an environment variable.
- Download and install MySQL Connector/J.
- Retrieve the JDBC connection string from the environment property passed to your server instance from Elastic Beanstalk and use MySQL Connector/J to access your Amazon RDS database.

To use Amazon RDS with MySQL Connector/J from your Elastic Beanstalk application

1. Create an Amazon RDS DB Instance. For instructions on how to do this, go to the Amazon Relational Database Service User Guide.
2. Configure your Amazon RDS DB security group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your Amazon EC2 security group using AWS Toolkit for Eclipse, see Amazon EC2 Security Groups (p. 601). For instructions on how to find the name of your Amazon EC2 security group using the AWS Management Console, see Amazon EC2 Security Groups (p. 201). For more information, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Download and install MySQL Connector/J for your development environment. For download and installation instructions, go to http://dev.mysql.com/downloads/connector/j.
4. Create a JDBC connection string using your Amazon RDS DB Instance's public DNS name, port number, and (optionally) database name and login credentials. The following example shows a JDBC connection string that would connect to the employees database on an RDS instance at mydbinstance.abcdefghijkl.us-east-1.rds.amazonaws.com using port 3306, with the user name "sa" and the password "mypassword".

   ```
   jdbc:mysql://mydbinstance.abcdefghijkl.us-east-1.rds.amazonaws.com:3306/em
   ployees?user=sa&password=mypassword
   ```

5. Configure your Elastic Beanstalk environment to pass the string to your Elastic Beanstalk application as an environment property. For instructions on how to do this, go to Using Custom Environment Properties with Elastic Beanstalk (p. 589).
6. Retrieve the JDBC connection string from the environment property passed to your server instance from Elastic Beanstalk and use MySQL Connector/J to access your Amazon RDS database. The following code example shows how to retrieve the JDBC_CONNECTION_STRING custom environment property from a Java Server Page (JSP).

   ```
   <p>
       The JDBC_CONNECTION_STRING environment variable is:
       <%= System.getProperty("JDBC_CONNECTION_STRING") %>
   </p>
   ```

   For more information on getting started using the MySQL Connector/J to access your MySQL database, go to http://dev.mysql.com/doc/connector-j/en/index.html.
7. Copy the MySQL Connector/J JAR file into your Elastic Beanstalk application's WEB-INF/lib directory.
8. Deploy your application to Elastic Beanstalk. For information on how to deploy your application using Elastic Beanstalk and the AWS Management Console, see Getting Started Using Elastic

Beanstalk (p. 4). For information on how to deploy your application using Eclipse, go to Getting Started with Elastic Beanstalk Deployment in Eclipse.