Amazon EC2 Container Service

Developer Guide

API Version 2014-11-13

Amazon EC2 Container Service: Developer Guide

Copyright © 2016 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Table of Contents

What is Amazon EC2 Container Service?

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances. Amazon ECS lets you launch and stop container-based applications with simple API calls, allows you to get the state of your cluster from a centralized service, and gives you access to many familiar Amazon EC2 features.

You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs, isolation policies, and availability requirements. Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

Amazon ECS can be used to create a consistent deployment and build experience, manage and scale batch and Extract-Transform-Load (ETL) workloads, and build sophisticated application architectures on a microservices model. For more information about Amazon ECS use cases and scenarios, see Container Use Cases.

AWS Elastic Beanstalk can also be used to rapidly develop, test, and deploy Docker containers in conjunction with other components of your application infrastructure; however, using Amazon ECS directly provides more fine-grained control and access to a wider set of use cases. For more information, see the AWS Elastic Beanstalk Developer Guide.

Features of Amazon ECS

Amazon ECS is a regional service that simplifies running application containers in a highly available manner across multiple Availability Zones within a region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can define task definitions and services that specify which Docker container images to run across your clusters. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.
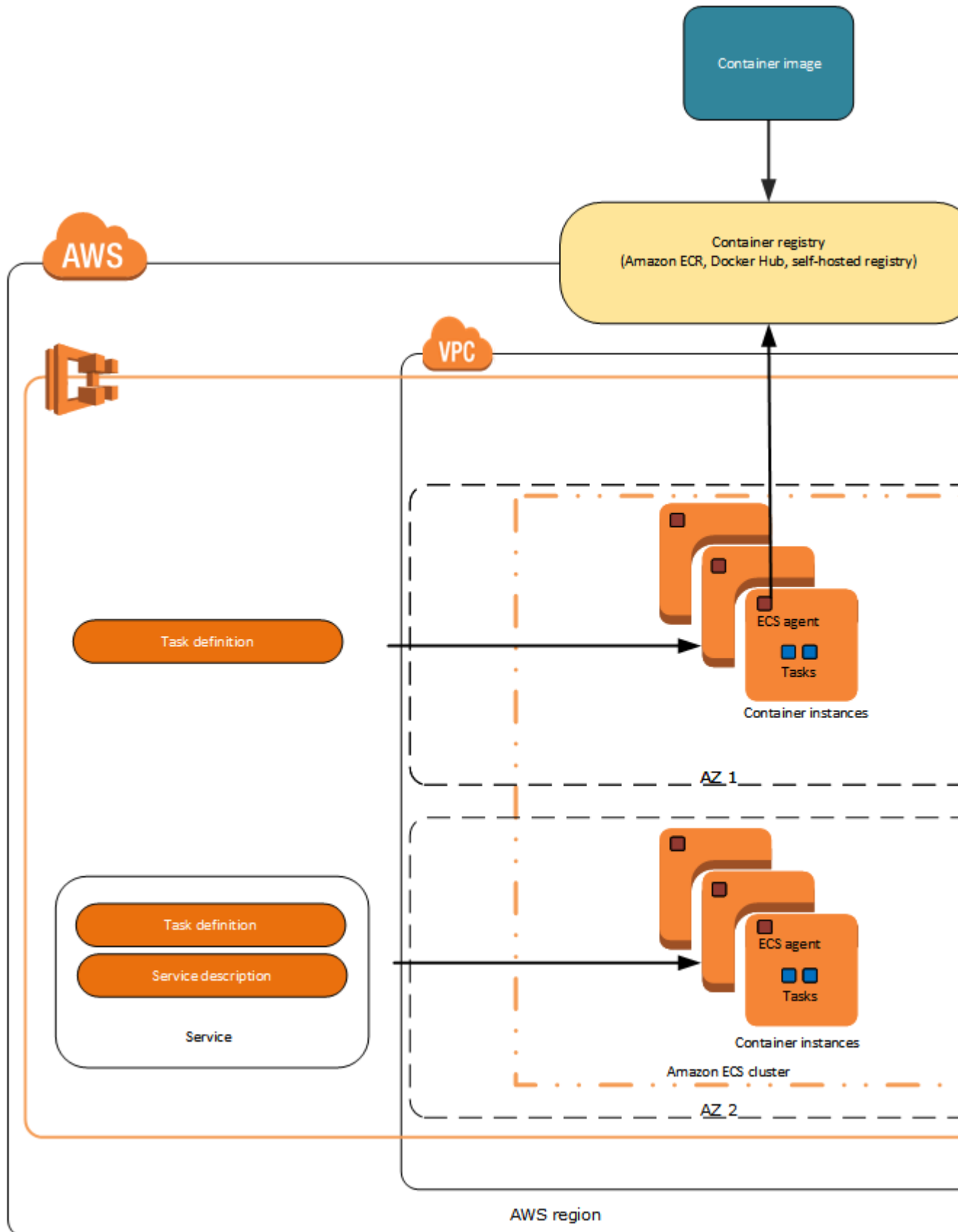
The following sections dive into these individual elements of the Amazon ECS architecture in more detail.

Containers and Images

To deploy applications on Amazon ECS, your application components must be architected to run in *containers*. A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc. Containers are created from a read-only template called an *image*.

Images are typically built from a Dockerfile, a plain text file that specifies all of the components that are included in the container. These images are then stored in a *registry* from which they can be downloaded and run on your container instances. For more information about container technology, see Docker Basics (p. 13).
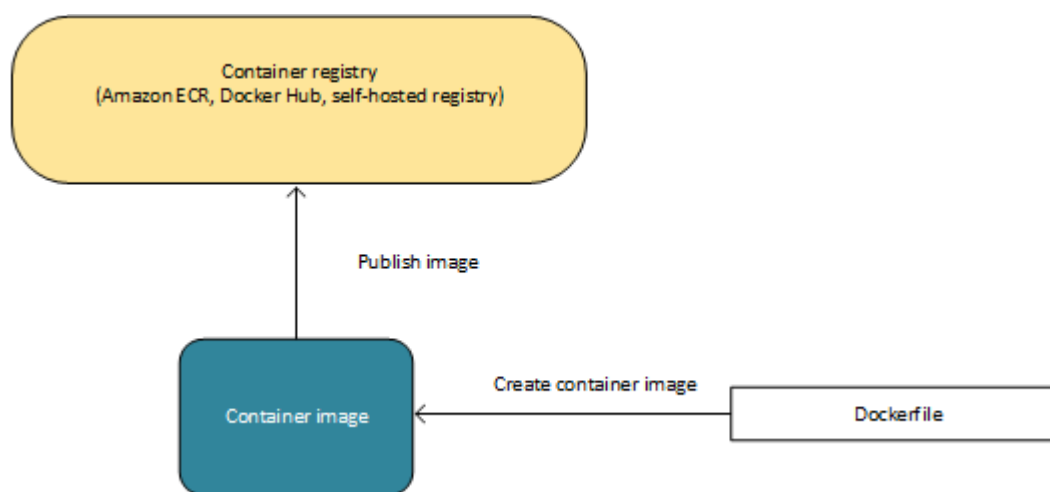
Task Definitions

To prepare your application to run on Amazon ECS, you create a *task definition*. The task definition is a text file in JSON format that describes one or more containers that form your application. It can be thought of as a blueprint for your application. Task definitions specify various parameters for your application, such as which containers to use and the repositories in which they are located, which ports should be opened on the container instance for your application, and what data volumes should be used with the containers in the task. For more information about creating task definitions, see Amazon ECS Task Definitions (p. 82).

The following is an example of a simple task definition containing a single container that runs an Nginx web server. For a more extended example demonstrating the use of multiple containers in a task definition, see Example Task Definitions (p. 112).

```
{
    "family": "webserver",
    "containerDefinitions": [
    {
        "name": "web",
        "image": "nginx",
        "cpu": 99,
        "memory": 100,
```

```
            "portMappings": [{
                    "containerPort": 80,
                    "hostPort": 80
            }]
    }]
}
```

Tasks and Scheduling

A *task* is the instantiation of a task definition on a container instance within your cluster. After you have created a task definition for your application within Amazon ECS, you can specify the number of tasks that will run on your cluster.

The Amazon ECS task scheduler is responsible for placing tasks on container instances. There are several different scheduling options available. For example, you can define a *service* that runs and maintains a specified number of tasks simultaneously. For more information about the different scheduling options available, see Scheduling Amazon ECS Tasks (p. 117).
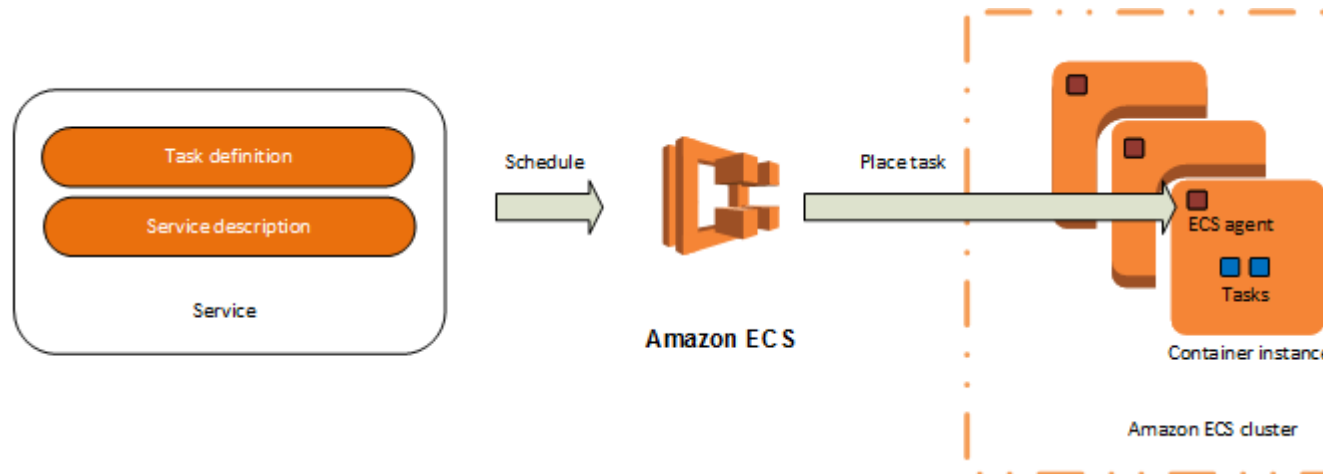
Clusters

When you run tasks using Amazon ECS, you place them on a *cluster*, which is a logical grouping of EC2 instances. Amazon ECS downloads your container images from a registry that you specify, and runs those images on the container instances within your cluster.

For more information about creating clusters, see Amazon ECS Clusters (p. 27). For more information about creating container instances, see Amazon ECS Container Instances (p. 32).

Container Agent

The *container agent* runs on each instance within an Amazon ECS cluster. It sends information about the instance's current running tasks and resource utilization to Amazon ECS, and starts and stops tasks whenever it receives a request from Amazon ECS. For more information, see Amazon ECS Container Agent (p. 58).
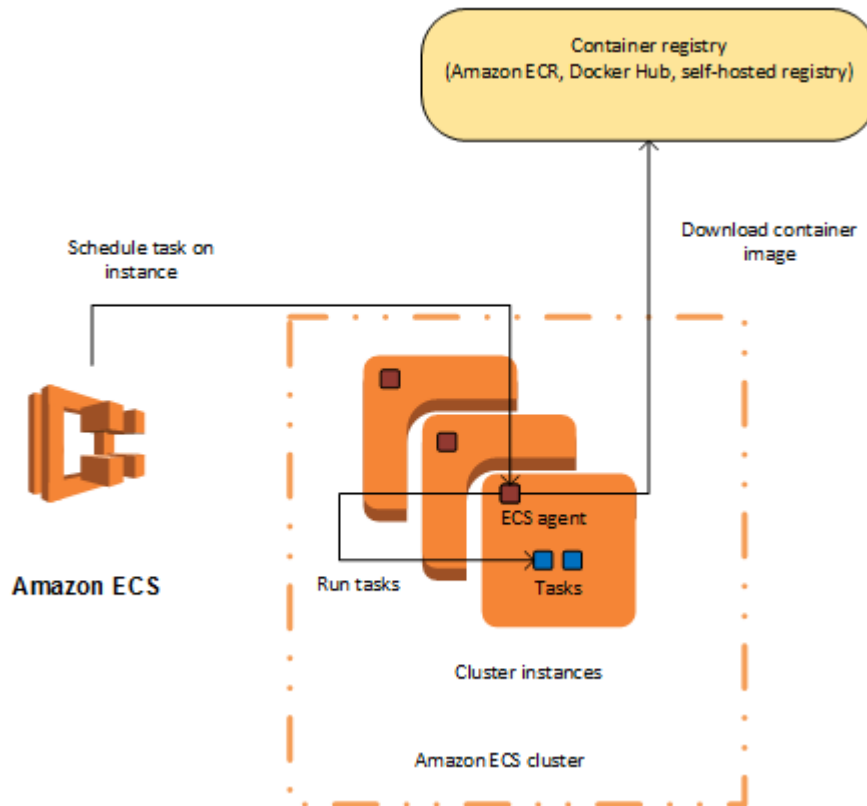
How to Get Started with Amazon ECS

If you are using Amazon ECS for the first time, the AWS Management Console for Amazon ECS provides a first-run wizard that steps you through defining a task definition for a web server, configuring a service, and launching your first cluster. The first-run wizard is highly recommended for users who have no prior experience with Amazon ECS. For more information, see the Getting Started with Amazon ECS (p. 20) tutorial.

Alternatively, you can install the AWS Command Line Interface (AWS CLI) to use Amazon ECS. For more information, see Setting Up with Amazon ECS (p. 8).

Related Services

Amazon ECS can be used in conjunction with the following AWS services:

AWS Identity and Access Management
IAM is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). In Amazon ECS, IAM can be used to control access at the container instance level using IAM roles, and at the task level using IAM task roles. For more information, see Amazon ECS IAM Policies, Roles, and Permissions (p. 168)

Auto Scaling
Auto Scaling is a web service that enables you to automatically launch or terminate EC2 instances based on user-defined policies, health status checks, and schedules. You can use Auto Scaling to scale out and scale in the container instances within a cluster in response to a number of

metrics. For more information, see Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 162).

Elastic Load Balancing

Elastic Load Balancing automatically distributes incoming application traffic across multiple EC2 instances in the cloud. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic. You can use Elastic Load Balancing to create an endpoint that balances traffic across services in a cluster. For more information, see Service Load Balancing (p. 124).

Amazon EC2 Container Registry

Amazon ECR is a managed AWS Docker registry service that is secure, scalable, and reliable. Amazon ECR supports private Docker repositories with resource-based permissions using IAM so that specific users or EC2 instances can access repositories and images. Developers can use the Docker CLI to push, pull, and manage images. For more information, see the Amazon EC2 Container Registry User Guide.

AWS CloudFormation

AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion. You can define clusters, task definitions, and services as entities in an AWS CloudFormation script. For more information, see AWS CloudFormation Template Reference.

Accessing Amazon ECS

You can work with Amazon ECS in any of the following ways:

AWS Management Console

The console is a browser-based interface to manage Amazon ECS resources. For a tutorial that guides you through the console, see Getting Started with Amazon ECS (p. 20).

AWS command line tools

You can use the AWS command line tools to issue commands at your system's command line to perform Amazon ECS and AWS tasks; this can be faster and more convenient than using the console. The command line tools are also useful for building scripts that perform AWS tasks.

AWS provides two sets of command line tools: the AWS Command Line Interface (AWS CLI) and the AWS Tools for Windows PowerShell. For more information, , see the AWS Command Line Interface User Guideand AWS Tools for Windows PowerShell User Guide.

Amazon ECS CLI

In addition to using the AWS CLI to access Amazon ECS resources, you can use the Amazon ECS CLI, which provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment using Docker Compose. For more information, see Using the Amazon ECS Command Line Interface (p. 199).

AWS SDKs

We also provide SDKs that enable you to access Amazon ECS from a variety of programming languages. The SDKs automatically take care of tasks such as:

- Cryptographically signing your service requests
- Retrying requests
- Handling error responses

For more information about available SDKs, see Tools for Amazon Web Services.

Pricing

There is no additional charge for using Amazon ECS beyond the underlying AWS resources used to host your applications. For more information, see AWS ECS Pricing.

Setting Up with Amazon ECS

If you've already signed up for Amazon Web Services (AWS) and have been using Amazon Elastic Compute Cloud (Amazon EC2), you are close to being able to use Amazon ECS. The set up process for the two services is very similar, as Amazon ECS uses EC2 instances in the clusters. The following guide prepares you for launching your first cluster using either the Amazon ECS first-run wizard or the Amazon ECS Command Line Interface (CLI).

Note
Because Amazon ECS uses many components of Amazon EC2, you use the Amazon EC2 console for many of these steps.

Complete the following tasks to get set up for Amazon ECS. If you have already completed any of these steps, you may skip them and move on to installing the custom AWS CLI.

1. Sign Up for AWS (p. 8)
2. Create an IAM User (p. 9)
3. Create an IAM Role for your Container Instances and Services (p. 10)
4. Create a Key Pair (p. 10)
5. (Optional) Install the Amazon ECS Command Line Interface (CLI) (p. 12)

Sign Up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services, including Amazon EC2 and Amazon ECS. You are charged only for the services that you use.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open http://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account number, because you'll need it for the next task.

Create an IAM User

Services in AWS, such as Amazon EC2 and Amazon ECS, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

To create an IAM user for yourself and add the user to an Administrators group

1. Sign in to the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Users**, and then choose **Add user**.

3. For **User name**, type a user name, such as `Administrator`. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 64 characters in length.

4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to select a new password the next time the user signs in.

5. Choose **Next: Permissions**.

6. On the **Set permissions for user** page, choose **Add user to group**.

7. Choose **Create group**.

8. In the **Create group** dialog box, type the name for the new group. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters in length.

9. For **Filter**, choose **Job function**.

10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.

12. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose Add permissions.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies for Administering AWS Resources.

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where *your_aws_account_id* is your AWS account number without the hyphens (for example, if your AWS account number is `1234-5678-9012`, your AWS account ID is `123456789012`):

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your_user_name @ your_aws_account_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, choose **Create Account Alias** and enter an alias, such as your company name. To sign in after you create an account alias, use the following URL:

```
https://your_account_alias.signin.aws.amazon.com/console/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

For more information about IAM, see the AWS Identity and Access Management User Guide.

Create an IAM Role for your Container Instances and Services

Before the Amazon ECS agent can register container instance into a cluster, the agent must know which account credentials to use. You can create an IAM role that allows the agent to know which account it should register the container instance with. When you launch an instance with the Amazon ECS-optimized AMI provided by Amazon using this role, the agent automatically registers the container instance into your default cluster.

The Amazon ECS container agent also makes calls to the Amazon EC2 and Elastic Load Balancing APIs on your behalf, so container instances can be registered and deregistered with load balancers. Before you can attach a load balancer to an Amazon ECS service, you must create an IAM role for your services to use before you start them. This requirement applies to any Amazon ECS service that you plan to use with a load balancer.

> **Note**
> The Amazon ECS instance and service roles are automatically created for you in the console first run experience, so if you intend to use the Amazon ECS console, you can move ahead to Create a Key Pair (p. 10). If you do not intend to use the Amazon ECS console, and instead plan to use the AWS CLI, complete the procedures in Amazon ECS Container Instance IAM Role (p. 180) and Amazon ECS Service Scheduler IAM Role (p. 182) before launching container instances or using Elastic Load Balancing load balancers with services.

Create a Key Pair

AWS uses public-key cryptography to secure the login information for your instance. A Linux instance, such as an Amazon ECS container instance, has no password to use for SSH access; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your container instance, then provide the private key when you log in using SSH.

If you haven't created a key pair already, you can create one using the Amazon EC2 console. Note that if you plan to launch instances in multiple regions, you'll need to create a key pair in each region. For more information about regions, see Regions and Availability Zones in the *Amazon EC2 User Guide for Linux Instances*.

To create a key pair

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for the key pair. You can select any region that's available to you, regardless of your location: however, key pairs are specific to a region. For example, if you plan to launch an instance in the US East (N. Virginia) region, you must create a key pair for the instance in the same region.

Note

Amazon ECS is available in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
EU (Frankfurt)	eu-central-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

3. Choose **Key Pairs** in the navigation pane.
4. Choose **Create Key Pair**.
5. Enter a name for the new key pair in the **Key pair name** field of the **Create Key Pair** dialog box, and then choose **Create**. Choose a name that is easy for you to remember, such as your IAM user name, followed by `-key-pair`, plus the region name. For example, *me*-key-pair-*useast1*.
6. The private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is `.pem`. Save the private key file in a safe place.

   Important

   This is the only chance for you to save the private key file. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

7. If you will use an SSH client on a Mac or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it.

```
$ chmod 400 your_user_name-key-pair-region_name.pem
```

For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your instance using your key pair

To connect to your Linux instance from a computer running Mac or Linux, specify the `.pem` file to your SSH client with the `-i` option and the path to your private key. To connect to your Linux instance from a computer running Windows, you can use either MindTerm or PuTTY. If you plan to use PuTTY, you'll need to install it and use the following procedure to convert the `.pem` file to a `.ppk` file.

(Optional) To prepare to connect to a Linux instance from Windows using PuTTY

1. Download and install PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/. Be sure to install the entire suite.

2. Start PuTTYgen (for example, from the **Start** menu, choose **All Programs, PuTTY, and PuTTYgen**).

3. Under **Type of key to generate**, choose **SSH-2 RSA**.
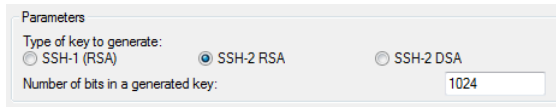
4. Choose **Load**. By default, PuTTYgen displays only files with the extension `.ppk`. To locate your `.pem` file, choose the option to display files of all types.
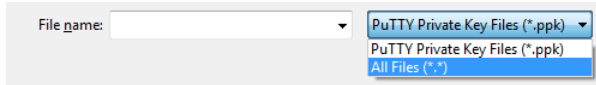
5. Select the private key file that you created in the previous procedure and choose **Open**. Choose **OK** to dismiss the confirmation dialog box.

6. Choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.

7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the `.ppk` file extension.

(Optional) Install the Amazon ECS Command Line Interface (CLI)

Note
This step is not required if you use the first-run wizard to create your cluster.

The Amazon EC2 Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose, a popular open-source tool for defining and running multi-container applications. For more information about installing and using the Amazon ECS CLI, see Using the Amazon ECS Command Line Interface (p. 199).

You can also choose to use Amazon ECS through the AWS AWS CLI. However, you will need to create your VPC and security groups separately, whereas both the Amazon ECS CLI and the first-run wizard will create this necessary infrastructure for you. For information about installing the AWS CLI or upgrading it to the latest version, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

Docker Basics

Docker is a technology that allows you to build, run, test, and deploy distributed applications that are based on Linux containers. Amazon ECS uses Docker images in task definitions to launch containers on EC2 instances in your clusters. For Amazon ECS product details, featured customer case studies, and FAQs, see the Amazon EC2 Container Service product detail pages.

The documentation in this guide assumes that readers possess a basic understanding of what Docker is and how it works. For more information about Docker, see What is Docker? and the Docker User Guide.

Topics

Installing Docker

Docker is available on many different operating systems, including most modern Linux distributions, like Ubuntu, and even Mac OSX and Windows. For more information about how to install Docker on your particular operating system, go to the Docker installation guide.

You don't even need a local development system to use Docker. If you are using Amazon EC2 already, you can launch an Amazon Linux instance and install Docker to get started.

To install Docker on an Amazon Linux instance

1. Launch an instance with the Amazon Linux AMI. For more information, see Launching an Instance in the *Amazon EC2 User Guide for Linux Instances*.

2. Connect to your instance. For more information, see Connect to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.

3. Update the installed packages and package cache on your instance.

   ```
   [ec2-user ~]$ sudo yum update -y
   ```

4. Install Docker.

   ```
   [ec2-user ~]$ sudo yum install -y docker
   ```

5. Start the Docker service.

```
[ec2-user ~]$ sudo service docker start
Starting cgconfig service:                                [  OK  ]
Starting docker:                                          [  OK  ]
```

6. Add the `ec2-user` to the `docker` group so you can execute Docker commands without using `sudo`.

```
[ec2-user ~]$ sudo usermod -a -G docker ec2-user
```

7. Log out and log back in again to pick up the new `docker` group permissions.

8. Verify that the `ec2-user` can run Docker commands without `sudo`.

```
[ec2-user ~]$ docker info
Containers: 2
Images: 24
Storage Driver: devicemapper
 Pool Name: docker-202:1-263460-pool
 Pool Blocksize: 65.54 kB
 Data file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata file: /var/lib/docker/devicemapper/devicemapper/metadata
 Data Space Used: 702.3 MB
 Data Space Total: 107.4 GB
 Metadata Space Used: 1.864 MB
 Metadata Space Total: 2.147 GB
 Library Version: 1.02.89-RHEL6 (2014-09-01)
Execution Driver: native-0.2
Kernel Version: 3.14.27-25.47.amzn1.x86_64
Operating System: Amazon Linux AMI 2014.09
```

(Optional) Sign up for a Docker Hub Account

Docker uses images that are stored in repositories to launch containers with. The most common Docker image repository (and the default repository for the Docker daemon) is Docker Hub. Although you don't need a Docker Hub account to use Amazon ECS or Docker, having a Docker Hub account gives you the freedom to store your modified Docker images so you can use them in your ECS task definitions.

For more information about Docker Hub, and to sign up for an account, go to https://hub.docker.com.

Docker Hub offers public and private registries. You can create a private registry on Docker Hub and configure Private Registry Authentication (p. 76) on your ECS container instances to use your private images in task definitions.

(Optional) Amazon EC2 Container Registry

Another registry option is Amazon EC2 Container Registry (Amazon ECR). Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. For Amazon ECR product details, featured customer case studies, and FAQs, see the Amazon EC2 Container Registry product detail pages. To finish this walkthrough using Amazon ECR, see Create a Docker Image in the *Amazon EC2 Container Registry User Guide*.

Create a Docker Image and Upload it to Docker Hub

Amazon ECS task definitions use Docker images to launch containers on the container instances in your clusters. In this section, you create a Docker image of a simple PHP web application, and test it on your local system or EC2 instance, and then push the image to your Docker Hub registry so you can use it in an ECS task definition.

To create a Docker image of a PHP web application

1. Install **git** and use it to clone the simple PHP application from your GitHub repository onto your system.

    a. Install git.

    ```
    [ec2-user ~]$ sudo yum install -y git
    ```

    b. Clone the simple PHP application onto your system.

    ```
    [ec2-user ~]$ git clone https://github.com/awslabs/ecs-demo-php-
    simple-app
    ```

2. Change directories to the ecs-demo-php-simple-app folder.

    ```
    [ec2-user ~]$ cd ecs-demo-php-simple-app
    ```

3. Examine the Dockerfile in this folder. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference.

    ```
    [ec2-user ecs-demo-php-simple-app]$ cat Dockerfile
    FROM ubuntu:12.04

    # Install dependencies
    RUN apt-get update -y
    RUN apt-get install -y git curl apache2 php5 libapache2-mod-php5 php5-
    mcrypt php5-mysql

    # Install app
    RUN rm -rf /var/www/*
    ADD src /var/www

    # Configure apache
    RUN a2enmod rewrite
    RUN chown -R www-data:www-data /var/www
    ENV APACHE_RUN_USER www-data
    ENV APACHE_RUN_GROUP www-data
    ENV APACHE_LOG_DIR /var/log/apache2

    EXPOSE 80

    CMD ["/usr/sbin/apache2", "-D",  "FOREGROUND"]
    ```

    This Dockerfile uses the Ubuntu 12.04 image. The RUN instructions update the package caches, install some software packages for the web server and PHP support, and then add your PHP

application to the web server's document root. The `EXPOSE` instruction exposes port 80 on the container, and the `CMD` instruction starts the web server.

4. Build the Docker image from your Dockerfile. Substitute *my-dockerhub-username* with your Docker Hub user name.

```
[ec2-user ecs-demo-php-simple-app]$ docker build -t my-dockerhub-username/
amazon-ecs-sample .
```

5. Run **docker images** to verify that the image was created correctly and that the image name contains a repository that you can push to (in this example, your Docker Hub user name).

```
[ec2-user ecs-demo-php-simple-app]$ docker images
REPOSITORY                                TAG                IMAGE ID
        CREATED              VIRTUAL SIZE
my-dockerhub-username/amazon-ecs-sample   latest             43c52559a0a1
        12 minutes ago       258.1 MB
ubuntu                                    12.04              78cef618c77e
        3 weeks ago          133.7 MB
```

6. Run the newly built image. The `-p 80:80` option maps the exposed port 80 on the container to port 80 on the host system. For more information about **docker run**, go to the Docker run reference.

```
[ec2-user ecs-demo-php-simple-app]$ docker run -p 80:80 my-dockerhub-
username/amazon-ecs-sample
apache2: Could not reliably determine the server's fully qualified domain
 name, using 172.17.0.2 for ServerName
```

Note

Output from the Apache web server is displayed in the terminal window. You can ignore the `"Could not reliably determine the server's fully qualified domain name"` message.

7. Open a browser and point to the server that is running Docker and hosting your container.

- If you are using an EC2 instance, this is the **Public DNS** value for the server, which is the same address you use to connect to the instance with SSH. Make sure that the security group for your instance allows inbound traffic on port 80.

- If you are running Docker locally, point your browser to http://localhost/.

- If you are using **docker-machine** on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the **docker-machine ip** command, substituting *machine-name* with the name of the docker machine you are using.

```
$ docker-machine ip machine-name
192.168.59.103
```

You should see a web page running the simple PHP app.
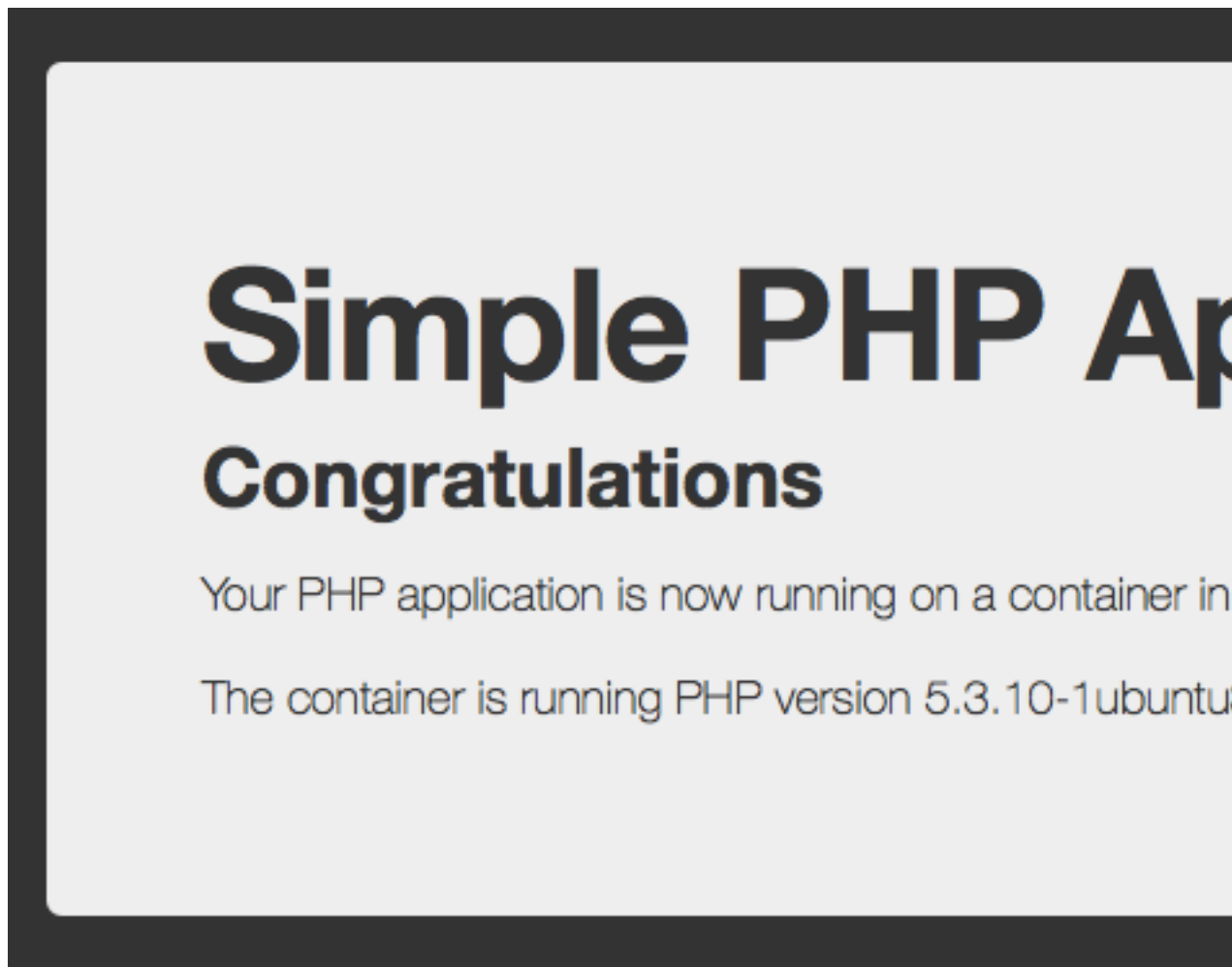
8. Stop the Docker container by typing **Ctrl+c**.

Next Steps

After the image push is finished, you can use the *my-dockerhub-username*/amazon-ecs-sample image in your Amazon ECS task definitions, which you can use to run tasks with.

To register a task definition with the `amazon-ecs-sample` image

1. Examine the `simple-app-task-def.json` file in the `ecs-demo-php-simple-app` folder.

```
{
    "family": "console-sample-app",
    "volumes": [
        {
            "name": "my-vol",
            "host": {}
        }
    ],
    "containerDefinitions": [
        {
```

```
            "environment": [],
            "name": "simple-app",
            "image": "amazon/amazon-ecs-sample",
            "cpu": 10,
            "memory": 500,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "my-vol",
                    "containerPath": "/var/www/my-vol"
                }
            ],
            "entryPoint": [
                "/usr/sbin/apache2",
                "-D",
                "FOREGROUND"
            ],
            "essential": true
        },
        {
            "name": "busybox",
            "image": "busybox",
            "cpu": 10,
            "memory": 500,
            "volumesFrom": [
            {
              "sourceContainer": "simple-app"
            }
            ],
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "/bin/sh -c \"while true; do /bin/date > /var/www/my-vol/
date; sleep 1; done\""
            ],
            "essential": false
        }
    ]
}
```

This task definition JSON file specifies two containers, one of which uses the amazon-ecs-
sample image. By default, this image is pulled from the Amazon Docker Hub repository, but you
can change the *amazon* repository defined above to your own repository if you want to use the
my-dockerhub-username/amazon-ecs-sample image you pushed earlier.

2. Register a task definition with the simple-app-task-def.json file.

```
[ec2-user ecs-demo-php-simple-app]$ aws ecs register-task-definition --
cli-input-json file://simple-app-task-def.json
```

The task definition is registered in the console-sample-app family as defined in the JSON file.

To run a task with the `console-sample-app` task definition

> **Important**
> Before you can run tasks in Amazon ECS, you need to launch container instances into your cluster. For more information about how to set up and launch container instances, see Setting Up with Amazon ECS (p. 8) and Getting Started with Amazon ECS (p. 20).

* Use the following AWS CLI command to run a task with the `console-sample-app` task definition.

```
[ec2-user ecs-demo-php-simple-app]$ aws ecs run-task --task-definition
 console-sample-app
```

Getting Started with Amazon ECS

Let's get started with Amazon EC2 Container Service (Amazon ECS) by creating a task definition, scheduling tasks, and configuring a cluster in the Amazon ECS console.

You can optionally create an Amazon EC2 Container Registry (Amazon ECR) image repository and push an image to it. For more information on Amazon ECR, see the *Amazon EC2 Container Registry User Guide*.

The Amazon ECS first run wizard will guide you through the process to get started with Amazon ECS. The wizard gives you the option of creating a cluster and launching our sample web application, or if you already have a Docker image you would like to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

Important
Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 8) and that your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 190) IAM policy example.

Choose your Amazon ECS first run wizard configuration options

1. Open the Amazon ECS console first run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.
2. Select your Amazon ECS first run options.

To create an Amazon ECS cluster and deploy a container application to it, check the top option. To create an Amazon ECR repository and push an image to it, which you can use in your Amazon ECS task definitions, check the bottom option. Choose **Continue** to proceed.

3. If you've chosen to create an Amazon ECR repository, then complete the next two sections of the first run wizard, **Configure repository** and **Build, tag, and push Docker image** . If you are not creating an Amazon ECR repository, skip ahead to .

Configure repository

A repository is a place that you store Docker images in Amazon ECR. Every time you push or pull an image from Amazon ECR, you specify the registry and repository location to tell Docker where to push the image to or where to pull it from.

- For **Repository name**, enter a unique name for your repository and choose **Next step**.

Build, tag, and push Docker image

In this section of the wizard, you use the Docker CLI to tag an existing local image (that you have built from a Dockerfile or pulled from another registry, such as Docker Hub) and then push the tagged image to your Amazon ECR registry.

1. Retrieve the **docker login** command that you can use to authenticate your Docker client to your registry by pasting the **aws ecr get-login** command from the console into a terminal window.

   Note
   The **get-login** command is available in the AWS CLI starting with version 1.9.15. You can check your AWS CLI version with the **aws --version** command.

2. Run the **docker login** command that was returned in the previous step. This command provides an authorization token that is valid for 12 hours.

   Important
   When you execute this **docker login** command, the command string can be visible by other users on your system in a process list (**ps -e**) display. Because the **docker login** command contains authentication credentials, there is a risk that other users on your system could view them this way and use them to gain push and pull access to your repositories. If you are not on a secure system, you should consider this risk and log in interactively by omitting the `-p` *`password`* option, and then entering the password when prompted.

3. (Optional) If you have a Dockerfile for the image you want to push, build the image and tag it for your new repository by pasting the **docker build** command from the console into a terminal window (make sure you are in the same directory as your Dockerfile).

4. Tag the image for your Amazon ECR registry and your new repository by pasting the **docker tag** command from the console into a terminal window. The console command assumes that your image was built from a Dockerfile in the previous step; if you did not build your image from a Dockerfile, replace the first instance of *`repository`*`:latest` with the image ID or image name of your local image that you want to push.

5. Push the newly tagged image to your Amazon ECR repository by pasting the **docker push** command into a terminal window.

6. Choose **Done** to finish.

Create a task definition

A task definition is like a blue print for your application. Every time you launch a task in Amazon ECS, you specify a task definition so the service knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.

1. Configure your task definition parameters.

   The first run wizard comes preloaded with a task definition, and you can see the `simple-app` container defined in the console. You can optionally rename the task definition or review and

edit the resources used by the container (such as CPU units and memory limits) by choosing the container name and editing the values shown (CPU units are under the **Advanced container configuration** menu). Task definitions created in the first run wizard are limited to a single container for simplicity's sake. You can create multi-container task definitions later in the Amazon ECS console.

> **Note**
> If you are using an Amazon ECR image in your task definition, be sure to use the full `registry/repository:tag` naming for your Amazon ECR images. For example, `aws_account_id`.dkr.ecr.`region`.amazonaws.com/`my-web-app`:`latest`.

For more information on what each of these task definition parameters does, see Task Definition Parameters (p. 88).

2. Choose **Next step** to continue.

Configure service

In this section of the wizard, you select how you would like to configure the Amazon ECS service that is created from your task definition. A service launches and maintains a specified number of copies of the task definition in your cluster. The **Amazon ECS sample** application is a web-based "Hello World" style application that is meant to run indefinitely, so by running it as a service, it will restart if the task becomes unhealthy or unexpectedly stops.

1. In the **Service Name** field, select a name for your service.

2. In the **Desired number of tasks** field, enter the number of tasks you would like to launch with your specified task definition.

   > **Note**
   > If your task definition contains static port mappings, the number of container instances you launch in the next section of the wizard must be greater than or equal to the number of tasks specified here.

3. (Optional) You can choose to use an Elastic Load Balancing load balancer with your service. When a task is launched from a service that is configured to use a load balancer, the container instance that the task is launched on is registered with the load balancer and traffic from the load balancer is distributed across the instances in the load balancer.

   > **Important**
   > Elastic Load Balancing load balancers do incur cost while they exist in your AWS resources. For more information on Elastic Load Balancing pricing, see Elastic Load Balancing Pricing.

   Complete the following steps to use a load balancer with your service.

   a. In the **Elastic load balancing** section, choose the **Container name: host port** menu, and then choose **simple-app:80**. The default values here are set up for the sample application, but you can configure different listener options for the load balancer. For more information, see Service Load Balancing (p. 124).

   b. In the **Service IAM Role** section, choose the **Select IAM role for service** menu, and then choose an existing Amazon ECS service (`ecsServiceRole`) role that you have already created, or click **Create new role** to create the required IAM role for your service.

4. Review your load balancer settings and click **Next Step**.

Configure cluster

In this section of the wizard, you name your cluster, and then configure the container instances that your tasks can be placed on, the address range that you can reach your instances and load balancer from, and the IAM roles to use with your container instances that let Amazon ECS take care of this configuration for you.

1. In the **Cluster name** field, choose a name for your cluster.

2. In the **EC2 instance type** field, choose the instance type to use for your container instances. Instance types with more CPU and memory resources can handle more tasks. For more information on the different instance types, see Amazon EC2 Instances.

3. In the **Number of instances** field, type the number of Amazon EC2 instances you want to launch into your cluster for tasks to be placed on. The more instances you have in your cluster, the more tasks you can place on them. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see Amazon EC2 Pricing.

   Note
   If you created a service with more than one desired task in it that exposes container ports on to container instance ports, such as the **Amazon ECS sample** application, you need to specify at least that many instances here.

4. Select a key pair name to use with your container instances. This is required for you to log into your instances with SSH; if you do not specify a key pair here, you cannot connect to your container instances with SSH. If you do not have a key pair, you can create one in the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

5. (Optional) In the **Security Group** section, you can choose a CIDR block that restricts access to your instances. The default value (**Anywhere**)allows access from the entire Internet.

6. In the **Container instance IAM role** section, choose an existing Amazon ECS container instance (`ecsInstanceRole`) role that you have already created, or choose **Create new role** to create the required IAM role for your container instances.

7. Click **Review and Launch** to proceed.

Review

1. Review your task definition, task configuration, and cluster configurations and click **Launch Instance & Run Service** to finish. You are directed to a **Launch Status** page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).

2. After the launch is complete, choose **View service** to view your service in the Amazon ECS console.

Cleaning Up your Amazon ECS Resources

When you are finished experiment with or using a particular Amazon ECS cluster, you should clean up the resources associated with it to avoid incurring charges for resources that you are not using.

Some Amazon ECS resources, such as tasks, services, clusters, and container instances, are cleaned up using the Amazon ECS console. Other resources, such as Amazon EC2 instances, Elastic Load Balancing load balancers, and Auto Scaling groups, must be cleaned up manually in the Amazon EC2 console or by deleting the AWS CloudFormation stack that created them.

Topics

Scale Down Services

If your cluster contains any services, you should first scale down the desired count of tasks in these services to 0 so that Amazon ECS does not try to start new tasks on your container instances while you are cleaning up. Follow the procedure in Updating a Service (p. 147) and enter 0 in the **Number of tasks** field.

Alternatively, you can use the following AWS CLI command to scale down your service. Be sure to substitute the region name, cluster name, and service name for each service that you are scaling down.

```
$ aws --region us-west-2 ecs update-service --cluster default --
service service_name --desired-count 0
```

Delete Services

Before you can delete a cluster, you must delete the services inside that cluster. After your service has scaled down to 0 tasks, you can delete it. For each service inside your cluster, follow the procedures in Deleting a Service (p. 148) to delete it.

Alternatively, you can use the following AWS CLI command to delete your services. Be sure to substitute the region name, cluster name, and service name for each service that you are deleting.

```
$ aws  --region us-west-2 ecs delete-service --cluster default --
service service_name
```

Deregister Container Instances

Before you can delete a cluster, you must deregister the container instances inside that cluster. For each container instance inside your cluster, follow the procedures in Deregister a Container Instance (p. 56) to deregister it.

Alternatively, you can use the following AWS CLI command to deregister your container instances. Be sure to substitute the region name, cluster name, and container instance ID for each container instance that you are deregistering.

```
$ aws --region us-west-2 ecs deregister-container-instance --cluster default
 --container-instance container_instance_id --force
```

Delete a Cluster

After you have removed the active resources from your Amazon ECS cluster, you can delete it. Use the following procedure to delete your cluster.

To delete a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region that your cluster is in.
3. In the navigation pane, select **Clusters**.
4. On the **Clusters** page, click the **x** in the upper-right-hand corner of the cluster you want to delete.
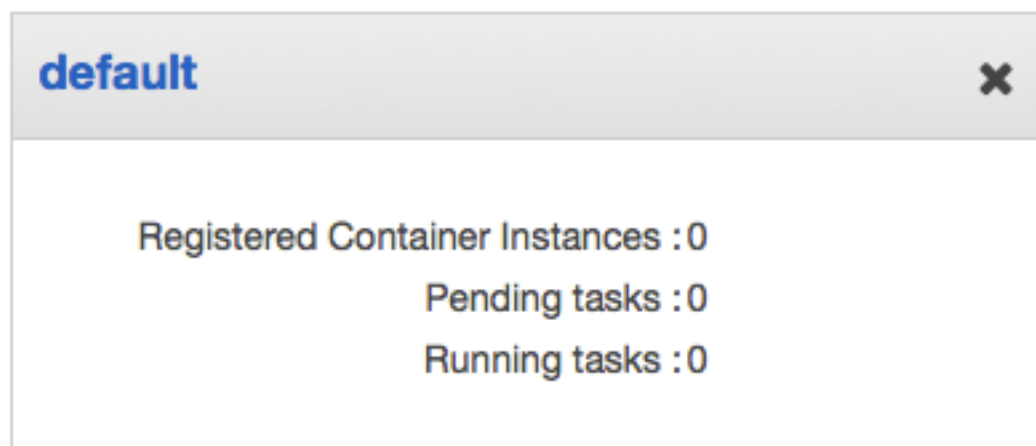
5. Choose **Yes, Delete** to delete the cluster.

Alternatively, you can use the following AWS CLI command to delete your cluster. Be sure to substitute the region name and cluster name for each cluster that you are deleting.

```
$ aws --region us-west-2 ecs delete-cluster --cluster default
```

Delete the AWS CloudFormation Stack

If you created your Amazon ECS resources by following the console first-run wizard, then your resources are contained in a AWS CloudFormation stack. You can completely clean up all of your remaining AWS resources that are associated with this stack by deleting it. Deleting the CloudFormation stack terminates the EC2 instances, removes the Auto Scaling group, deletes any Elastic Load Balancing load balancers, and removes the Amazon VPC subnets and Internet gateway associated with the cluster.

To delete the AWS CloudFormation stack

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/.
2. From the navigation bar, select the region that your cluster was created in.
3. Select the stack that is associated with your Amazon ECS resources. The **Stack Name** value starts with `EC2ContainerService-default`.
4. Choose **Delete Stack** and then choose **Yes, Delete** to delete your stack resources.

Amazon ECS Clusters

An Amazon EC2 Container Service (Amazon ECS) cluster is a logical grouping of container instances that you can place tasks on. When you first use the Amazon ECS service, a default cluster is created for you, but you can create multiple clusters in an account to keep your resources separate.

Topics

Cluster Concepts

- Clusters can contain multiple different container instance types.
- Clusters are region-specific.
- Container instances can only be a part of one cluster at a time.
- You can create custom IAM policies for your clusters to allow or restrict users' access to specific clusters. For more information, see the Clusters (p. 192) section in Amazon ECS IAM Policy Examples (p. 190).

Creating a Cluster

You can create a Amazon ECS cluster using the AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 8). After you've created your cluster, you can register container instances into it and run tasks and services.

To create a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. From the navigation bar, select the region to use.

    Note
    Amazon ECS is available in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
EU (Frankfurt)	eu-central-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

3. In the navigation pane, choose **Clusters**.

4. On the **Clusters** page, select **Create Cluster**.

5. In the **Cluster name** field, enter a name for your cluster. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

6. Choose **Create** to create your cluster.

Scaling a Cluster

If your cluster was created with the console first-run experience described in Getting Started with Amazon ECS (p. 20) after November 24th, 2015, then the Auto Scaling group associated with the AWS CloudFormation stack created for your cluster can be scaled up or down to add or remove container instances. You can perform this scaling operation from within the Amazon ECS console.

If your cluster was not created with the console first-run experience described in Getting Started with Amazon ECS (p. 20) after November 24th, 2015, then you cannot scale your cluster from the Amazon ECS console. However, you can still modify existing Auto Scaling groups associated with your cluster in the Auto Scaling console. If you do not have an Auto Scaling group associated with your cluster, you can manually launch or terminate container instances from the Amazon EC2 console; for more information see Launching an Amazon ECS Container Instance (p. 39).

To scale a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. From the navigation bar, choose the region that your cluster exists in.

3. In the navigation pane, choose **Clusters**.

4. Choose the cluster that you want to scale.

5. On the **Cluster : *name*** page, choose the **ECS Instances** tab.
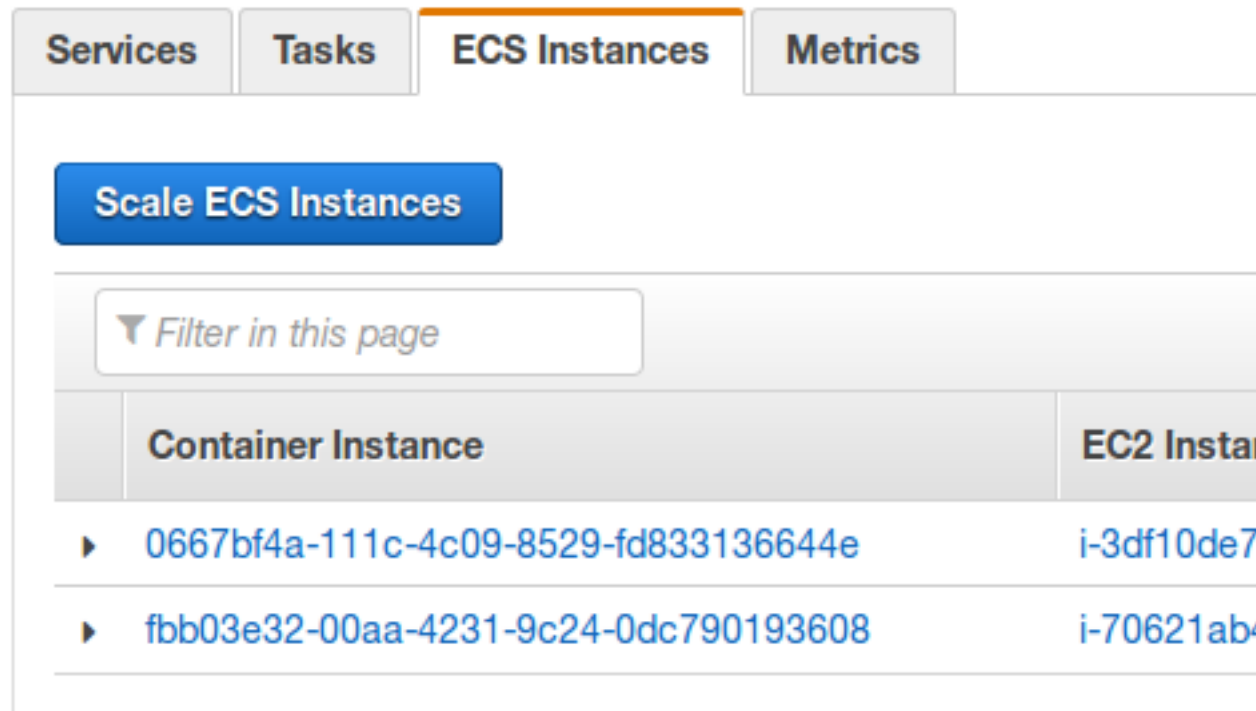
If a **Scale ECS Instances** button appears, then you can scale your cluster in the next step. If not, you must manually adjust your Auto Scaling group to scale up or down your instances, or you can manually launch or terminate your container instances in the Amazon EC2 console.

6. Choose **Scale ECS Instances**.

7. In the **Desired number of instances** field, enter the number of instances you wish to scale your cluster to and choose **Scale**.

   Note
   If you reduce the number of container instances in your cluster, any tasks that are running on terminated instances are stopped.

Deleting a Cluster

If you are finished using a cluster, you can delete it. When you delete a cluster in the Amazon ECS console, the associated resources that are deleted with it vary depending on how the cluster was created. Step 5 (p. 30) of the following procedure changes based on that condition.

If your cluster was created with the console first-run experience described in Getting Started with Amazon ECS (p. 20) after November 24th, 2015, then the AWS CloudFormation stack that was created for your cluster is also deleted when you delete your cluster.

If your cluster was created manually or with the console first run experience prior to November 24th, 2015, then you must deregister (or terminate) any container instances associated with the cluster before you can delete it. For more information, see Deregister a Container Instance (p. 56). In this case, after the cluster is deleted, you should delete any remaining AWS CloudFormation stack resources or Auto Scaling groups associated with the cluster to avoid incurring any future charges for those resources. For more information, see Delete the AWS CloudFormation Stack (p. 26).

To delete a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. From the navigation bar, select the region to use.

   Note
   Amazon ECS is available in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
EU (Frankfurt)	eu-central-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

3. In the navigation pane, choose **Clusters**.

4. On the **Clusters** page, find the cluster you want to delete, and choose the **X** in the upper-right-hand corner of the cluster window.

   Note
   If your cluster has registered container instances, you must deregister or terminate them. For more information, see Deregister a Container Instance (p. 56).
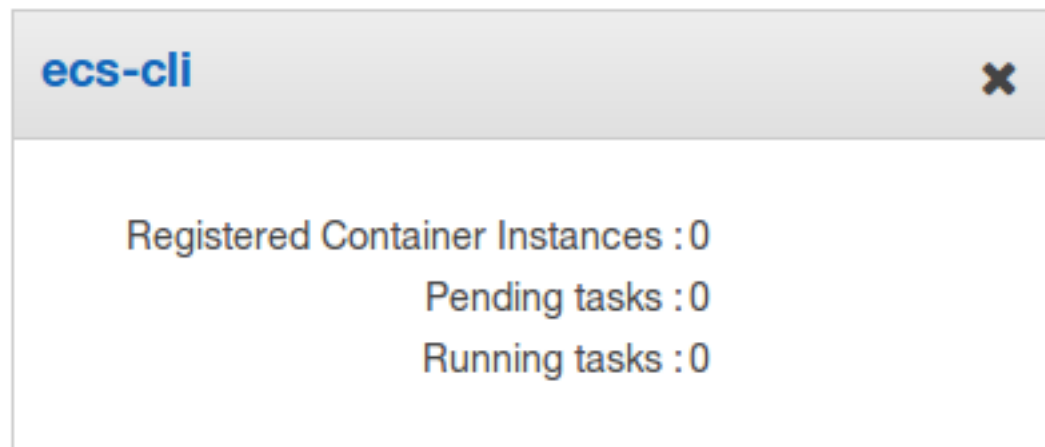
5. Choose **Delete** to delete the cluster. You will see one of two confirmation prompts:

   - **Deleting the cluster also deletes the CloudFormation stack EC2ContainerService-*cluster_name*:** Deleting this cluster cleans up the associated

resources that were created with the cluster, including Auto Scaling groups, VPCs or load balancers.

- **Deleting the cluster does not affect CloudFormation resources...:** Deleting this cluster does not clean up any resources that are associated with the cluster, including Auto Scaling groups, VPCs or load balancers. Also, any container instances that are registered with this cluster must be deregistered or terminated before you can delete the cluster; for more information, see Deregister a Container Instance (p. 56). You can visit the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/ to update or delete any of these resources; for more information, see Delete the AWS CloudFormation Stack (p. 26).

Amazon ECS Container Instances

An Amazon EC2 Container Service (Amazon ECS) container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into a cluster. When you run tasks with Amazon ECS, your tasks are placed on your active container instances.

Topics

Container Instance Concepts

- Your container instance must be running the Amazon ECS container agent to register into one of your clusters. If you are using the Amazon ECS-optimized AMI, the agent is already installed. If you want to use a different operating system, you need to install the agent. For more information, see Amazon ECS Container Agent (p. 58).

- Because the Amazon ECS container agent makes calls to Amazon ECS on your behalf, you need to launch container instances with an IAM role that authenticates to your account and provides the required resource permissions. For more information, see Amazon ECS Container Instance IAM Role (p. 180).

- If any of the containers associated with your tasks require external connectivity, you can map their network ports to ports on the host Amazon ECS container instance so they are reachable from the

Internet. Your container instance security group must allow inbound access to the ports you want to expose. For more information, see Create a Security Group in the Amazon VPC Getting Started Guide.

- Amazon ECS strongly recommends launching your container instances inside a VPC, because Amazon VPC delivers more control over your network and offers more extensive configuration capabilities. For more information, see Amazon EC2 and Amazon Virtual Private Cloud in the *Amazon EC2 User Guide for Linux Instances.*

- Container instances need external network access to communicate with the Amazon ECS service endpoint, so if your container instances do not have public IP addresses, then they must use network address translation (NAT) or an HTTP proxy to provide this access. For more information, see NAT Instances in the *Amazon VPC User Guide* and HTTP Proxy Configuration (p. 80) in this guide.

- The type of EC2 instance that you choose for your container instances determines the resources available in your cluster. Amazon EC2 provides different instance types, each with different CPU, memory, storage, and networking capacity that you can use to run your tasks. For more information, see Amazon EC2 Instances.

- Because each container instance has unique state information that is stored locally on the container instance and within Amazon ECS, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances with the latest Amazon ECS-optimized AMI in the new cluster. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances* and Launching an Amazon ECS Container Instance (p. 39).

- Because each container instance has unique state information that is stored locally on the container instance and within Amazon ECS, you cannot stop a container instance and change its instance type. Instead, we recommend that you terminate the container instance and launch a new container instance with the desired instance size and the latest Amazon ECS-optimized AMI in your desired cluster. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances* and Launching an Amazon ECS Container Instance (p. 39).

Container Instance Life Cycle

When the Amazon ECS container agent registers an instance into your cluster, the container instance reports its status as `ACTIVE` and its agent connection status as `TRUE`. This container instance can accept run task requests.

If you stop (not terminate) an Amazon ECS container instance, the status remains `ACTIVE`, but the agent connection status transitions to `FALSE` within a few minutes. Any tasks that were running on the container instance stop. If you start the container instance again, the container agent reconnects with the Amazon ECS service, and you are able to run tasks on the instance again.

Important
If you stop and start a container instance, or reboot that instance, some older versions of the Amazon ECS container agent register the instance again without deregistering the original container instance ID, so Amazon ECS will list more container instances in your cluster than you actually have. (If you have duplicate container instance IDs for the same Amazon EC2 instance ID, you can safely deregister the duplicates that are listed as `ACTIVE` with an agent connection status of `FALSE`.) This issue is fixed in the current version of the Amazon ECS container agent. To update to the current version, see Updating the Amazon ECS Container Agent (p. 63).

If you deregister or terminate a container instance, the container instance status changes to `INACTIVE` immediately, and the container instance is no longer reported when you list your container instances. However, you can still describe the container instance for one hour following termination. After one hour, the instance description is no longer available.

Check the Instance Role for your Account

The Amazon ECS container agent makes calls to the Amazon ECS APIs on your behalf, so container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you.

In most cases, the Amazon ECS instance role is automatically created for you in the console first-run experience. You can use the following procedure to check and see if your account already has an Amazon ECS service role.

To check for the `ecsInstanceRole` in the IAM console

1. Sign in to the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsInstanceRole`. If the role exists, you do not need to create it. If the role does not exist, follow the procedures in Amazon ECS Container Instance IAM Role (p. 180) to create the role.

Container Instance AMIs

The basic Amazon EC2 Container Service (Amazon ECS) container instance specification consists of the following:

Required

*   A modern Linux distribution running at least version 3.10 of the Linux kernel.
*   The Amazon ECS container agent (preferably the latest version). For more information, see Amazon ECS Container Agent (p. 58).
*   A Docker daemon running at least version 1.5.0, and any Docker runtime dependencies. For more information, see Check runtime dependencies in the Docker documentation.

    Note
    For the best experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS agent version that you are using. For more information, see Amazon ECS-optimized AMI Container Agent Versions (p. 61).

Recommended

*   An initialization and nanny process to run and monitor the Amazon ECS agent. The Amazon ECS-optimized AMI uses the `ecs-init` upstart process. For more information, see the `ecs-init` project on GitHub.

The Amazon ECS-optimized AMI is preconfigured with these requirements and recommendations, and we recommend that you use the Amazon ECS-optimized AMI for your container instances unless your application requires a specific operating system or a Docker version that is not yet available in that AMI. For more information, see Amazon ECS-optimized AMI (p. 34).

Amazon ECS-optimized AMI

The Amazon ECS-optimized AMI is the recommended AMI for you to use to launch your Amazon ECS container instances. Although you can create your own container instance AMI that meets the

basic specifications outlined in Container Instance AMIs (p. 34), the Amazon ECS-optimized AMI is preconfigured and tested on Amazon ECS by AWS engineers, and it is the simplest AMI for you to get started and to get your containers running on AWS quickly.

The current Amazon ECS-optimized AMI (`amzn-ami-2016.09.b-amazon-ecs-optimized`) consists of:

- The latest minimal version of the Amazon Linux AMI
- The latest version of the Amazon ECS container agent (`1.13.1`)
- The recommended version of Docker for the latest Amazon ECS container agent (`1.11.2`)
- The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (`1.13.1-1`)

How to Launch the Latest Amazon ECS-optimized AMI

The following are several ways that you can launch the latest Amazon ECS-optimized AMI into your cluster:

- The Amazon ECS console first-run wizard launches your container instances with the latest Amazon ECS-optimized AMI. For more information, see Getting Started with Amazon ECS (p. 20).
- You can launch your container instances manually in the Amazon EC2 console by following the procedures in Launching an Amazon ECS Container Instance (p. 39), or by choosing the **EC2 console link** in the table below that corresponds to your cluster's region.
- Use an **AMI ID** from the table below that corresponds to your cluster's region with the AWS CLI, the AWS SDKs, or an AWS CloudFormation template to launch your instances.

The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI Name	AMI ID	EC2 console link
`us-east-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-eca289fb`	Launch instance
`us-east-2`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-446f3521`	Launch instance
`us-west-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-9fadf8ff`	Launch instance
`us-west-2`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-7abc111a`	Launch instance
`eu-west-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-a1491ad2`	Launch instance
`eu-central-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-54f5303b`	Launch instance
`ap-northeast-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-9cd57ffd`	Launch instance

Region	AMI Name	AMI ID	EC2 console link
`ap-southeast-1`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-a900a3ca`	Launch instance
`ap-southeast-2`	**amzn-ami-2016.09.b-amazon-ecs-optimized**	`ami-5781be34`	Launch instance

For previous versions of the Amazon ECS-optimized AMI and its corresponding Docker and Amazon ECS container agent versions, see Amazon ECS-optimized AMI Container Agent Versions (p. 61).

Storage Configuration

By default, the Amazon ECS-optimized AMI ships with 30 GiB of total storage (which you can modify at launch time to increase or decrease the available storage on your container instance). This storage is used for the operating system and for Docker images and metadata. The sections below describe the storage configuration of the Amazon ECS-optimized AMI, based on the AMI version.

Version 2015.09.d and later

Amazon ECS-optimized AMIs from version `2015.09.d` and later launch with an 8 GiB volume for the operating system that is attached at `/dev/xvda` and mounted as the root of the file system. There is an additional 22 GiB volume that is attached at `/dev/xvdcz` that Docker uses for image and metadata storage. The volume is configured as a Logical Volume Management (LVM) device and it is accessed directly by Docker via the `devicemapper` back end. Because the volume is not mounted, you cannot use standard storage information commands (such as **df -h**) to determine the available storage. However, you can use LVM commands and **docker info** to find the available storage by following the procedure below. For more information about LVM, see the LVM HOWTO in The Linux Documentation Project.

The **docker-storage-setup** utility configures the LVM volume group and logical volume for Docker when the instance launches. By default, **docker-storage-setup** creates a volume group called `docker`, adds `/dev/xvdcz` as a physical volume to that group, and then creates a logical volume called `docker-pool` that uses 99% of the available storage in the volume group. The remaining 1% of the available storage is reserved for metadata.

> **Note**
> Earlier Amazon ECS-optimized AMI versions (`2015.09.d` to `2016.03.a`) create a logical volume that uses 40% of the available storage in the volume group. When the logical volume becomes 60% full, the logical volume is increased in size by 20%.

To determine the available storage for Docker

- You can use the LVM commands, **vgs** and **lvs**, or the **docker info** command to view available storage for Docker.

  > **Note**
  > The LVM command output displays storage values in GiB (2^30 bytes), and **docker info** displays storage values in GB (10^9 bytes).

- You can view the available storage in the volume group with the **vgs** command. This command shows the total size of the volume group and the available space in the volume group that can be used to grow the logical volume. The example below shows a 22 GiB volume with 204 MiB of free space.

```
[ec2-user ~]$ sudo vgs
  VG     #PV #LV #SN Attr   VSize  VFree
```

```
docker   1   1   0 wz--n- 22.00g 204.00m
```

- You can view the available space in the logical volume with the **lvs** command. The example below shows a logical volume that is 21.75 GiB in size, and it is 7.63% full. This logical volume has the ability to grow until there is no more free space in the volume group.

```
[ec2-user@ ~]$ sudo lvs
  LV         VG     Attr       LSize   Pool Origin Data%  Meta%  Move
 Log Cpy%Sync Convert
  docker-pool docker twi-aot--- 21.75g              7.63   4.96
```

- The **docker info** command also provides information about how much data space it is using, and how much data space is available. However, its available space value is based on the logical volume size that it is using.

  Note
  Because **docker info** displays storage values as GB (10^9 bytes), instead of GiB (2^30 bytes), the values displayed here look larger for the same amount of storage displayed with **lvs**, but the values are equal (23.35 GB = 21.75 GiB).

```
[ec2-user ~]$ docker info | grep "Data Space"
 Data Space Used: 1.782 GB
 Data Space Total: 23.35 GB
 Data Space Available: 21.57 GB
```

To extend the Docker logical volume

The easiest way to add storage to your container instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you are unable to do this, you can add storage to the volume group that Docker uses and extend its logical volume by following these steps.

Note
If your container instance storage is filling up too quickly, there are a few actions that you can take to reduce this effect:

- (Amazon ECS container agent 1.8.0 and later) Reduce the amount of time that stopped or exited containers remain on your container instances. The ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION agent configuration variable sets the time duration to wait from when a task is stopped until the docker container is removed (by default, this value is 3 hours). However, as this removes the docker container data, be aware that if this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. For more information, see Amazon ECS Container Agent Configuration (p. 69).
- Remove non-running containers and unused images from your container instances. You can use the following example commands to manually remove stopped containers and unused images; however, deleted containers cannot be inspected later, and deleted images must be pulled again prior to starting new containers from them.

  To remove non-running containers, execute the following command on your container instance:

```
docker rm $(docker ps -aq)
```

  To remove unused images, execute the following command on your container instance:

```
docker rmi $(docker images -q)
```

1. Create a new Amazon EBS volume in the same Availability Zone as your container instance. For more information, see Creating an Amazon EBS Volume in the *Amazon EC2 User Guide for Linux Instances*.

2. Attach the volume to your container instance. The default location for the Docker data volume is `/dev/xvdcz`. For consistency, attach additional volumes in reverse alphabetical order from that device name (for example, `/dev/xvdcy`). For more information, see Attaching an Amazon EBS Volume to an Instance in the *Amazon EC2 User Guide for Linux Instances*.

3. Connect to your container instance using SSH. For more information, see Connect to your Container Instance (p. 42).

4. Check the size of your `docker-pool` logical volume. The example below shows a logical volume of 409.19 GiB.

```
[ec2-user ~]$ sudo lvs
  LV         VG     Attr       LSize    Pool Origin Data%  Meta%  Move Log
 Cpy%Sync Convert
  docker-pool docker twi-aot--- 409.19g                    0.16   0.08
```

5. Check the current available space in your volume group. The example below shows 612.75 GiB in the `VFree` column.

```
[ec2-user ~]$ sudo vgs
  VG     #PV #LV #SN Attr   VSize    VFree
  docker   1   1   0 wz--n- 1024.00g 612.75g
```

6. Add the new volume to the `docker` volume group, substituting the device name to which you attached the new volume. In this example, a 1 TiB volume was previously added and attached to `/dev/xvdcy`.

```
[ec2-user ~]$ sudo vgextend docker /dev/xvdcy
  Physical volume "/dev/sdcy" successfully created
  Volume group "docker" successfully extended
```

7. Verify that your volume group size has increased with the **vgs** command. The `VFree` column should show the increased storage size. The example below now has 1.6 TiB in the `VFree` column, which is 1 TiB larger than it was previously. Your `VFree` column should be the sum of the original `VFree` value and the size of the volume you attached.

```
[ec2-user ~]$ sudo vgs
  VG     #PV #LV #SN Attr   VSize VFree
  docker   2   1   0 wz--n- 2.00t 1.60t
```

8. Extend the `docker-pool` logical volume with the size of the volume you added earlier. The command below adds 1024 GiB to the logical volume, which is entered as `1024G`.

```
[ec2-user ~]$ sudo lvextend -L+1024G /dev/docker/docker-pool
  Size of logical volume docker/docker-pool_tdata changed from 409.19 GiB
 (104752 extents) to 1.40 TiB (366896 extents).
  Logical volume docker-pool successfully resized
```

9. Verify that your logical volume has increased in size.

```
[ec2-user ~]$ sudo lvs
  LV         VG     Attr       LSize Pool Origin Data%  Meta%  Move Log
 Cpy%Sync Convert
  docker-pool docker twi-aot--- 1.40t                   0.04   0.12
```

10. (Optional) Verify that **docker info** also recognizes the added storage space.

> **Note**
> Because **docker info** displays storage values as GB (10^9 bytes), instead of GiB (2^30 bytes), the values displayed here look larger for the same amount of storage displayed with **lvs**, but the values are equal (1.539 TB =1.40 TiB).

```
[ec2-user ~]$ docker info | grep "Data Space"
 Data Space Used: 109.6 MB
 Data Space Total: 1.539 TB
 Data Space Available: 1.539 TB
```

Version 2015.09.c and earlier

Amazon ECS-optimized AMIs from version `2015.09.c` and earlier launch with a single 30 GiB volume that is attached at `/dev/xvda` and mounted as the root of the file system. This volume shares the operating system and all Docker images and metadata. You can determine the available storage on your container instance with standard storage information commands (such as **df -h**).

There is no practical way to add storage (that Docker can use) to instances launched from these AMIs without stopping them, so if you find that your container instances need more storage than the default 30 GiB, you should terminate each instance and launch another in its place with the latest Amazon ECS-optimized AMI and a large enough data storage volume.

Launching an Amazon ECS Container Instance

You can launch an Amazon ECS container instance using the AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 8). After you've launched your instance, you can use it to run tasks.

To launch a container instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. From the navigation bar, select the region to use.

   > **Note**
   > Amazon ECS is available in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
EU (Frankfurt)	eu-central-1
Asia Pacific (Tokyo)	ap-northeast-1

Region Name	Region
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

3. From the console dashboard, choose **Launch Instance**.

4. On the **Choose an Amazon Machine Image (AMI)** page, choose **Community AMIs**.

5. Choose an AMI for your container instance. You can choose the Amazon ECS-optimized AMI, or another operating system, such as CoreOS or Ubuntu. If you do not choose the Amazon ECS-optimized AMI, you need to follow the procedures in Installing the Amazon ECS Container Agent (p. 58).

   Note
   For Amazon ECS-specific CoreOS installation instructions, see https://coreos.com/docs/running-coreos/cloud-providers/ecs/.

   To use the Amazon ECS-optimized AMI, type **amazon-ecs-optimized** in the **Search community AMIs** field and press the **Enter** key. Choose **Select** next to the **amzn-ami-2016.09.b-amazon-ecs-optimized** AMI. The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI ID
us-east-1	ami-eca289fb
us-east-2	ami-446f3521
us-west-1	ami-9fadf8ff
us-west-2	ami-7abc111a
eu-west-1	ami-a1491ad2
eu-central-1	ami-54f5303b
ap-northeast-1	ami-9cd57ffd
ap-southeast-1	ami-a900a3ca
ap-southeast-2	ami-5781be34

6. On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. The t2.micro instance type is selected by default. The instance type that you select determines the resources available for your tasks to run on.

7. Choose **Next: Configure Instance Details**.

8. On the **Configure Instance Details** page, set the **Auto-assign Public IP** field depending on whether or not you want your instance to be accessible from the public Internet. If your instance should be accessible from the Internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If your instance should not be accessible from the Internet, set this field to **Disable**.

   Note
   Container instances need external network access to communicate with the Amazon ECS service endpoint, so if your container instances do not have public IP addresses, then they must use network address translation (NAT) or an HTTP proxy to provide this access. For more information, see NAT Instances in the *Amazon VPC User Guide* and HTTP Proxy Configuration (p. 80) in this guide.

9. On the **Configure Instance Details** page, select the `ecsInstanceRole` **IAM role** value that you created for your container instances in Setting Up with Amazon ECS (p. 8).

    Important

    If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent will not connect to your cluster. For more information, see Amazon ECS Container Instance IAM Role (p. 180).

10. (Optional) Configure your Amazon ECS container instance with user data, such as the agent environment variables from Amazon ECS Container Agent Configuration (p. 69); Amazon EC2 user data scripts are executed only once, when the instance is first launched.

    By default, your container instance launches into your default cluster. If you want to launch into your own cluster instead of the default, choose the **Advanced Details** list and paste the following script into the **User data** field, replacing *your_cluster_name* with the name of your cluster.

    ```
    #!/bin/bash
    echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
    ```

    Or, if you have an `ecs.config` file in Amazon S3 and have enabled Amazon S3 read-only access to your container instance role, choose the **Advanced Details** list and paste the following script into the **User data** field, replacing *your_bucket_name* with the name of your bucket to install the AWS CLI and write your configuration file at launch time.

    Note

    For more information about this configuration, see Storing Container Instance Configuration in Amazon S3 (p. 74).

    ```
    #!/bin/bash
    yum install -y aws-cli
    aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
    ```

11. Choose **Next: Add Storage**.

12. On the **Add Storage** page, configure the storage for your container instance.

    If you are using an Amazon ECS-optimized AMI prior to the **2015.09.d** version, your instance has a single volume that is shared by the operating system and Docker.

    If you are using the **2015.09.d** or later Amazon ECS-optimized AMI, your instance has two volumes configured. The **Root** volume is for the operating system's use, and the second Amazon EBS volume (attached to `/dev/xvdcz`) is for Docker's use.

    You can optionally increase or decrease the volume sizes for your instance to meet your application needs.

13. Choose **Review and Launch**.

14. On the **Review Instance Launch** page, under **Security Groups**, you'll see that the wizard created and selected a security group for you. Instead, select the security group that you created in Setting Up with Amazon ECS (p. 8) using the following steps:

    a. Choose **Edit security groups**.

    b. On the **Configure Security Group** page, ensure that the **Select an existing security group** option is selected.

    c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.

15. On the **Review Instance Launch** page, choose **Launch**.

16. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

When you are ready, select the acknowledgment field, and then choose **Launch Instances**.

17. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

18. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is `pending`. After the instance starts, its state changes to `running`, and it receives a public DNS name. (If the **Public DNS** column is hidden, choose the **Show/Hide** icon and select **Public DNS**.)

Connect to your Container Instance

To perform basic administrative tasks on your instance, such as updating or installing software or accessing diagnostic logs, you need to connect to the instance using SSH. To connect to your instance using SSH, your container instances must meet the following prerequisites:

- Your container instances need external network access to connect using SSH, so if your container instances are running in a private VPC, they need an SSH bastion instance to provide this access. For more information, see the Securely connect to Linux instances running in a private Amazon VPC blog post.

- Your container instances must have been launched with a valid Amazon EC2 key pair. Amazon ECS container instances have no password, and you use a key pair to log in using SSH. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance.

- SSH uses port 22 for communication. Port 22 must be open in your container instance security group for you to connect to your instance using SSH.

    Note
    The Amazon ECS console first-run experience creates a security group for your container instances without inbound access on port 22. If your container instances were launched from the console first-run experience, you need to add inbound access to port 22 on the security group used for those instances. For more information, see Authorizing Network Access to Your Instances in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your container instance

1. Find the public IP or DNS address for your container instance.

    a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
    b. Select the cluster that hosts your container instance.
    c. On the **Cluster** page, choose **ECS Instances**.
    d. On the **Container Instance** column, select the container instance to connect to.
    e. On the **Container Instance** page, record the **Public IP** or **Public DNS** for your instance.

2. Find the default username for your container instance AMI. The user name for instances launched with the Amazon ECS-optimized AMI is `ec2-user`. For Ubuntu AMIs, the default user name is `ubuntu`. For CoreOS, the default user name is `core`.

3. If you are using a Mac or Linux computer, connect to your instance with the following command, substituting the path to your private key and the public address for your instance:

```
$ ssh -i /path/to/my-key-pair.pem ec2-
user@ec2-198-51-100-1.compute-1.amazonaws.com
```

If you are using a Windows computer, see Connecting to Your Linux Instance from Windows Using PuTTY in the *Amazon EC2 User Guide for Linux Instances*.

Important
If you experience any issues connecting to your instance, see Troubleshooting
Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Using CloudWatch Logs with Container Instances

You can configure your container instances to send log information to CloudWatch Logs. This enables
you to view different logs from your container instances in one convenient location. This topic helps you
get started using CloudWatch Logs on your container instances that were launched with the Amazon
ECS-optimized AMI.

To send container logs from your tasks to CloudWatch Logs, see Using the awslogs Log
Driver (p. 107). For more information on CloudWatch Logs, see Monitoring Log Files in the *Amazon
CloudWatch User Guide*.

Topics

CloudWatch Logs IAM Policy

Before your container instances can send log data to CloudWatch Logs, you must create an IAM policy
to allow your container instances to use the CloudWatch Logs APIs, and then you must attach that
policy to the `ecsInstanceRole`.

To create the `ECS-CloudWatchLogs` IAM policy

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/
    iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. On the **Create Policy** page, choose **Create Your Own Policy**.
5. On the **Review Policy** page, enter the following information and choose **Create Policy**.

    a. In the **Policy Name** field, enter `ECS-CloudWatchLogs`.
    b. In the **Policy Document** field, paste the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
```

```
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

To attach the `ECS-CloudWatchLogs` policy to your `ecsInstanceRole`

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/
    iam/.
2. In the navigation pane, choose **Roles**.
3. Choose `ecsInstanceRole`. If the role does not exist, follow the procedures in Amazon ECS
    Container Instance IAM Role (p. 180) to create the role.
4. Choose the **Permissions** tab.
5. In the **Managed Policies** section, choose **Attach Policy**.
6. In the **Filter** box, type **ECS-CloudWatchLogs** to narrow the available policies to attach.
7. Check the box to the left of the **ECS-CloudWatchLogs** policy and choose **Attach Policy**.

Installing the CloudWatch Logs Agent

After you have added the `ECS-CloudWatchLogs` policy to your `ecsInstanceRole`, you can install
the CloudWatch Logs agent on your container instances.

> **Note**
> This procedure was written for the Amazon ECS-optimized AMI, and may not work on other
> operating systems. For information on installing the agent on other operating systems, see
> Getting Started with CloudWatch Logs in the *Amazon CloudWatch User Guide*.

To install the CloudWatch Logs agent

*   Run the following command to install the CloudWatch Logs agent.

```
[ec2-user ~]$ sudo yum install -y awslogs
```

After you have installed the agent, proceed to the next section to configure the agent.

Configuring and Starting the CloudWatch Logs Agent

The CloudWatch Logs agent configuration file (`/etc/awslogs/awslogs.conf`) describes the log
files to send to CloudWatch Logs. The agent configuration file's `[general]` section defines common
configurations that apply to all log streams, and you can add individual log stream sections for each
file on your container instances that you want to monitor. For more information, see CloudWatch Logs
Agent Reference in the *Amazon CloudWatch User Guide*.

The example configuration file below is configured for the Amazon ECS-optimized AMI, and it provides
log streams for several common log files:

`/var/log/dmesg`
    The message buffer of the Linux kernel.

`/var/log/messages`
    Global system messages.

`/var/log/docker`
> Docker daemon log messages.

`/var/log/ecs/ecs-init.log`
> Log messages from the `ecs-init` upstart job.

`/var/log/ecs/ecs-agent.log`
> Log messages from the Amazon ECS container agent.

`/var/log/ecs/audit.log`
> Log messages from the IAM roles for tasks credential provider.

You can use the example file below for your Amazon ECS container instances, but you must substitute the `{cluster}` and `{container_instance_id}` entries with the cluster name and container instance ID for each container instance so that the log streams are grouped by cluster name and separate for each individual container instance. The procedure that follows the example configuration file has steps to replace the cluster name and container instance ID placeholders.

```
[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/dmesg]
file = /var/log/dmesg
log_group_name = /var/log/dmesg
log_stream_name = {cluster}/{container_instance_id}

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %b %d %H:%M:%S

[/var/log/docker]
file = /var/log/docker
log_group_name = /var/log/docker
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%S.%f

[/var/log/ecs/ecs-init.log]
file = /var/log/ecs/ecs-init.log.*
log_group_name = /var/log/ecs/ecs-init.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

[/var/log/ecs/ecs-agent.log]
file = /var/log/ecs/ecs-agent.log.*
log_group_name = /var/log/ecs/ecs-agent.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

[/var/log/ecs/audit.log]
file = /var/log/ecs/audit.log.*
log_group_name = /var/log/ecs/audit.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ
```

To configure the CloudWatch Logs agent

1. Back up the existing CloudWatch Logs agent configuration file.

```
[ec2-user ~]$ sudo mv /etc/awslogs/awslogs.conf /etc/awslogs/
awslogs.conf.bak
```

2. Create a blank configuration file.

```
[ec2-user ~]$ sudo touch /etc/awslogs/awslogs.conf
```

3. Open the /etc/awslogs/awslogs.conf file with a text editor, and copy the example file above into it.

4. Install the **jq** JSON query utility.

```
[ec2-user ~]$ sudo yum install -y jq
```

5. Query the Amazon ECS introspection API to find the cluster name and set it to an environment variable.

```
[ec2-user ~]$ cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r
'. | .Cluster')
```

6. Replace the *{cluster}* placeholders in the file with the value of the environment variable you set in the previous step.

```
[ec2-user ~]$ sudo sed -i -e "s/{cluster}/$cluster/g" /etc/awslogs/
awslogs.conf
```

7. Query the Amazon ECS introspection API to find the container instance ID and set it to an environment variable.

```
[ec2-user ~]$ container_instance_id=$(curl -s http://localhost:51678/v1/
metadata | jq -r '. | .ContainerInstanceArn' | awk -F/ '{print $2}' )
```

8. Replace the *{container_instance_id}* placeholders in the file with the value of the environment variable you set in the previous step.

```
[ec2-user ~]$ sudo sed -i -e "s/{container_instance_id}/
$container_instance_id/g" /etc/awslogs/awslogs.conf
```

To configure the CloudWatch Logs agent region

By default, the CloudWatch Logs agent sends data to the us-east-1 region. If you would like to send your data to a different region, such as the region that your cluster is located in, you can set the region in the /etc/awslogs/awscli.conf file.

1. Open the /etc/awslogs/awscli.conf file with a text editor.

2. In the [default] section, replace us-east-1 with the region where you want to view log data.

3. Save the file and exit your text editor.

To start the CloudWatch Logs agent

1. Start the CloudWatch Logs agent with the following command.

```
[ec2-user ~]$ sudo service awslogs start
```

```
Starting awslogs:                                               [  OK  ]
```

2. Use the **chkconfig** command to ensure that the CloudWatch Logs agent starts at every system boot.

```
[ec2-user ~]$ sudo chkconfig awslogs on
```

Viewing CloudWatch Logs

After you have given your container instance role the proper permissions to send logs to CloudWatch Logs, and you have configured and started the agent, your container instance should be sending its log data to CloudWatch Logs. You can view and search these logs in the AWS Management Console.

Note
New instance launches may take a few minutes to send data to CloudWatch Logs.

To view your CloudWatch Logs data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. Choose **Logs** in the left navigation.

3. You should see the log groups you configured in Configuring and Starting the CloudWatch Logs Agent (p. 44).

Log Groups

Create Metric Filter	Actions ▾

Filter: Log Group Name Prefix ✕

	Log Groups
☐	/var/log/dmesg
☐	/var/log/docker
☐	/var/log/ecs/ecs-agent.log
☐	/var/log/ecs/ecs-init.log
☐	/var/log/messages
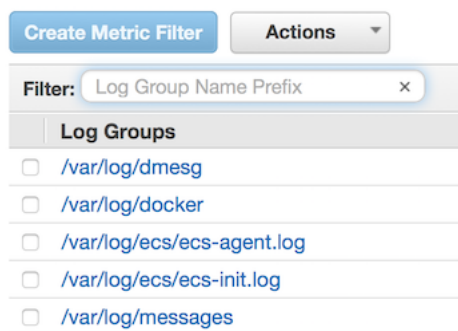
4. Choose a log group that you would like to view.

5. Choose a log stream to view. The streams are identified by the cluster name and container instance ID that sent the logs.

Configuring CloudWatch Logs at Launch with User Data

When you launch an Amazon ECS container instance in Amazon EC2, you have the option of passing user data to the instance that can be used to perform common automated configuration tasks and even run scripts after the instance starts. You can pass several types of user data to instances, including shell scripts, `cloud-init` directives, and Upstart jobs. You can also pass this data into the launch wizard as plain text, as a file (this is useful for launching instances via the command line tools), or as base64-encoded text (for API calls).

The example user data block below performs the following tasks:

- Installs the `awslogs` package, which contains the CloudWatch Logs agent
- Installs the **jq** JSON query utility
- Writes the configuration file for the CloudWatch Logs agent and configures the region to send data to (the region that the container instance is located)
- Gets the cluster name and container instance ID after the Amazon ECS container agent starts and then writes those values to the CloudWatch Logs agent configuration file log streams
- Starts the CloudWatch Logs agent
- Configures the CloudWatch Logs agent to start at every system boot

```
Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
MIME-Version: 1.0
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
# Install awslogs and the jq JSON parser
yum install -y awslogs jq
```

```
# Inject the CloudWatch Logs configuration file contents
cat > /etc/awslogs/awslogs.conf <<- EOF
[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/dmesg]
file = /var/log/dmesg
log_group_name = /var/log/dmesg
log_stream_name = {cluster}/{container_instance_id}

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %b %d %H:%M:%S

[/var/log/docker]
file = /var/log/docker
log_group_name = /var/log/docker
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%S.%f

[/var/log/ecs/ecs-init.log]
file = /var/log/ecs/ecs-init.log.*
log_group_name = /var/log/ecs/ecs-init.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

[/var/log/ecs/ecs-agent.log]
file = /var/log/ecs/ecs-agent.log.*
log_group_name = /var/log/ecs/ecs-agent.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

[/var/log/ecs/audit.log]
file = /var/log/ecs/audit.log.*
log_group_name = /var/log/ecs/audit.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

EOF

--==BOUNDARY==
MIME-Version: 1.0
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
# Set the region to send CloudWatch Logs data to (the region where the
 container instance is located)
region=$(curl 169.254.169.254/latest/meta-data/placement/availability-zone |
 sed s'/.$//')
sed -i -e "s/region = us-east-1/region = $region/g" /etc/awslogs/awscli.conf

--==BOUNDARY==
MIME-Version: 1.0
Content-Type: text/upstart-job; charset="us-ascii"

#upstart-job
description "Configure and start CloudWatch Logs agent on Amazon ECS
 container instance"
```

```
author "Amazon Web Services"
start on started ecs

script
 exec 2>>/var/log/ecs/cloudwatch-logs-start.log
 set -x

 until curl -s http://localhost:51678/v1/metadata
 do
  sleep 1
 done

 # Grab the cluster and container instance ARN from instance metadata
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster')
 container_instance_id=$(curl -s http://localhost:51678/v1/metadata | jq -r
 '. | .ContainerInstanceArn' | awk -F/ '{print $2}' )

 # Replace the cluster name and container instance ID placeholders with the
 actual values
 sed -i -e "s/{cluster}/$cluster/g" /etc/awslogs/awslogs.conf
 sed -i -e "s/{container_instance_id}/$container_instance_id/g" /etc/awslogs/
awslogs.conf

 service awslogs start
 chkconfig awslogs on
end script
--==BOUNDARY==--
```

If you have created the `ECS-CloudWatchLogs` policy and attached it to your `ecsInstanceRole`
as described in CloudWatch Logs IAM Policy (p. 43), then you can add the above user data block
to any container instances that you launch manually, or you can add it to an Auto Scaling launch
configuration, and your container instances that are launched with this user data will begin sending
their log data to CloudWatch Logs as soon as they launch. For more information, see Launching an
Amazon ECS Container Instance (p. 39).

Managing Container Instances Remotely

You can use the Amazon EC2 Run Command feature to securely and remotely manage the
configuration of your Amazon ECS container instances. Run Command provides a simple way of
performing common administrative tasks without having to log on locally to the instance. You can
manage configuration changes across your clusters by simultaneously executing commands on
multiple container instances. Run Command reports the status and results of each command.

Here are some examples of the types of tasks you can perform with Run Command:

- Install or uninstall packages
- Perform security updates
- Clean up Docker images
- Stop or start services
- View system resources
- View log files
- Perform file operations

This topic covers basic installation of Run Command on the Amazon ECS-optimized AMI and a few simple use cases, but it is by no means exhaustive. For more information about Run Command, see Manage Amazon EC2 Instances Remotely in the *Amazon EC2 User Guide for Linux Instances*.

Topics

Run Command IAM Policy

Before you can send commands to your container instances with Run Command, you must attach an IAM policy that allows access to the Amazon EC2 Simple Systems Manager (SSM) APIs to the `ecsInstanceRole`. The procedure below describes how to attach the `AmazonEC2RoleforSSM` managed policy to your container instance role so that instances launched with this role can use Run Command.

To attach the `AmazonEC2RoleforSSM` policy to your `ecsInstanceRole`

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Choose `ecsInstanceRole`. If the role does not exist, follow the procedures in Amazon ECS Container Instance IAM Role (p. 180) to create the role.

4. Choose the **Permissions** tab.

5. In the **Managed Policies** section, choose **Attach Policy**.

6. For **Filter**, type **AmazonEC2RoleforSSM** to narrow the available policies to attach.

7. Select the check box for **AmazonEC2RoleforSSM** policy and choose **Attach Policy**.

Installing the SSM Agent on the Amazon ECS-optimized AMI

After you have attached the `AmazonEC2RoleforSSM` policy to your `ecsInstanceRole`, you can install the SSM agent on your container instances. The SSM agent processes Run Command requests and configures the instances that are specified in the request. Use the following procedures to install the SSM agent on your Amazon ECS-optimized AMI container instances.

To manually install the SSM agent on existing Amazon ECS-optimized AMI container instances

1. Connect to your container instance. (p. 42)

2. Install the SSM agent RPM. The SSM agent is available in all regions that Amazon ECS is available in, and each region has its own region-specific download URL; the example command below works for all regions that Amazon ECS supports, but you can avoid cross-region data transfer costs for the RPM download by substituting the region of your container instance.

```
[ec2-user ~]$ sudo yum install -y https://amazon-ssm-us-east-1.s3.amazonaws.com/latest/linux_amd64/amazon-ssm-agent.rpm
```

To install the SSM agent on new instance launches with Amazon EC2 user data

- Launch one or more container instances by following the procedure in Launching an Amazon ECS
  Container Instance (p. 39), but in Step 10 (p. 41), copy and paste the user data script below
  into the **User data** field. You can also add the commands from this user data script to another
  existing script that you may have to perform other tasks, such as setting the cluster name for the
  instance to register into.

  > **Note**
  > The user data script below installs the **jq** JSON parser and uses that to determine the
  > region of the container instance. Then it downloads and installs the SSM agent.

  ```
  #!/bin/bash
  # Install JQ JSON parser
  yum install -y jq

  # Get the current region from the instance metadata
  region=$(curl -s http://169.254.169.254/latest/dynamic/instance-identity/
  document | jq -r .region)

  # Install the SSM agent RPM
  yum install -y https://amazon-ssm-$region.s3.amazonaws.com/latest/
  linux_amd64/amazon-ssm-agent.rpm
  ```

Using Run Command

After you have attached the `AmazonEC2RoleforSSM` policy to your `ecsInstanceRole`, and installed
the SSM agent on your container instances, you can start using Run Command to send commands
to your container instances. The following topics in the *Amazon EC2 User Guide for Linux Instances*
explain how to run commands and shell scripts on your instances and view the resulting output:

- Running Shell Scripts with Run Command
- Viewing Command Output in the Amazon EC2 Console

For more information about Run Command, see Manage Amazon EC2 Instances Remotely in the
Amazon EC2 User Guide for Linux Instances.

Example: To update container instance software with Run Command

One of the most common use cases for Run Command on Amazon ECS container instances is to
update the instance software on your entire fleet of container instances at once, simultaneously.

1. Attach the `AmazonEC2RoleforSSM` policy to your `ecsInstanceRole`. (p. 51)
2. Install the SSM agent on your container instances. For more information, see Installing the SSM
   Agent on the Amazon ECS-optimized AMI (p. 51).
3. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
4. In the left navigation, choose **Commands**.
5. Choose **Run a command**.
6. For **Command document**, choose **AWS-RunShellScript**.
7. In the **Target instances** section, choose **Select instances** and check the container instances to
   send the update command to.
8. In the **Commands** section, enter the command or commands to send to your container instances.
   In this example, the command below updates the instance software, but you can send any
   command that you want.

```
yum update -y
```

9. Choose **Run** to send the command to the specified instances.
10. (Optional) Choose **View result** to see the results of your command.
11. (Optional) Choose a command from the list of recent commands to view the command output.

12. (Optional) Choose the **Output** tab, and then choose **View Output**. The image below shows a snippet of the container instance output for the **yum update** command.

   Note
   Unless you configure a command to save the output to an Amazon S3 bucket, then the command output is truncated at 2500 characters.

Starting a Task at Container Instance Launch Time

Depending on your application architecture design, you may need to run a specific container on every container instance to deal with operations or security concerns such as monitoring, security, metrics, service discovery, or logging.

One method you can use to accomplish this goal is to configure your container instances to call the **docker run** command with the user data script at launch or in some init system such as Upstart or **systemd**. While this works, it has some disadvantages because Amazon ECS has no knowledge of the container and cannot monitor the CPU, memory, ports, or any other resources used. To ensure that Amazon ECS can properly account for all task resources, you should create a task definition for the container you want to run on your container instances, and use Amazon ECS to place the task at launch time with EC2 user data.

The EC2 user data script in the following procedure uses the Amazon ECS introspection API to identify the container instance and then it uses the AWS CLI and the **start-task** command to run a specified task on itself during start up.

To start a task at container instance launch time

1. If you have not done so already, create a task definition with the container you want to run on your container instance at launch by following the procedures in Creating a Task Definition (p. 84).

2. Modify your `ecsInstanceRole` IAM role to add permissions for the `StartTask` API operation. For more information, see Amazon ECS Container Instance IAM Role (p. 180).

    a. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

    b. In the navigation pane, choose **Roles**.

    c. Choose the `ecsInstanceRole`. If the role does not exist, use the procedure in Amazon ECS Container Instance IAM Role (p. 180) to create the role and return to this procedure. If the role does exist, select the role to view the attached policies.

    d. In the **Inline Policies** section, choose **Create Role Policy**.

    e. On the **Set Permissions** page, choose **Custom Policy** and then choose **Select**.

    f. In the **Policy Name** field, enter `StartTask`.

    g. In the **Policy Document** field, copy and paste the following policy and choose **Apply Policy**.

    ```
    {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Action": [
             "ecs:StartTask"
           ],
           "Resource": "*"
         }
       ]
    }
    ```

3. Launch one or more container instances by following the procedure in Launching an Amazon ECS Container Instance (p. 39), but in Step 10 (p. 41), copy and paste the MIME multipart user data script below into the **User data** field, substituting *your_cluster_name* with the cluster you want the container instance to register into and *my_task_def* and the task definition you want to run on the instance at launch.

    > **Note**
    > The MIME multipart content below uses a shell script to set configuration values and install packages, and an Upstart job to start the task after the **ecs** service is running and the introspection API is available.

    ```
    Content-Type: multipart/mixed; boundary="==BOUNDARY=="
    MIME-Version: 1.0
    ```

```
--==BOUNDARY==
MIME-Version: 1.0
Content-Type: text/text/x-shellscript; charset="us-ascii"

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name

# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

# Install the AWS CLI and the jq JSON parser
yum install -y aws-cli jq
--==BOUNDARY==
MIME-Version: 1.0
Content-Type: text/text/upstart-job; charset="us-ascii"

#upstart-job
description "Amazon EC2 Container Service (start task on instance boot)"
author "Amazon Web Services"
start on started ecs

script
 exec 2>>/var/log/ecs/ecs-start-task.log
 set -x
 until curl -s http://localhost:51678/v1/metadata
 do
   sleep 1
 done

 # Grab the container instance ARN and AWS region from instance metadata
 instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F/ '{print $NF}' )
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .Cluster' | awk -F/ '{print $NF}' )
 region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F: '{print $4}')

 # Specify the task definition to run at launch
 task_definition=my_task_def

 # Run the AWS CLI start-task command to start your task on this container
 instance
 aws ecs start-task --cluster $cluster --task-definition $task_definition
 --container-instances $instance_arn --started-by $instance_arn --region
 $region
end script
--==BOUNDARY==--
```

4. Verify that your container instances launch into the correct cluster and that your tasks have started.

    a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

    b. From the navigation bar, choose the region that your cluster is in.

    c. In the navigation pane, choose **Clusters**.

    d. Choose the cluster that hosts your container instances.

    e. On the **Cluster** page, choose the **Tasks** tab.

Each container instance you launched should have your task running on it, and the container instance ARN should be in the **Started By** column.

If you do not see your tasks, you can log into your container instances with SSH and check the `/var/log/ecs/ecs-start-task.log` file for debugging information.

Deregister a Container Instance

When you are finished with a container instance, you can deregister it from your cluster.

Following deregistration, the container instance is no longer able to accept new tasks. If you have tasks running on the container instance when you deregister it, these tasks remain running until you terminate the instance or the tasks stop through some other means, but they are orphaned (no longer monitored or accounted for by Amazon ECS). If an orphaned task on your container instance is part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different container instance if possible. Any containers in orphaned service tasks that are registered with a Classic Load Balancer or an Application Load Balancer target group are deregistered, and they will begin connection draining according to the settings on the load balancer or target group.

If you intend to use the container instance for some other purpose after deregistration, you should stop all of the tasks running on the container instance before deregistration to avoid any orphaned tasks from consuming resources.

Important
Because each container instance has unique state information, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances with the latest Amazon ECS-optimized AMI in the new cluster. For

more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances* and Launching an Amazon ECS Container Instance (p. 39).

Deregistering a container instance removes the instance from a cluster, but it does not terminate the EC2 instance; if you are finished using the instance, be sure to terminate it in the Amazon EC2 console to stop billing. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Note
If you terminate a running container instance with a connected Amazon ECS container agent, the agent automatically deregisters the instance from your cluster (stopped container instances or instances with disconnected agents are not automatically deregistered when terminated).

To deregister a container instance

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, choose the region that your container instance is registered in.
3. In the navigation pane, choose **Clusters**.
4. Choose the cluster that hosts your container instance.
5. On the **Cluster :** *name* page, choose the **ECS Instances** tab.
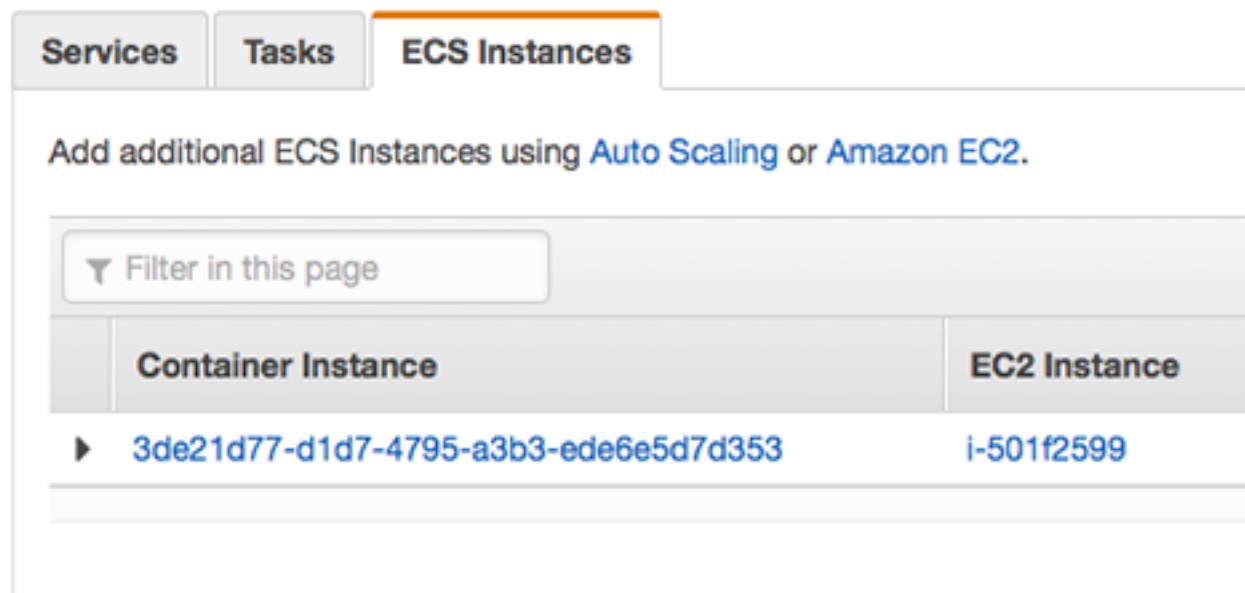
6. Choose the container instance ID that you want to deregister.
7. On the **Container Instance :** *id* page, choose **Deregister**.
8. Review the deregistration message, and choose **Yes, Deregister** to deregister the container instance.
9. If you are finished with the container instance, you should terminate the underlying Amazon EC2 instance. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Note
If your instance is maintained by an Auto Scaling group or AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack; otherwise, the Auto Scaling group will recreate the instance after you terminate it.

Amazon ECS Container Agent

The Amazon ECS container agent allows container instances to connect to your cluster. The Amazon ECS container agent is included in the Amazon ECS-optimized AMI, but you can also install it on any EC2 instance that supports the Amazon ECS specification. The Amazon ECS container agent is only supported on EC2 instances.

Note
The source code for the Amazon ECS container agent is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently provide support for running modified copies of this software.

Topics

Installing the Amazon ECS Container Agent

If your container instance was not launched from an AMI that includes the Amazon ECS container agent, you can install it using the following procedure.

Note
The Amazon ECS container agent is included in the Amazon ECS-optimized AMI and does not require installation.

To install the Amazon ECS container agent on an Amazon Linux EC2 Instance

1. Launch an Amazon Linux instance with an IAM role that allows access to Amazon ECS. For more information, see Amazon ECS Container Instance IAM Role (p. 180).
2. Connect to your instance.

3. Install the `ecs-init` package. For more information on `ecs-init`, you can view the source code on GitHub.

```
[ec2-user ~]$ sudo yum install -y ecs-init
```

4. Start the Docker daemon.

```
[ec2-user ~]$ sudo service docker start
Starting cgconfig service:                              [  OK  ]
Starting docker:                                    [  OK  ]
```

5. Start the `ecs-init` upstart job.

```
[ec2-user ~]$ sudo start ecs
ecs start/running, process 2804
```

6. (Optional) You can verify that the agent is running and see some information on your new container instance with the agent introspection API. For more information, see the section called "Amazon ECS Container Agent Introspection" (p. 78).

```
[ec2-user ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.13.1 (efe53c6)"
}
```

To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance

1. Launch an EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see Amazon ECS Container Instance IAM Role (p. 180).

2. Connect to your instance.

3. Install Docker on your instance. Amazon ECS requires a minimum Docker version of 1.5.0 (version 1.11.2 is recommended), and the default Docker versions in many system package managers, such as **yum** or **apt-get** do not meet this minimum requirement. For information about installing the latest Docker version on your particular Linux distribution, go to https://docs.docker.com/engine/installation/.

    Note
    The Amazon Linux AMI always includes the recommended version of Docker for use with Amazon ECS. You can install Docker on Amazon Linux with the **sudo yum install docker -y** command.

4. Check your Docker version to verify that your system meets the minimum version requirement.

```
ubuntu:~$ sudo docker version
Client version: 1.4.1
Client API version: 1.16
Go version (client): go1.3.3
Git commit (client): 5bc2ff8
OS/Arch (client): linux/amd64
Server version: 1.4.1
Server API version: 1.16
Go version (server): go1.3.3
Git commit (server): 5bc2ff8
```

In this example, the Docker version is `1.4.1`, which is below the minimum version of 1.5.0. This instance needs to upgrade its Docker version before proceeding. For information about installing the latest Docker version on your particular Linux distribution, go to https://docs.docker.com/engine/installation/.

5. Run the following commands on your container instance to enable IAM roles for tasks. For more information, see IAM Roles for Tasks (p. 186).

```
sysctl -w net.ipv4.conf.all.route_localnet=1
iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT
 --to-destination 127.0.0.1:51679
iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j
 REDIRECT --to-ports 51679
```

6. Pull and run the latest Amazon ECS container agent on your container instance.

   Note
   You should use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted if it exits. For more information, see Automatically start containers and Restart policies in the Docker documentation. The Amazon ECS-optimized AMI uses the `ecs-init` RPM for this purpose, and you can view the source code for this RPM on GitHub.

   The following example agent run command is broken into separate lines to show each option.

   - The `--env=ECS_CLUSTER=`*`cluster_name`* option is not required if you want to register into your default cluster.
   - You can optionally store your agent environment variables in a file (which can be downloaded to your container instances from Amazon S3 at launch time using EC2 user data) and pass them all at one time with the `--env-file` *`path_to_env_file`* option. This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see Storing Container Instance Configuration in Amazon S3 (p. 74) and Private Registry Authentication (p. 76).
   - If your task definitions specify log configuration options for a particular log driver, the Amazon ECS container agent running on your container instances must register the specified log driver with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable. For example, to register a container instance with the `json-file` and `awslogs` logging drivers, add the `--env=ECS_AVAILABLE_LOGGING_DRIVERS=`*`["json-file","awslogs"]`* option to the **docker run** command below. For more information, see Amazon ECS Container Agent Configuration (p. 69).
   - Operating systems with SELinux enabled require the `--privileged` option in your **docker run** command. In addition, for SELinux-enabled container instances, we recommend that you add the `:Z` option to the `/log` and `/data` volume mounts; however, the host mounts for these volumes must exist before you run the command or you will receive a `no such file or directory` error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:
     - Create the host volume mount points on your container instance.

       ```
       $ sudo mkdir -p /var/log/ecs
       $ sudo mkdir -p /var/lib/ecs/data
       ```

     - Add the `--privileged` option to the **docker run** command below.
     - Append the `:Z` option to the `/log` and `/data` container volume mounts (for example, `--volume=/var/log/ecs/:/log:Z`) to the **docker run** command below.

   For more information on these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 69).

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run/docker.sock:/var/run/docker.sock \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--net=host \
--env=ECS_LOGFILE=/log/ecs-agent.log \
--env=ECS_LOGLEVEL=info \
--env=ECS_DATADIR=/data \
--env=ECS_CLUSTER=cluster_name \
--env=ECS_ENABLE_TASK_IAM_ROLE=true \
--env=ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true \
amazon/amazon-ecs-agent:latest
```

Note

If you receive an `Error response from daemon: Cannot start container` message, you can delete the failed container with the **sudo docker rm ecs-agent** command and try running the agent again.

Amazon ECS Container Agent Versions

Each Amazon ECS container agent version supports a different feature set and provides bug fixes from previous versions. When possible, we always recommend using the latest version of the Amazon ECS container agent. To update your container agent to the latest version, see Updating the Amazon ECS Container Agent (p. 63).

Launching your container instances from the most recent Amazon ECS-optimized AMI ensures that you receive the current container agent version. To launch a container instance with the latest Amazon ECS-optimized AMI, see Launching an Amazon ECS Container Instance (p. 39).

To install the latest version of the Amazon ECS container agent on another operating system, see Installing the Amazon ECS Container Agent (p. 58). The table in Amazon ECS-optimized AMI Container Agent Versions (p. 61) shows the Docker version that is tested on Amazon Linux for each agent version.

To see which features and enhancements are included with each agent release, see https://github.com/aws/amazon-ecs-agent/releases.

Amazon ECS-optimized AMI Container Agent Versions

The Amazon ECS-optimized AMI comes prepackaged with the Amazon ECS container agent, Docker, and the `ecs-init` service that controls the starting and stopping of the agent at boot and shutdown. The following table lists the container agent version, the `ecs-init` version, and the Docker version that is tested and packaged with each Amazon ECS-optimized AMI.

Note

As new Amazon ECS-optimized AMIs and Amazon ECS agent versions are released, older versions are still available for launch in Amazon EC2; however, we encourage you to update to the latest version (p. 63) of the Amazon ECS agent and to keep your container instance software up to date. If you request support for an older version of the Amazon ECS agent through AWS Support, you may be asked to move to the latest version as a part of the support process.

Amazon ECS-optimized AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2016.09.b	1.13.1	1.11.2	1.13.1-1
2016.09.a	1.13.0	1.11.2	1.13.0-1
2016.03.j	1.13.0	1.11.2	1.13.0-1
2016.03.i	1.12.2	1.11.2	1.12.2-1
2016.03.h	1.12.1	1.11.2	1.12.1-1
2016.03.g	1.12.0	1.11.2	1.12.0-1
2016.03.f	1.11.1	1.11.2	1.11.1-1
2016.03.e	1.11.0	1.11.2	1.11.0-1
2016.03.d	1.10.0	1.11.1	1.10.0-1
2016.03.c	1.10.0	1.11.1	1.10.0-1
2016.03.b	1.9.0	1.9.1	1.9.0-1
2016.03.a	1.8.2	1.9.1	1.8.2-1
2015.09.g	1.8.1	1.9.1	1.8.1-1
2015.09.f	1.8.0	1.9.1	1.8.0-1
2015.09.e	1.7.1	1.9.1	1.7.1-1
2015.09.d	1.7.1	1.9.1	1.7.1-1
2015.09.c	1.7.0	1.7.1	1.7.0-1
2015.09.b	1.6.0	1.7.1	1.6.0-1
2015.09.a	1.5.0	1.7.1	1.5.0-1
2015.03.g	1.4.0	1.7.1	1.4.0-2
2015.03.f	1.4.0	1.6.2	1.4.0-1
2015.03.e	1.3.1	1.6.2	1.3.1-1
2015.03.d	1.2.1	1.6.2	1.2.0-2
2015.03.c	1.2.0	1.6.2	1.2.0-1
2015.03.b	1.1.0	1.6.0	1.0-3
2015.03.a	1.0.0	1.5.0	1.0-1

For more information about the Amazon ECS-optimized AMI, including AMI IDs for the latest version in each region, see Amazon ECS-optimized AMI (p. 34).

Updating the Amazon ECS Container Agent

Occasionally, you may need to update the Amazon ECS container agent to pick up bug fixes and new features. Updating the Amazon ECS container agent does not interrupt running tasks or services on the container instance. The process for updating the agent differs depending on whether your container instance was launched with the Amazon ECS-optimized AMI or another operating system.

Topics

Checking your Amazon ECS Container Agent Version

You can check the version of the container agent that is running on your container instances to see if you need to update it. The container instance view in the Amazon ECS console provides the agent version. Use the following procedure to check your agent version.

To check if your Amazon ECS container agent is running the latest version in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, select the cluster that hosts the container instance or instances to check.
3. On the **Cluster :** *cluster_name* page, choose **ECS Instances**.
4. Note the **Agent version** column for your container instances. If you are using an outdated agent version on any of your container instances, the console alerts you with a message and flags the outdated agent version.

If your agent version is 1.13.1, you are running the latest container agent. If your agent version is below 1.13.1, you can update your container agent with the following procedures:

- If your container instance is running the Amazon ECS-optimized AMI, see Updating the Amazon ECS Container Agent on the Amazon ECS-optimized AMI (p. 65).

- If your container instance is not running the Amazon ECS-optimized AMI, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs) (p. 67).

> **Important**
> To update the Amazon ECS agent version from versions prior to v1.0.0 on your Amazon ECS-optimized AMI we recommend that you terminate your current container instance and launch a new instance with the most recent Amazon ECS-Optimized AMI. Any container instances that use a preview version of the Amazon ECS-optimized AMI should be retired and replaced with the most recent AMI. For more information, see Launching an Amazon ECS Container Instance (p. 39).

You can also use the Amazon ECS container agent introspection API to check the agent version from the container instance itself. For more information, see Amazon ECS Container Agent Introspection (p. 78).

To check if your Amazon ECS container agent is running the latest version with the introspection API

1. Log into your container instance via SSH.

2. Query the introspection API.

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python -mjson.tool
{
    "Cluster": "default",
    "ContainerInstanceArn": "arn:aws:ecs:us-
west-2:<aws_account_id>:container-instance/4d3910c1-27c8-410c-b1df-
f5d06fab4305",
    "Version": "Amazon ECS Agent - v1.13.1 (efe53c6)"
}
```

Note
The introspection API added `Version` information in the version v1.0.0 of the Amazon
ECS container agent. If `Version` is not present when querying the introspection API, or
the introspection API is not present in your agent at all, then the version you are running
is v0.0.3 or earlier, and you should update it.

Updating the Amazon ECS Container Agent on the Amazon ECS-optimized AMI

If you are using the Amazon ECS-optimized AMI, you have several options to get the latest version of
the Amazon ECS container agent (shown in order of recommendation):

- Terminate your current container instances and launch the latest version of the Amazon ECS-
  optimized AMI (either manually or by updating your Auto Scaling launch configuration with the latest
  AMI). This provides a fresh container instance with the most current tested and validated versions
  of Amazon Linux, Docker, `ecs-init`, and the Amazon ECS container agent. For more information,
  see Amazon ECS-optimized AMI (p. 34).
- Connect to the instance with SSH and update the `ecs-init` package (and its dependencies) to the
  latest version. This operation provides the most current tested and validated versions of Docker and
  `ecs-init` that are available in the Amazon Linux repositories and the latest version of the Amazon
  ECS container agent. For more information, see To update the `ecs-init` package on the Amazon
  ECS-optimized AMI (p. 65).
- Update the container agent with the `UpdateContainerAgent` API operation, either through the
  console or with the AWS CLI or AWS SDKs. For more information, see Updating the Amazon ECS
  Container Agent with the `UpdateContainerAgent` API Operation (p. 66).

To update the `ecs-init` package on the Amazon ECS-optimized AMI

1. Log into your container instance via SSH. For more information, see Connect to your Container
   Instance (p. 42).
2. Update the `ecs-init` package with the following command.

```
[ec2-user ~]$ sudo yum update -y ecs-init
```

Note
The `ecs-init` package and the Amazon ECS container agent are updated immediately;
however, newer versions of Docker are not loaded until the Docker daemon is restarted
(either by rebooting the instance, or by running **sudo service docker restart** to restart
Docker and then **sudo start ecs** to restart the container agent).

Updating the Amazon ECS Container Agent with the `UpdateContainerAgent` API Operation

Important

This update process is only supported on the Amazon ECS-optimized AMI. For container instances that are running other operating systems, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs) (p. 67).

To update the Amazon ECS agent version from versions prior to v1.0.0 on your Amazon ECS-optimized AMI we recommend that you terminate your current container instance and launch a new instance with the most recent Amazon ECS-Optimized AMI. Any container instances that use a preview version of the Amazon ECS-optimized AMI should be retired and replaced with the most recent AMI. For more information, see Launching an Amazon ECS Container Instance (p. 39).

The update process begins when you request an agent update, either through the console or with the AWS CLI or AWS SDKs. Amazon ECS checks your current agent version against the latest available agent version, and if an update is possible, the update process progresses as shown in the flow chart below. If an update is not available, for example, if the agent is already running the most recent version, then a `NoUpdateAvailableException` is returned.
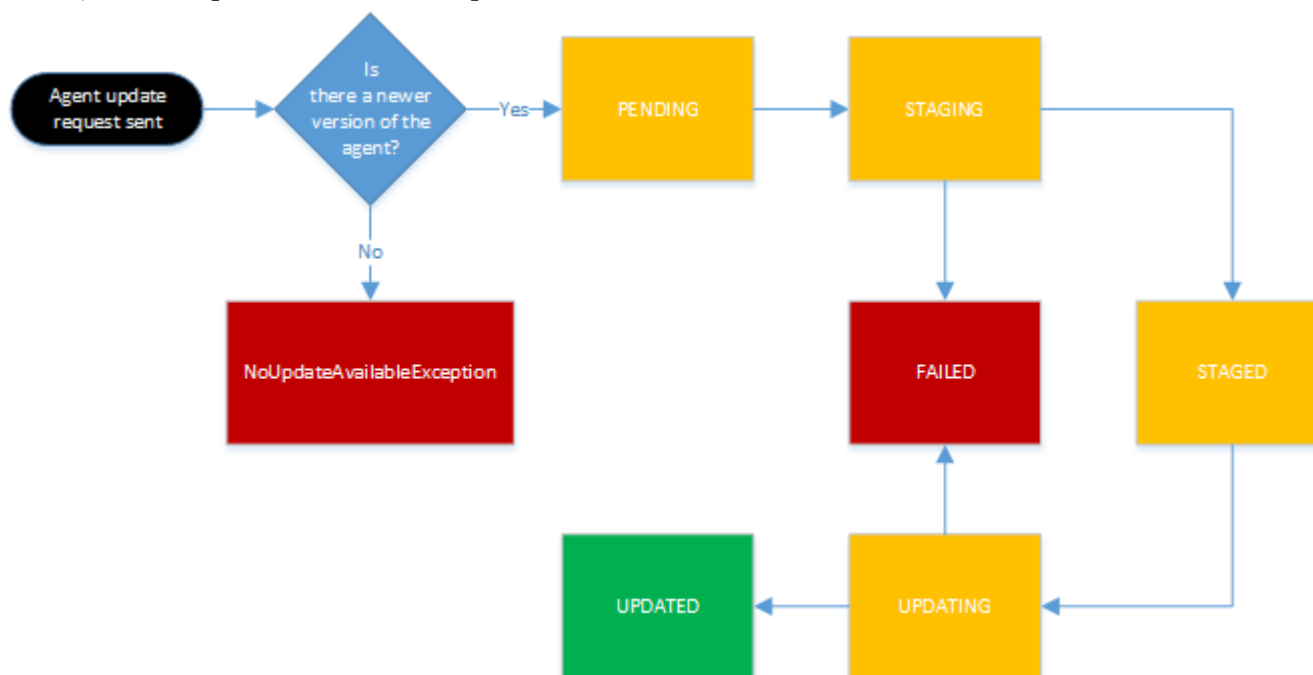
The stages in the update process shown above are as follows:

PENDING

An agent update is available, and the update process has started.

STAGING

The agent has begun downloading the agent update. If the agent cannot download the update, or if the contents of the update are incorrect or corrupted, then the agent sends a notification of the failure and the update transitions to the `FAILED` state.

STAGED

The agent download has completed and the agent contents have been verified.

UPDATING

The `ecs-init` service is restarted and it picks up the new agent version. If the agent is for some reason unable to restart, the update transitions to the `FAILED` state; otherwise, the agent signals Amazon ECS that the update is complete.

To update the Amazon ECS container agent on the Amazon ECS-optimized AMI in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, select the cluster that hosts the container instance or instances to check.
3. On the **Cluster : *cluster_name*** page, choose **ECS Instances**.
4. Select the container instance to update.
5. On the Container Instance page, choose **Update agent**.

To update the Amazon ECS container agent on the Amazon ECS-optimized AMI with the AWS CLI

*   Use the following command to update the Amazon ECS container agent on your container instance:

```
$ aws ecs update-container-agent --cluster cluster_name --container-
instance container_instance_id
```

Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs)

To manually update the Amazon ECS container agent (for non-Amazon ECS-optimized AMIs)

1. Log into your container instance via SSH.
2. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state.

```
[ec2-user ~]$ docker inspect ecs-agent | grep ECS_DATADIR
            "ECS_DATADIR=/data",
```

>   **Important**
>   If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before updating your agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can update them while tasks are running without issues.

3. Stop the Amazon ECS container agent.

```
[ec2-user ~]$ docker stop ecs-agent
ecs-agent
```

4. Delete the agent container.

```
[ec2-user ~]$ docker rm ecs-agent
ecs-agent
```

5. Run the following commands on your container instance to enable IAM roles for tasks. For more information, see IAM Roles for Tasks (p. 186).

```
sysctl -w net.ipv4.conf.all.route_localnet=1
iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT
 --to-destination 127.0.0.1:51679
iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j
 REDIRECT --to-ports 51679
```

6. Pull the latest Amazon ECS container agent image from Docker Hub.

```
[ec2-user ~]$ docker pull amazon/amazon-ecs-agent:latest
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

7. Run the latest Amazon ECS container agent on your container instance.

    Note
    You should use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted if it exits. For more information, see Automatically start containers and Restart policies in the Docker documentation. The Amazon ECS-optimized AMI uses the `ecs-init` RPM for this purpose, and you can view the source code for this RPM on GitHub.

    The following example agent run command is broken into separate lines to show each option.

    *   The `--env=ECS_CLUSTER=`*cluster_name* option is not required if you want to register into your default cluster.

    *   You can optionally store your agent environment variables in a file (which can be downloaded to your container instances from Amazon S3 at launch time using EC2 user data) and pass them all at one time with the `--env-file` *path_to_env_file* option. This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see Storing Container Instance Configuration in Amazon S3 (p. 74) and Private Registry Authentication (p. 76).

    *   If your task definitions specify log configuration options for a particular log driver, the Amazon ECS container agent running on your container instances must register the specified log driver with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable. For example, to register a container instance with the `json-file` and `awslogs` logging drivers, add the `--env=ECS_AVAILABLE_LOGGING_DRIVERS=`*["json-file","awslogs"]* option to the **docker run** command below. For more information, see Amazon ECS Container Agent Configuration (p. 69).

    *   Operating systems with SELinux enabled require the `--privileged` option in your **docker run** command. In addition, for SELinux-enabled container instances, we recommend that you add the `:Z` option to the `/log` and `/data` volume mounts; however, the host mounts for these volumes must exist before you run the command or you will receive a `no such file or directory` error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:

        *   Create the host volume mount points on your container instance.

            ```
            $ sudo mkdir -p /var/log/ecs
            $ sudo mkdir -p /var/lib/ecs/data
            ```

        *   Add the `--privileged` option to the **docker run** command below.

- Append the `:Z` option to the `/log` and `/data` container volume mounts (for example, `--volume=/var/log/ecs/:/log:Z`) to the **docker run** command below.

For more information about these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 69).

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run/docker.sock:/var/run/docker.sock \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--net=host \
--env=ECS_LOGFILE=/log/ecs-agent.log \
--env=ECS_LOGLEVEL=info \
--env=ECS_DATADIR=/data \
--env=ECS_CLUSTER=cluster_name \
--env=ECS_ENABLE_TASK_IAM_ROLE=true \
--env=ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true \
amazon/amazon-ecs-agent:latest
```

> **Note**
> If you receive an `Error response from daemon: Cannot start container` message, you can delete the failed container with the **sudo docker rm ecs-agent** command and try running the agent again.

Amazon ECS Container Agent Configuration

The Amazon ECS container agent supports a number of configuration options, most of which should be set through environment variables. The following environment variables are available, and all of them are optional.

If your container instance was launched with the Amazon ECS-optimized AMI, you can set these environment variables in the `/etc/ecs/ecs.config` file and then restart the agent. You can also create a custom `ecs.config` file that contains these configuration variables and store it in Amazon S3 for container instances to download at run time using Amazon EC2 user data. For more information, see Storing Container Instance Configuration in Amazon S3 (p. 74).

If you are manually starting the Amazon ECS container agent (for non-Amazon ECS-optimized AMIs), you can use these environment variables in the **docker run** command that you use to start the agent with the syntax `--env=VARIABLE_NAME=VARIABLE_VALUE`. For sensitive information, such as authentication credentials for private repositories, you should store your agent environment variables in a file and pass them all at one time with the `--env-file path_to_env_file` option.

Topics

- Available Parameters (p. 70)
- Storing Container Instance Configuration in Amazon S3 (p. 74)

Available Parameters

Environment Key	Example Values	Description	Default Value
ECS_CLUSTER	MyCluster	The cluster that this agent should check into.	default
ECS_RESERVED_PORTS	[22, 80, 5000, 8080]	An array of ports that should be marked as unavailable for scheduling on this container instance.	[22, 2375, 2376, 51678]
ECS_RESERVED_PORTS_UDP	[53, 123]	An array of UDP ports that should be marked as unavailable for scheduling on this container instance.	[]
ECS_ENGINE_AUTH_TYPE	dockercfg | docker	Required for private registry authentication. This is the type of authentication data in ECS_ENGINE_AUTH_DATA. For more information, see Authentication Formats (p. 76).	Null
ECS_ENGINE_AUTH_DATA	Example (ECS_ENGINE_AUTH_TYPE=dockercfg): {"https://index.docker.io/v1/": {"auth":"*zq212MzEXAMPLE7o6T25Dk0i*", "email":"*email@example.com*"}} Example (ECS_ENGINE_AUTH_TYPE=docker): {"https://index.docker.io/v1/": {"username":"*my_name*", "password":"*my_password*", "email":"*email@example.com*"}}	Required for private registry authentication. If ECS_ENGINE_AUTH_TYPE=dockercfg, then the ECS_ENGINE_AUTH_DATA value should be the contents of a Docker configuration file (~/.dockercfg or ~/.docker/config.json) created by running **docker login**. If ECS_ENGINE_AUTH_TYPE=docker, then the ECS_ENGINE_AUTH_DATA value should be a JSON representation of the registry server to authenticate against, as well as the authentication parameters required by that registry (such as user name, password, and email address for that account).	Null
AWS_DEFAULT_REGION	us-east-1	The region to be used in API requests as well as to infer the correct back-end host.	Taken from EC2 instance metadata.
AWS_ACCESS_KEY_ID	AKIAIOSFODNN7EXAMPLE	The access key used by the agent for all calls.	Taken from EC2 instance metadata.
AWS_SECRET_ACCESS_KEY	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY	The secret key used by the agent for all calls.	Taken from EC2

Environment Key	Example Values	Description	Default Value
			instance metadata.
DOCKER_HOST	unix:///var/run/docker.sock	Used to create a connection to the Docker daemon; behaves similarly to the environment variable as used by the Docker client.	unix:/// var/run/ docker.sock
ECS_LOGLEVEL	crit | error | warn | info | debug	The level to log at on stdout.	info
ECS_LOGFILE	/ecs-agent.log	The path to output full debugging information to. If blank, no logs are recorded. If this value is set, logs at the debug level (regardless of ECS_LOGLEVEL) are written to that file.	Null
ECS_CHECKPOINT	true | false	Whether or not to save the checkpoint state to the location specified with ECS_DATADIR.	If ECS_DATADIR is explicitly set to a non-empty value, then ECS_CHECKPOINT is set to true; otherwise, it is set to false.
ECS_DATADIR	/data	The name of the persistent data directory on the container that is running the Amazon ECS container agent. The directory is used to save information about the cluster and the agent state.	Null
ECS_UPDATES_ENABLED	true | false	Whether to exit for ECS agent updates when they are requested.	false
ECS_UPDATE_DOWNLOAD_DIR	/cache	The filesystem location to place update tarballs within the container when they are downloaded.	
ECS_DISABLE_METRICS	true | false	Whether to disable CloudWatch metrics for Amazon ECS. If this value is set to true, CloudWatch metrics are not collected.	false
ECS_DOCKER_GRAPHPATH	/var/lib/docker	Used to create the path to the state file of launched containers. The state file is used to read utilization metrics of containers.	/var/lib/ docker

Environment Variable Key	Example Values	Description	Default Value
AWS_SESSION_TOKEN		The session token used for temporary credentials.	Taken from EC2 instance metadata.
ECS_RESERVED_MEMORY	32	The amount of memory, in MiB, to reserve for processes that are not managed by ECS.	0
ECS_AVAILABLE_LOGGING_DRIVERS	["json-file","syslog"] For information about how to use the `awslogs` log driver, see Using the awslogs Log Driver (p. 107). For more information about the different log drivers available for your Docker version and how to configure them, see Configure logging drivers in the Docker documentation.	The logging drivers available on the container instance. The Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable before containers placed on that instance can use log configuration options for those drivers in tasks.	`["json-file"]`
ECS_DISABLE_PRIVILEGED	true or false	Whether launching privileged containers is disabled on the container instance. If this value is set to `true`, privileged containers are not permitted.	`false`
ECS_SELINUX_CAPABLE	true or false	Whether SELinux is available on the container instance.	`false`
ECS_APPARMOR_CAPABLE	true or false	Whether AppArmor is available on the container instance.	`false`
ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION	1h (Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", and "h".)	Time duration to wait from when a task is stopped until the docker container is removed. As this removes the docker container data, be aware that if this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. The minimum duration is `1m`; any value shorter than 1 minute is ignored.	3h
ECS_CONTAINER_STOP_TIMEOUT	30s (Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", and "h".)	Time duration to wait from when a task is stopped before its containers are forcefully killed if they do not exit normally on their own.	30s

Environment Key	Example Values	Description	Default Value
HTTP_PROXY	10.0.0.131:3128	The hostname (or IP address) and port number of an HTTP proxy to use for the ECS agent to connect to the Internet (for example, if your container instances do not have external network access through an Amazon VPC Internet gateway or NAT gateway or instance). If this variable is set, you must also set the `NO_PROXY` variable to filter EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 80).	Null
NO_PROXY	169.254.169.254,/var/run/docker.sock	The HTTP traffic that should not be forwarded to the specified `HTTP_PROXY`. You must specify `169.254.169.254,/var/run/docker.sock` to filter EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 80).	Null
ECS_ENABLE_TASK_IAM_ROLE	true | false	Whether IAM roles for tasks should be enabled on the container instance. For more information, see IAM Roles for Tasks (p. 186).	false
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST	true | false	Whether IAM roles for tasks should be enabled on the container instance when the agent is started with the `host` network mode.	false
ECS_DISABLE_IMAGE_CLEANUP	true	Whether to disable automated image cleanup for the Amazon ECS agent. For more information, see Automated Task and Image Cleanup (p. 75).	false
ECS_IMAGE_CLEANUP_INTERVAL	30m	The time interval between automated image cleanup cycles. If set to less than 10 minutes, the value is ignored.	30m
ECS_IMAGE_MINIMUM_CLEANUP_AGE	30m	The minimum time interval between when an image is pulled and when it can be considered for automated image cleanup.	1h

Environment Key	Example Values	Description	Default Value
ECS_NUM_IMAGES_DELETE_PER_CYCLE	5	The maximum number of images to delete in a single automated image cleanup cycle. If set to less than 1, the value is ignored.	5

Storing Container Instance Configuration in Amazon S3

Amazon ECS container agent configuration is controlled with the environment variables described above. The Amazon ECS-optimized AMI checks for these variables in `/etc/ecs/ecs.config` when the container agent starts and configures the agent accordingly. Certain innocuous environment variables, such as `ECS_CLUSTER`, can be passed to the container instance at launch time through Amazon EC2 user data and written to this file without consequence. However, other sensitive information, such as your AWS credentials or the `ECS_ENGINE_AUTH_DATA` variable, should never be passed to an instance in user data or written to `/etc/ecs/ecs.config` in a way that they would show up in a `.bash_history` file.

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch time. You can store a copy of your `ecs.config` file in a private bucket, and then use Amazon EC2 user data to install the AWS CLI and copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

To allow Amazon S3 read-only access for your container instance role

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose the IAM role you use for your container instances (this role is likely titled `ecsInstanceRole`). For more information, see Amazon ECS Container Instance IAM Role (p. 180).
4. Under **Managed Policies**, choose **Attach Policy**.
5. On the **Attach Policy** page, type `S3` into the **Filter** field to narrow the policy results.
6. Check the box to the left of the **AmazonS3ReadOnlyAccess** policy and click **Attach Policy**.

To store an `ecs.config` file in Amazon S3

1. Create an `ecs.config` file with valid environment variables and values from Amazon ECS Container Agent Configuration (p. 69) using the following format. This example configures private registry authentication. For more information, see Private Registry Authentication (p. 76).

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

2. Create a private bucket in Amazon S3 to store your configuration file. For more information, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.
3. Upload the `ecs.config` file to your Amazon S3 bucket. For more information, see Add an Object to a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.

To load an `ecs.config` file from Amazon S3 at launch

1. Complete the above procedures in this section to allow read-only Amazon S3 access to your container instances and store an `ecs.config` file in a private Amazon S3 bucket.

2. Launch new container instances by following the steps in Launching an Amazon ECS Container Instance (p. 39). In Step 10 (p. 41), use the following example script that installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`.

    ```
    #!/bin/bash
    yum install -y aws-cli
    aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
    ```

Automated Task and Image Cleanup

Each time a task is placed on a container instance, the Amazon ECS container agent checks to see if the images referenced in the task are the most recent of the specified tag in the repository; if not, it pulls the images from their respective repositories. If you frequently update the images in your tasks and services (for example, if you are using a continuous integration and continuous deployment (CI/CD) pipeline), your container instance storage can quickly fill up with Docker images that you are no longer using and will likely never use again.

Likewise, containers that belong to stopped tasks can also consume container instance storage with log information, data volumes, and other artifacts. These artifacts are useful for debugging containers that have stopped unexpectedly, but most of this storage can be safely freed up after a period of time.

By default, the Amazon ECS container agent automatically cleans up stopped tasks and Docker images that are not being used by any tasks on your container instances.

> **Note**
> The automated image cleanup feature requires at least version 1.13.0 of the Amazon ECS container agent. To update your agent to the latest version, see Updating the Amazon ECS Container Agent (p. 63).

Tunable Parameters

The following agent configuration variables are available to tune your automated task and image cleanup experience. For more information about how to set these variables on your container instances, see Amazon ECS Container Agent Configuration (p. 69).

ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION
> This variable specifies the time to wait before removing any containers that belong to stopped tasks. The image cleanup process cannot delete an image as long as there is a container that references it. After images are not referenced by any containers (either stopped or running), then the image becomes a candidate for cleanup. By default, this parameter is set to 3 hours but you can reduce this period to as low as 1 minute, if you need to for your application.

ECS_DISABLE_IMAGE_CLEANUP
> If you set this variable to `true`, then automated image cleanup is disabled on your container instance and no images are automatically removed.

ECS_IMAGE_CLEANUP_INTERVAL
> This variable specifies how frequently the automated image cleanup process should check for images to delete. The default is every 30 minutes but you can reduce this period to as low as 10 minutes to remove images more frequently.

ECS_IMAGE_MINIMUM_CLEANUP_AGE
> This variable specifies the minimum amount of time between when an image was pulled and when it may become a candidate for removal; this is used to prevent cleaning up images that have just been pulled. The default is 1 hour.

ECS_NUM_IMAGES_DELETE_PER_CYCLE
> This variable specifies how many images may be removed during a single cleanup cycle. The default is 5 and the minimum is 1.

Cleanup Workflow

When the Amazon ECS container agent is running and automated image cleanup is not disabled, the agent checks for Docker images that are not referenced by running or stopped containers at a frequency determined by the ECS_IMAGE_CLEANUP_INTERVAL variable. If unused images are found and they are older than the minimum cleanup time specified by the ECS_IMAGE_MINIMUM_CLEANUP_AGE variable, the agent removes up to the maximum number of images that are specified with the ECS_NUM_IMAGES_DELETE_PER_CYCLE variable, with the least-recently referenced images deleted first. After the images are removed, the agent waits until the next interval and repeats the process again.

Private Registry Authentication

The Amazon ECS container agent can authenticate with private registries, including Docker Hub, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions.

The agent looks for two environment variables when it launches: ECS_ENGINE_AUTH_TYPE, which specifies the type of authentication data that is being sent, and ECS_ENGINE_AUTH_DATA, which contains the actual authentication credentials.

The Amazon ECS-optimized AMI scans the /etc/ecs/ecs.config file for these variables when the container instance launches, and each time the service is started (with the **sudo start ecs** command). AMIs that are not Amazon ECS-optimized should store these environment variables in a file and pass them with the --env-file *path_to_env_file* option to the **docker run** command that starts the container agent.

> **Important**
> We do not recommend that you inject these authentication environment variables at instance launch time with Amazon EC2 user data or pass them with the --env option to the **docker run** command. These methods are not appropriate for sensitive data like authentication credentials. To safely add authentication credentials to your container instances, see Storing Container Instance Configuration in Amazon S3 (p. 74).

Authentication Formats

There are two available formats for private registry authentication, dockercfg and docker.

dockercfg Authentication Format

The dockercfg format uses the authentication information stored in the configuration file that is created when you run the **docker login** command. You can create this file by running **docker login** on your local system (or by logging into a container instance and running the command there) and entering your registry user name, password, and email address. Depending on your Docker version, this file is saved as either ~/.dockercfg or ~/.docker/config.json.

```
$ cat ~/.docker/config.json
```

```
{"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

In the above example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at run time. If you are not using the Amazon ECS-optimized AMI and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file` `path_to_env_file` option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

docker Authentication Format

The `docker` format uses a JSON representation of the registry server that the agent should authenticate with, as well as the authentication parameters required by that registry (such as user name, password, and the email address for that account). For a Docker Hub account, the JSON representation looks like this:

```
{
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  }
}
```

In this example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at run time. If you are not using the Amazon ECS-optimized AMI and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file` `path_to_env_file` option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

Enabling Private Registries

Use the following procedure to enable private registries for your container instances.

To enable private registries in the Amazon ECS-optimized AMI

1. Log into your container instance via SSH.

2. Open the `/etc/ecs/ecs.config` file and add the `ECS_ENGINE_AUTH_TYPE` and `ECS_ENGINE_AUTH_DATA` values for your registry and account.

   ```
   [ec2-user ~]$ sudo vi /etc/ecs/ecs.config
   ```

   This example authenticates a Docker Hub user account.

   ```
   ECS_ENGINE_AUTH_TYPE=docker
   ```

```
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

3. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state.

```
[ec2-user ~]$ docker inspect ecs-agent | grep ECS_DATADIR
            "ECS_DATADIR=/data",
```

> **Important**
> If the previous command does not return the `ECS_DATADIR` environment variable, you
> must stop any tasks running on this container instance before stopping the agent. Newer
> agents with the `ECS_DATADIR` environment variable save their state and you can stop
> and start them while tasks are running without issues. For more information, see Updating
> the Amazon ECS Container Agent (p. 63).

4. Stop the `ecs` service.

```
[ec2-user ~]$ sudo stop ecs
ecs stop/waiting
```

5. Restart the `ecs` service.

```
[ec2-user ~]$ sudo start ecs
ecs start/running, process 2959
```

6. (Optional) You can verify that the agent is running and see some information about your new
   container instance by querying the agent introspection API. For more information, see the section
   called "Amazon ECS Container Agent Introspection" (p. 78).

```
[ec2-user ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.13.1 (efe53c6)"
}
```

Amazon ECS Container Agent Introspection

The Amazon ECS container agent provides an API for gathering details about the container instance
that the agent is running on and the associated tasks that are running on that instance. You can use
the **curl** command from within the container instance to query the Amazon ECS container agent (port
51678) and return container instance metadata or task information.

To view container instance metadata, such as the container instance ID, the Amazon ECS cluster the
container instance is registered into, and the Amazon ECS container agent version info, log into your
container instance via SSH and run the following command:

```
[ec2-user ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.13.1 (efe53c6)"
}
```

To view information about all of the tasks that are running on a container instance, log into your
container instance via SSH and run the following command:

```
[ec2-user ~]$ curl http://localhost:51678/v1/tasks
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-east-1:<aws_account_id>:task/example5-58ff-46c9-
ae05-543f8example",
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Family": "hello_world",
      "Version": "8",
      "Containers": [
        {
          "DockerId":
 "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
          "Name": "mysql"
        },
        {
          "DockerId":
 "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
          "Name": "wordpress"
        }
      ]
    }
  ]
}
```

You can view information for a particular task that is running on a container instance by
specifying a task ARN (append ?taskarn=*task_arn* to the request) or the Docker ID (append ?
dockerid=*docker_id* to the request) for an individual container inside a task. To get task information
with a Docker ID, log into your container instance via SSH and run the following command:

> **Note**
> The Amazon ECS container agent introspection API requires full Docker IDs, not the short
> version that is shown with **docker ps**. You can get the full Docker ID for a container by
> running the **docker ps --no-trunc** command on the container instance.

```
[ec2-user ~]$ curl http://localhost:51678/v1/tasks?
dockerid=9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1
{
  "Arn": "arn:aws:ecs:us-east-1:<aws_account_id>:task/example5-58ff-46c9-
ae05-543f8example",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Family": "hello_world",
  "Version": "8",
  "Containers": [
    {
      "DockerId":
 "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
      "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
      "Name": "mysql"
    },
    {
```

```
      "DockerId":
 "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
      "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
      "Name": "wordpress"
    }
  ]
}
```

HTTP Proxy Configuration

If you need to configure your Amazon ECS container agent to use an HTTP proxy, you may do so by setting the following variables in the `/etc/ecs/ecs.config` file at launch time (with Amazon EC2 user data), or you can manually edit the configuration file and restart the agent afterwards:

`HTTP_PROXY=`*`10.0.0.131`*`:`*`3128`*
> Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the ECS agent to connect to the Internet (for example, if your container instances do not have external network access through an Amazon VPC Internet gateway or NAT gateway or instance).

`NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`
> Set this value to `169.254.169.254,169.254.170.2,/var/run/docker.sock` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

The above variable only affect the Amazon ECS container agent; they do not configure Docker or any other services (such as **yum**) to use the proxy.

Example HTTP proxy user data script

The example user data `cloud-boothook` script below configures the Amazon ECS container agent, the Docker daemon, and **yum** to use an HTTP proxy that you specify. You can also specify a cluster name that the container instance will register itself into.

To use this script when you launch a container instance, follow the steps in Launching an Amazon ECS Container Instance (p. 39), and in Step 10 (p. 41), copy and paste the `cloud-boothook` script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

```
#cloud-boothook
# Configure Yum, the Docker daemon, and the ECS agent to use an HTTP proxy

# Specify proxy host, port number, and ECS cluster name to use
PROXY_HOST=10.0.0.131
PROXY_PORT=3128
CLUSTER_NAME=proxy-test

# Set Yum HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_yum_http_proxy ]; then
 echo "proxy=http://$PROXY_HOST:$PROXY_PORT" >> /etc/yum.conf
 echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_yum_http_proxy
fi

# Set Docker HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ]; then
 echo "export HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/
docker
 echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi

# Set ECS agent HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_ecs-agent_http_proxy ]; then
 echo "ECS_CLUSTER=$CLUSTER_NAME" >> /etc/ecs/ecs.config
 echo "HTTP_PROXY=$PROXY_HOST:$PROXY_PORT" >> /etc/ecs/ecs.config
 echo "NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock" >> /etc/
ecs/ecs.config
 echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_ecs-agent_http_proxy
fi
```

Amazon ECS Task Definitions

A task definition is required to run Docker containers in Amazon ECS. Some of the parameters you can specify in a task definition include:

- Which Docker images to use with the containers in your task
- How much CPU and memory to use with each container
- Whether containers are linked together in a task
- The Docker networking mode to use for the containers in your task
- What (if any) ports from the container are mapped to the host container instance
- Whether the task should continue to run if the container finishes or fails
- The command the container should run when it is started
- What (if any) environment variables should be passed to the container when it starts
- Any data volumes that should be used with the containers in the task
- What (if any) IAM role your tasks should use for permissions

You can define multiple containers and data volumes in a task definition. For a complete description of the parameters available in a task definition, see Task Definition Parameters (p. 88).

Your entire application stack does not need to exist on a single task definition, and in most cases it should not. Your application can span multiple task definitions by combining related containers into their own task definitions, each representing a single component. For more information, see Application Architecture (p. 83).

Topics

Application Architecture

When you're considering how to model task definitions and services, it helps to think about what processes need to run together on the same instance and how you will scale each component. As an example, imagine an application that consists of the following components:

- A front-end service that displays information on a web page
- A back-end service that provides APIs for the front-end service
- A data store

In your development environment, you probably run all three containers together on your Docker host. You might be tempted to use the same approach for your production environment, but this approach has several drawbacks:

- Changes to one component can impact all three components, which may be a larger scope for the change than you want
- Each component is more difficult to scale because you have to scale every container proportionally
- Task definitions can only have 10 container definitions and your application stack might require more, either now or in the future
- Every container in a task definition must land on the same container instance, which may limit your instance choices to the largest sizes

Instead, you should create task definitions that group the containers that are used for a common purpose, and separate the different components into multiple task definitions. In this example, three task definitions each specify one container. The example cluster below has three container instances registered with three front-end service containers, two back-end service containers, and one data store service container.
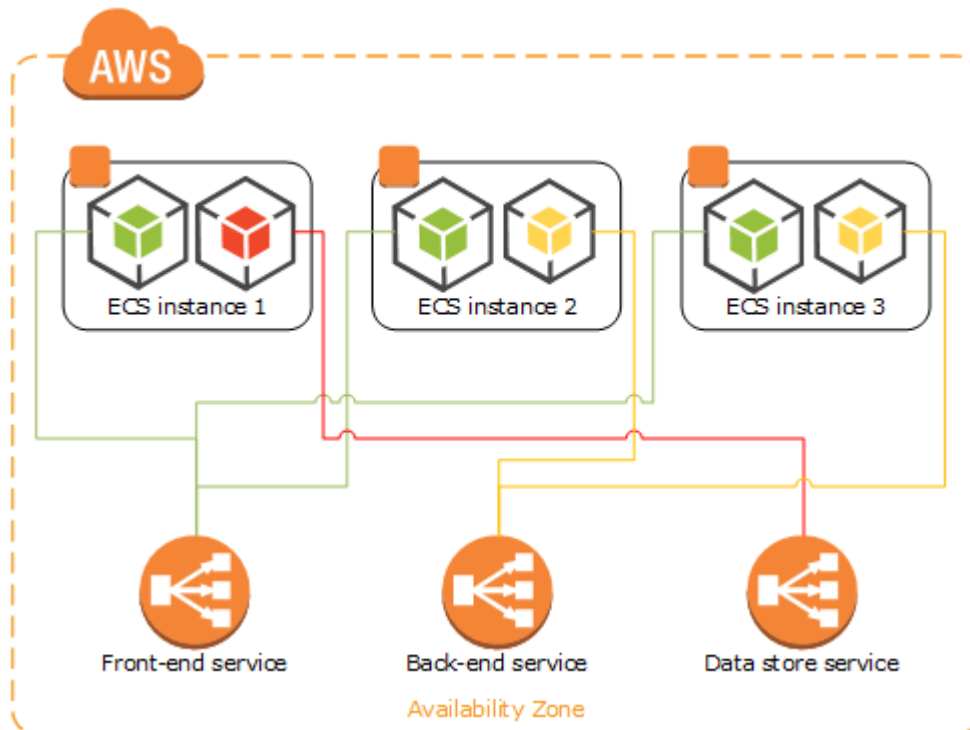
You can group related containers in a task definition, such as linked containers that must be run together. For example, you could add a log streaming container to your front-end service and include that in the same task definition.

After you have your task definitions, you can create services from them to maintain the availability of your desired tasks. For more information, see Creating a Service (p. 142). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see Service Load Balancing (p. 124). When your application requirements change, you can update your services to scale the number of desired tasks up or down, or to deploy newer versions of the containers in your tasks. For more information, see Updating a Service (p. 147).

Creating a Task Definition

Before you can run Docker containers on Amazon ECS, you need to create a task definition.

To create a new task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, choose the region to register your task definition in.
3. In the navigation pane, choose **Task Definitions**.
4. On the **Task Definitions** page, select **Create new Task Definition**.
5. (Optional) If you have a JSON representation of your task definition that you would like to use, complete the following steps:

   a. On the **Create a Task Definition** page, scroll to the bottom of the page and choose **Configure via JSON**.

   b. Paste your task definition JSON into the text area and choose **Save**.

   c. Verify your information and select **Create**.

6. In the **Task Definition Name** field, enter a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
7. (Optional) In the **Task Role** field, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 186).

   > **Note**
   > Only roles that have the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For help creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 188).

8. (Optional) In the **Network Mode** field, choose the Docker network mode that you would like to use for the containers in your task. The available network modes correspond to those described in Network settings in the Docker run reference.

   The default Docker network mode is `bridge`. If the network mode is set to `none`, you cannot specify port mappings in your container definitions, and the task's containers do not have external connectivity. The `host` network mode offers the highest networking performance for containers because they use the host network stack instead of the virtualized network stack provided by the `bridge` mode; however, exposed container ports are mapped directly to the corresponding host port, so you cannot take advantage of dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

9. For each container in your task definition, complete the following steps.

   a. Choose **Add Container Definition**.

   b. Fill out each required field and any optional fields to use in your container definitions (more container definition parameters are available in the **Advanced container configuration** menu). For more information, see Task Definition Parameters (p. 88).

c.   Select **Add** to add your container to the task definition.

10. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see
Using Data Volumes in Tasks (p. 102).

a.   In the **Name** field, enter a name for your volume. Up to 255 letters (uppercase and
lowercase), numbers, hyphens, and underscores are allowed.

b.   (Optional) In the **Source Path** field, enter the path on the host container instance to present to
the container. If this you leave this field empty, then the Docker daemon assigns a host path
for you. If you specify a source path, then the data volume persists at the specified location
on the host container instance until you delete it manually. If the source path does not exist
on the host container instance, the Docker daemon creates it. If the location does exist, the
contents of the source path folder are exported to the container.

11. Choose **Create** to finish.

Task Definition Template

An empty task definition template is shown below. You can use this template to create your task
definition which can then be pasted into the console JSON input area or saved to a file and used with
the AWS CLI `--cli-input-json` option. For more information about these parameters, see Task
Definition Parameters (p. 88).

```
{
    "family": "",
    "taskRoleArn": "",
    "networkMode": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [
                ""
            ],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": ""
                }
            ],
            "essential": true,
            "entryPoint": [
                ""
            ],
            "command": [
                ""
            ],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "mountPoints": [
```

```
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
            "hostname": "",
            "user": "",
            "workingDirectory": "",
            "disableNetworking": true,
            "privileged": true,
            "readonlyRootFilesystem": true,
            "dnsServers": [
                ""
            ],
            "dnsSearchDomains": [
                ""
            ],
            "extraHosts": [
                {
                    "hostname": "",
                    "ipAddress": ""
                }
            ],
            "dockerSecurityOptions": [
                ""
            ],
            "dockerLabels": {
                "KeyName": ""
            },
            "ulimits": [
                {
                    "name": "",
                    "softLimit": 0,
                    "hardLimit": 0
                }
            ],
            "logConfiguration": {
                "logDriver": "",
                "options": {
                    "KeyName": ""
                }
            }
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {
                "sourcePath": ""
            }
        }
    ]
```

```
}
```

Note

You can generate the above task definition template with the following AWS CLI command.

```
$ aws ecs register-task-definition --generate-cli-skeleton
```

Using a Docker Image in a Task Definition

This procedure demonstrates how to define a task definition that uses the `amazon-ecs-sample` image created in the article Docker Basics (p. 13).

To register a task definition with the `amazon-ecs-sample` image

1. Examine the `simple-app-task-def.json` file in the `ecs-demo-php-simple-app` folder.

```
{
    "family": "console-sample-app",
    "volumes": [
        {
            "name": "my-vol",
            "host": {}
        }
    ],
    "containerDefinitions": [
        {
            "environment": [],
            "name": "simple-app",
            "image": "amazon/amazon-ecs-sample",
            "cpu": 10,
            "memory": 500,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "my-vol",
                    "containerPath": "/var/www/my-vol"
                }
            ],
            "entryPoint": [
                "/usr/sbin/apache2",
                "-D",
                "FOREGROUND"
            ],
            "essential": true
        },
        {
            "name": "busybox",
            "image": "busybox",
            "cpu": 10,
            "memory": 500,
```

```
            "volumesFrom": [
            {
                "sourceContainer": "simple-app"
            }
            ],
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "/bin/sh -c \"while true; do /bin/date > /var/www/my-vol/
date; sleep 1; done\""
            ],
            "essential": false
        }
    ]
}
```

This task definition JSON file specifies two containers, one of which uses the `amazon-ecs-sample` image. By default, this image is pulled from the Amazon Docker Hub repository, but you can change the *amazon* repository defined above to your own repository if you want to use the *my-dockerhub-username*/amazon-ecs-sample image you pushed earlier.

2. Register a task definition with the `simple-app-task-def.json` file.

```
[ec2-user ecs-demo-php-simple-app]$ aws ecs register-task-definition --
cli-input-json file://simple-app-task-def.json
```

The task definition is registered in the `console-sample-app` family as defined in the JSON file.

To run a task with the `console-sample-app` task definition

Important
Before you can run tasks in Amazon ECS, you need to launch container instances into your cluster. For more information about how to set up and launch container instances, see Setting Up with Amazon ECS (p. 8) and Getting Started with Amazon ECS (p. 20).

*   Use the following AWS CLI command to run a task with the `console-sample-app` task definition.

```
[ec2-user ecs-demo-php-simple-app]$ aws ecs run-task --task-definition
 console-sample-app
```

Task Definition Parameters

Task definitions are split into four basic parts: the task family, the IAM task role, container definitions, and volumes. The family is the name of the task, and each family can have multiple revisions. The IAM task role specifies the permissions that containers in the task should have. Container definitions specify which image to use, how much CPU and memory the container are allocated, and many more options. Volumes allow you to share data between containers and even persist the data on the container instance when the containers are no longer running. The family and container definitions are required in a task definition, while task role, network mode, and volumes are optional.

Topics
*   Family (p. 89)

Family

`family`
> Type: string

> Required: yes

> When you register a task definition, you give it a family, which is similar to a name for multiple versions of the task definition, specified with a revision number. The first task definition that is registered into a particular family is given a revision of 1, and any task definitions registered after that are given a later sequential revision number.

Task Role

`taskRoleArn`
> Type: string

> Required: no

> When you register a task definition, you can provide a task role for an IAM role that allows the containers in the task permission to call the AWS APIs that are specified in its associated policies on your behalf. For more information, see IAM Roles for Tasks (p. 186).

Network Mode

`networkMode`
> Type: string

> Required: no

> When you register a task definition, you can specify the Docker networking mode to use with its containers. The default Docker network mode is `bridge`. If the network mode is set to `none`, you cannot specify port mappings in your container definitions, and the task's containers do not have external connectivity. The `host` network mode offers the highest networking performance for containers because they use the host network stack instead of the virtualized network stack provided by the `bridge` mode; however, exposed container ports are mapped directly to the corresponding host port, so you cannot take advantage of dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

Container Definitions

When you register a task definition, you must specify a list of container definitions that are passed to the Docker daemon on a container instance. The following parameters are allowed in a container definition.

Topics

Standard Container Definition Parameters

The following task definition parameters are either required or used in most container definitions.

name
> Type: string
>
> Required: yes
>
> The name of a container. If you are linking multiple containers together in a task definition, the `name` of one container can be entered in the `links` of another container to connect the containers. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This parameter maps to `name` in the Create a container section of the Docker Remote API and the `--name` option to **docker run**.

image
> Type: string
>
> Required: yes
>
> The image used to start a container. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. You can also specify other repositories with `repository-url/image:tag`. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to `Image` in the Create a container section of the Docker Remote API and the `IMAGE` parameter of **docker run**.
>
> - Images in Amazon ECR repositories use the full `registry/repository:tag` naming convention. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`
> - 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`).

memory
> Type: integer
>
> Required: no
>
> The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to `Memory` in the Create a container section of the Docker Remote API and the `--memory` option to **docker run**.
>
> You must specify a non-zero integer for one or both of `memory` or `memoryReservation` in container definitions. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance on which the container is placed; otherwise, the value of `memory` is used.
>
> The Docker daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

memoryReservation
> Type: integer
>
> Required: no
>
> The soft limit (in MiB) of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory to this soft limit; however, your

container can consume more memory when it needs to, up to either the hard limit specified with the `memory` parameter (if applicable), or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the Create a container section of the Docker Remote API and the `--memory` option to **docker run**.

You must specify a non-zero integer for one or both of `memory` or `memoryReservation` in container definitions. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance on which the container is placed; otherwise, the value of `memory` is used.

For example, if your container normally uses 128 MiB of memory, but occasionally bursts to 256 MiB of memory for short periods of time, you can set a `memoryReservation` of 128 MiB, and a `memory` hard limit of 300 MiB. This configuration would allow the container to only reserve 128 MiB of memory from the remaining resources on the container instance, but also allow the container to consume more memory resources when needed.

portMappings
    Type: object array

    Required: no

    Port mappings allow containers to access ports on the host container instance to send or receive traffic. This parameter maps to `PortBindings` in the Create a container section of the Docker Remote API and the `--publish` option to **docker run**. If the network mode of a task definition is set to `none`, then you cannot specify port mappings. If the network mode of a task definition is set to `host`, then host ports must either be undefined or they must match the container port in the port mapping.

> **Note**
> After a task reaches the `RUNNING` status, manual and automatic host and container port assignments are visible in the **Network Bindings** section of a container description of a selected task in the Amazon ECS console, or the `networkBindings` section of **describe-tasks** AWS CLI command output or `DescribeTasks` API responses.

hostPort
    Type: integer

    Required: no

    The port number on the container instance to reserve for your container. You can specify a non-reserved host port for your container port mapping (this is referred to as *static* host port mapping), or you can omit the `hostPort` (or set it to `0`) while specifying a `containerPort` and your container will automatically receive a port (this is referred to as *dynamic* host port mapping) in the ephemeral port range for your container instance operating system and Docker version.

    The default ephemeral port range is 49153 to 65535, and this range is used for Docker versions prior to 1.6.0. For Docker version 1.6.0 and later, the Docker daemon tries to read the ephemeral port range from `/proc/sys/net/ipv4/ip_local_port_range` (which is 32768 to 61000 on the latest Amazon ECS-optimized AMI); if this kernel parameter is unavailable, the default ephemeral port range is used. You should not attempt to specify a host port in the ephemeral port range, since these are reserved for automatic assignment. In general, ports below 32768 are outside of the ephemeral port range.

    The default reserved ports are 22 for SSH, the Docker ports 2375 and 2376, and the Amazon ECS container agent port 51678. Any host port that was previously user-specified for a running task is also reserved while the task is running (after a task stops, the host port is released). The current reserved ports are displayed in the `remainingResources` of **describe-container-instances** output, and a container instance may have up to 100 reserved ports at a time, including the default reserved ports (automatically assigned ports do not count toward the 100 reserved ports limit).

containerPort
　　Type: integer

　　Required: yes, when `portMappings` are used

　　The port number on the container that is bound to the user-specified or automatically assigned
　　host port. If you specify a container port and not a host port, your container automatically
　　receives a host port in the ephemeral port range (for more information, see `hostPort`). Port
　　mappings that are automatically assigned in this way do not count toward the 100 reserved
　　ports limit of a container instance.

protocol
　　Type: string

　　Required: no

　　The protocol used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

　　Important
　　UDP support is only available on container instances that were launched with version
　　1.2.0 of the Amazon ECS container agent (such as the `amzn-ami-2015.03.c-`
　　`amazon-ecs-optimized` AMI) or later, or with container agents that have been
　　updated to version 1.3.0 or later. To update your container agent to the latest version,
　　see Updating the Amazon ECS Container Agent (p. 63).

If you are specifying a host port, use the following syntax:

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

If you want an automatically assigned host port, use the following syntax:

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

Advanced Container Definition Parameters

The following advanced container definition parameters provide extended capabilities to the **docker
run** command that is used to launch containers on your Amazon ECS container instances.

Topics
- Environment (p. 93)
- Network Settings (p. 95)
- Storage and Logging (p. 96)
- Security (p. 99)
- Resource Limits (p. 100)
- Docker Labels (p. 101)

Environment

cpu

Type: integer

Required: no

The number of `cpu` units to reserve for the container. A container instance has 1,024 `cpu` units for every CPU core. This parameter specifies the minimum amount of CPU to reserve for a container, and containers share unallocated CPU units with other containers on the instance with the same ratio as their allocated amount. This parameter maps to `CpuShares` in the Create a container section of the Docker Remote API and the `--cpu-shares` option to **docker run**.

> **Note**
> You can determine the number of CPU units that are available per Amazon EC2 instance type by multiplying the vCPUs listed for that instance type on the Amazon EC2 Instances detail page by 1,024.

For example, if you run a single-container task on a single-core instance type with 512 CPU units specified for that container, and that is the only task running on the container instance, that container could use the full 1,024 CPU unit share at any given time. However, if you launched another copy of the same task on that container instance, each task would be guaranteed a minimum of 512 CPU units when needed, and each container could float to higher CPU usage if the other container was not using it, but if both tasks were 100% active all of the time, they would be limited to 512 CPU units.

The Docker daemon on the container instance uses the CPU value to calculate the relative CPU share ratios for running containers. For more information, see CPU share constraint in the Docker documentation. The minimum valid CPU share value that the Linux kernel will allow is 2; however, the CPU parameter is not required, and you can use CPU values below 2 in your container definitions. For CPU values below 2 (including null), the behavior varies based on your Amazon ECS container agent version:

- **Agent versions <= 1.1.0:** Null and zero CPU values are passed to Docker as 0, which Docker then converts to 1,024 CPU shares. CPU values of 1 are passed to Docker as 1, which the Linux kernel converts to 2 CPU shares.
- **Agent versions >= 1.2.0:** Null, zero, and CPU values of 1 are passed to Docker as 2.

essential

Type: Boolean

Required: no

If the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason, all other containers that are part of the task are stopped. If the `essential` parameter of a container is marked as `false`, then its failure does not affect the rest of the containers in a task. If this parameter is omitted, a container is assumed to be essential.

All tasks must have at least one essential container. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see Application Architecture (p. 83).

```
"essential": true|false
```

entryPoint

> **Important**
> Early versions of the Amazon ECS container agent do not properly handle `entryPoint` parameters. If you have problems using `entryPoint`, update your container agent or enter your commands and arguments as `command` array items instead.

Type: string array

Required: no

The entry point that is passed to the container. This parameter maps to `Entrypoint` in the Create a container section of the Docker Remote API and the `--entrypoint` option to **docker run**. For more information about the Docker `ENTRYPOINT` parameter, go to https://docs.docker.com/engine/reference/builder/#entrypoint.

```
"entryPoint": ["string", ...]
```

command
    Type: string array

    Required: no

    The command that is passed to the container. This parameter maps to `Cmd` in the Create a container section of the Docker Remote API and the `COMMAND` parameter to **docker run**. For more information about the Docker `CMD` parameter, go to https://docs.docker.com/engine/reference/builder/#cmd.

```
"command": ["string", ...]
```

workingDirectory
    Type: string

    Required: no

    The working directory in which to run commands inside the container. This parameter maps to `WorkingDir` in the Create a container section of the Docker Remote API and the `--workdir` option to **docker run**.

```
"workingDirectory": "string"
```

environment
    Type: object array

    Required: no

    The environment variables to pass to a container. This parameter maps to `Env` in the Create a container section of the Docker Remote API and the `--env` option to **docker run**.

    Important
    We do not recommend using plain text environment variables for sensitive information, such as credential data.

    name
        Type: string

        Required: yes, when `environment` is used

        The name of the environment variable.

    value
        Type: string

        Required: yes, when `environment` is used

        The value of the environment variable.

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

Network Settings

disableNetworking

 Type: Boolean

Required: no

When this parameter is true, networking is disabled within the container. This parameter maps to `NetworkDisabled` in the Create a container section of the Docker Remote API.

```
"disableNetworking": true|false
```

links

 Type: string array

Required: no

The `link` parameter allows containers to communicate with each other without the need for port mappings. The `name:internalName` construct is analogous to `name:alias` in Docker links. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. For more information about linking Docker containers, go to https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/. This parameter maps to `Links` in the Create a container section of the Docker Remote API and the `--link` option to **docker run**.

> **Important**
> Containers that are collocated on a single container instance may be able to communicate with each other without requiring links or host port mappings. Network isolation is achieved on the container instance using security groups and VPC settings.

```
"links": ["name:internalName", ...]
```

hostname

 Type: string

Required: no

The hostname to use for your container. This parameter maps to `Hostname` in the Create a container section of the Docker Remote API and the `--hostname` option to **docker run**.

```
"hostname": "string"
```

dnsServers

 Type: Boolean

Required: no

A list of DNS servers that are presented to the container. This parameter maps to `Dns` in the Create a container section of the Docker Remote API and the `--dns` option to **docker run**.

Type: string array

Required: no

```
"dnsServers": ["string", ...]
```

dnsSearchDomains
Type: string array

Required: no

A list of DNS search domains that are presented to the container. This parameter maps to
DnsSearch in the Create a container section of the Docker Remote API and the --dns-search
option to **docker run**.

```
"dnsSearchDomains": ["string", ...]
```

extraHosts
Type: object array

Required: no

A list of hostnames and IP address mappings to append to the /etc/hosts file on the container.
This parameter maps to ExtraHosts in the Create a container section of the Docker Remote API
and the --add-host option to **docker run**.

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
   ]
```

hostname
Type: string

Required: yes, when extraHosts are used

The hostname to use in the /etc/hosts entry.

ipAddress
Type: string

Required: yes, when extraHosts are used

The IP address to use in the /etc/hosts entry.

Storage and Logging

readonlyRootFilesystem
Type: Boolean

Required: no

When this parameter is true, the container is given read-only access to its root file system. This
parameter maps to ReadonlyRootfs in the Create a container section of the Docker Remote API
and the --read-only option to **docker run**.

```
"readonlyRootFilesystem": true|false
```

mountPoints
    Type: object array

    Required: no

    The mount points for data volumes in your container. This parameter maps to `Volumes` in the
    Create a container section of the Docker Remote API and the `--volume` option to **docker run**.

    sourceVolume
        Type: string

        Required: yes, when `mountPoints` are used

        The name of the volume to mount.

    containerPath
        Type: string

        Required: yes, when `mountPoints` are used

        The path on the container to mount the host volume at.

    readOnly
        Type: boolean

        Required: no

        If this value is `true`, the container has read-only access to the volume. If this value is `false`,
        then the container can write to the volume. The default value is `false`.

```
"mountPoints": [
            {
                "sourceVolume": "string",
                "containerPath": "string",
                "readOnly": true|false
            }
        ]
```

volumesFrom
    Type: object array

    Required: no

    Data volumes to mount from another container. This parameter maps to `VolumesFrom` in the
    Create a container section of the Docker Remote API and the `--volumes-from` option to **docker
    run**.

    sourceContainer
        Type: string

        Required: yes, when `volumesFrom` is used

        The name of the container to mount volumes from.

    readOnly
        Type: Boolean

        Required: no

        If this value is `true`, the container has read-only access to the volume. If this value is `false`,
        then the container can write to the volume. The default value is `false`.

```
"volumesFrom": [
            {
                "sourceContainer": "string",
```

```
                            "readOnly": true|false
                     }
                 ]
```

logConfiguration

    Type: LogConfiguration object

    Required: no

    The log configuration specification for the container. This parameter maps to `LogConfig` in the Create a container section of the Docker Remote API and the `--log-driver` option to `docker run`. By default, containers use the same logging driver that the Docker daemon uses; however the container may use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the container definition. To use a different logging driver for a container, the log system must be configured properly on the container instance (or on a different log server for remote logging options). For more information on the options for different supported log drivers, see Configure logging drivers in the Docker documentation.

    For more information on using the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see Using the awslogs Log Driver (p. 107).

> **Note**
>
> Amazon ECS currently supports a subset of the logging drivers available to the Docker daemon (shown in the valid values below). Currently unsupported log drivers may be available in future releases of the Amazon ECS container agent.

    This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance, run the following command, and look for the server API version in the output:

```
$ sudo docker version
```

> **Note**
>
> The Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable before containers placed on that instance can use these log configuration options. For more information, see Amazon ECS Container Agent Configuration (p. 69).

```
"logConfiguration": {
      "logDriver": "json-
file"|"syslog"|"journald"|"gelf"|"fluentd"|"awslogs",
      "options": {"string": "string"
         ...}
```

logDriver

    Type: string

    Valid values: `"json-file"` | `"syslog"` | `"journald"` | `"gelf"` | `"fluentd"` | `"awslogs"`

    Required: yes, when `logConfiguration` is used

    The log driver to use for the container. The valid values listed above are log drivers that the Amazon ECS container agent can communicate with by default.

> **Note**
>
> If you have a custom driver that is not listed above that you would like to work with the Amazon ECS container agent, you can fork the Amazon ECS container agent

project that is available on GitHub and customize it to work with that driver. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently provide support for running modified copies of this software.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance, run the following command, and look for the server API version in the output:

```
$ sudo docker version
```

options
    Type: string to string map

    Required: no

    The configuration options to send to the log driver. This parameter requires version 1.19 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance, run the following command, and look for the server API version in the output:

```
$ sudo docker version
```

Security

privileged
    Type: Boolean

    Required: no

    When this parameter is true, the container is given elevated privileges on the host container instance (similar to the root user). This parameter maps to Privileged in the Create a container section of the Docker Remote API and the --privileged option to **docker run**.

```
"privileged": true|false
```

user
    Type: string

    Required: no

    The user name to use inside the container. This parameter maps to User in the Create a container section of the Docker Remote API and the --user option to **docker run**.

```
"user": "string"
```

dockerSecurityOptions
    Type: string array

    Required: no

    A list of strings to provide custom labels for SELinux and AppArmor multi-level security systems. This parameter maps to SecurityOpt in the Create a container section of the Docker Remote API and the --security-opt option to **docker run**.

```
"dockerSecurityOptions": ["string", ...]
```

> **Note**
> The Amazon ECS container agent running on a container instance must register with
> the `ECS_SELINUX_CAPABLE=true` or `ECS_APPARMOR_CAPABLE=true` environment
> variables before containers placed on that instance can use these security options. For
> more information, see Amazon ECS Container Agent Configuration (p. 69).

Resource Limits

`ulimits`
> Type: object array
>
> Required: no
>
> A list of `ulimits` to set in the container. This parameter maps to `Ulimits` in the Create a
> container section of the Docker Remote API and the `--ulimit` option to **docker run**. This
> parameter requires version 1.18 of the Docker Remote API or greater on your container instance.
> To check the Docker Remote API version on your container instance, log into your container
> instance, run the following command, and look for the server API version in the output:

```
$ sudo docker version
```

```
"ulimits": [
        {
          "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|
          "softLimit": integer,
          "hardLimit": integer
        }
        ...
     ]
```

> `name`
> > Type: string
> >
> > Valid values: `"core"` | `"cpu"` | `"data"` | `"fsize"` | `"locks"` | `"memlock"` |
> > `"msgqueue"` | `"nice"` | `"nofile"` | `"nproc"` | `"rss"` | `"rtprio"` | `"rttime"`
> > | `"sigpending"` | `"stack"`
> >
> > Required: yes, when `ulimits` are used
> >
> > The `type` of the `ulimit`.
>
> `hardLimit`
> > Type: integer
> >
> > Required: yes, when `ulimits` are used
> >
> > The hard limit for the `ulimit` type.
>
> `softLimit`
> > Type: integer
> >
> > Required: yes, when `ulimits` are used
> >
> > The soft limit for the `ulimit` type.

Docker Labels

`dockerLabels`

> Type: string to string map
>
> Required: no
>
> A key/value map of labels to add to the container. This parameter maps to `Labels` in the Create a container section of the Docker Remote API and the `--label` option to **docker run**. This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance, run the following command, and look for the server API version in the output:

```
$ sudo docker version
```

```
"dockerLabels": {"string": "string"
       ...}
```

Volumes

When you register a task definition, you can optionally specify a list of volumes that will be passed to the Docker daemon on a container instance and become available for other containers on the same container instance to access. For more information, see Using Data Volumes in Tasks (p. 102). The following parameters are allowed in a container definition:

`name`

> Type: string
>
> Required: yes
>
> The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This name is referenced in the `sourceVolume` parameter of container definition `mountPoints`.

`host`

> Type: object
>
> Required: no
>
> The contents of the `host` parameter determine whether your data volume persists on the host container instance and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.
>
> By default, Docker-managed volumes are created in `/var/lib/docker/vfs/dir/`. You can change this default location by writing `OPTIONS="-g=/my/path/for/docker/volumes"` to `/etc/sysconfig/docker` on the container instance.
>
> `sourcePath`
>
> > Type: string
> >
> > Required: no
> >
> > The path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you.
> >
> > If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host container instance until you delete it manually. If the

sourcePath value does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

```
[
  {
    "name": "string",
    "host": {
      "sourcePath": "string"
    }
  }
]
```

Using Data Volumes in Tasks

There are several use cases for using data volumes in Amazon ECS task definitions. Some common examples are to provide persistent data volumes for use with containers, to define an empty, nonpersistent data volume and mount it on multiple containers on the same container instance, and to share defined data volumes at different locations on different containers on the same container instance.

Note

For operating systems that use devicemapper (such as Amazon Linux and the Amazon ECS-optimized AMI), only file systems that are available when the Docker daemon is started will be available to Docker containers. You can use a cloud boothook to mount your file system before the Docker daemon starts, or you can restart the Docker daemon and the Amazon ECS container agent after the file system is mounted to make the file system available to your container volume mounts.

To provide persistent data volumes for containers

When a volume is defined with a sourcePath value, the data volume persists even after all containers that referenced it have stopped. Any files that exist in the at the sourcePath are presented to the containers at the containerPath value, and any files that are written to the containerPath value by running containers that mount the data volume are written to the sourcePath value on the container instance.

Important

Amazon ECS does not sync your data volumes across container instances. Tasks that use persistent data volumes can be placed on any container instance in your cluster that has available capacity. If your tasks require persistent data volumes after stopping and restarting, you should always specify the same container instance at task launch time with the AWS CLI start-task command.

1. In the task definition volumes section, define a data volume with name and sourcePath values.

```
"volumes": [
  {
    "name": "webdata",
    "host": {
      "sourcePath": "/ecs/webdata"
    }
  }
]
```

2. In the containerDefinitions section, define a container with mountPoints that reference the name of the defined volume and the containerPath value to mount the volume at on the container.

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "nginx",
    "cpu": 99,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webdata",
        "containerPath": "/usr/share/nginx/html"
      }
    ]
  }
]
```

To provide nonpersistent empty data volumes for containers

In some cases, you want containers to share the same empty data volume, but you aren't interested in keeping the data after the task has finished. For example, you may have two database containers that need to access the same scratch file storage location during a task.

1. In the task definition `volumes` section, define a data volume with the name `database_scratch`.

    Note
    Because the `database_scratch` volume does not specify a source path, the Docker daemon manages the volume for you. When no containers reference this volume, the Amazon ECS container agent task cleanup service eventually deletes it (by default, this happens 3 hours after the container exits, but you can configure this duration with the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent variable). For more information, see Amazon ECS Container Agent Configuration (p. 69). If you need this data to persist, specify a `sourcePath` value for the volume.

    ```
    "volumes": [
      {
        "name": "database_scratch",
        "host": {}
      }
    ]
    ```

2. In the `containerDefinitions` section, create the database container definitions so they mount the nonpersistent data volumes.

    ```
    "containerDefinitions": [
      {
        "name": "database1",
        "image": "my-repo/database",
        "cpu": 100,
        "memory": 100,
        "essential": true,
    ```

```
        "mountPoints": [
          {
            "sourceVolume": "database_scratch",
            "containerPath": "/var/scratch"
          }
        ]
      },
      {
        "name": "database2",
        "image": "my-repo/database",
        "cpu": 100,
        "memory": 100,
        "essential": true,
        "mountPoints": [
          {
            "sourceVolume": "database_scratch",
            "containerPath": "/var/scratch"
          }
        ]
      }
    ]
```

To mount a defined volume on multiple containers

You can define a data volume in a task definition and mount that volume at different locations on
different containers. For example, your host container has a website data folder at /data/webroot,
and you may want to mount that data volume as read-only on two different web servers that have
different document roots.

1. In the task definition volumes section, define a data volume with the name webroot and the
    source path /data/webroot.

```
    "volumes": [
      {
        "name": "webroot",
        "host": {
          "sourcePath": "/data/webroot"
        }
      }
    ]
```

2. In the containerDefinitions section, define a container for each web server with
    mountPoints values that associate the webroot volume with the containerPath value
    pointing to the document root for that container.

```
    "containerDefinitions": [
      {
        "name": "web-server-1",
        "image": "my-repo/ubuntu-apache",
        "cpu": 100,
        "memory": 100,
        "portMappings": [
          {
            "containerPort": 80,
            "hostPort": 80
          }
        ],
```

```
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "web-server-2",
      "image": "my-repo/sles11-apache",
      "cpu": 100,
      "memory": 100,
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 8080
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/srv/www/htdocs",
          "readOnly": true
        }
      ]
    }
  ]
```

To mount volumes from another container using `volumesFrom`

You can define one or more volumes on a container, and then use the `volumesFrom` parameter
in a different container definition (within the same task) to mount all of the volumes from the
`sourceContainer` at their originally defined mount points. The `volumesFrom` parameter applies to
volumes defined in the task definition, and those that are built into the image with a Dockerfile.

1. (Optional) To share a volume that is built into an image, you need to build the image with the
    volume declared in a VOLUME instruction. The following example Dockerfile uses an `httpd` image
    and then adds a volume and mounts it at `dockerfile_volume` in the Apache document root
    (which is the folder used by the `httpd` web server):

    ```
    FROM httpd
    VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]
    ```

    You can build an image with this Dockerfile and push it to a repository, such as Docker Hub, and
    use it in your task definition. The example `my-repo/httpd_dockerfile_volume` image used in
    the following steps was built with the above Dockerfile.

2. Create a task definition that defines your other volumes and mount points for the containers. In this
    example `volumes` section, you create an empty volume called `empty`, which the Docker daemon
    will manage. There is also a host volume defined called `host_etc`, which exports the `/etc` folder
    on the host container instance.

    ```
    {
      "family": "test-volumes-from",
    ```

```
    "volumes": [
      {
        "name": "empty",
        "host": {}
      },
      {
        "name": "host_etc",
        "host": {
          "sourcePath": "/etc"
        }
      }
    ],
```

In the container definitions section, create a container that mounts the volumes defined earlier. In this example, the web container (which uses the image built with a volume in the Dockerfile) mounts the empty and host_etc volumes.

```
    "containerDefinitions": [
      {
        "name": "web",
        "image": "my-repo/httpd_dockerfile_volume",
        "cpu": 100,
        "memory": 500,
        "portMappings": [
          {
            "containerPort": 80,
            "hostPort": 80
          }
        ],
        "mountPoints": [
          {
            "sourceVolume": "empty",
            "containerPath": "/usr/local/apache2/htdocs/empty_volume"
          },
          {
            "sourceVolume": "host_etc",
            "containerPath": "/usr/local/apache2/htdocs/host_etc"
          }
        ],
        "essential": true
      },
```

Create another container that uses volumesFrom to mount all of the volumes that are associated with the web container. All of the volumes on the web container will likewise be mounted on the busybox container (including the volume specified in the Dockerfile that was used to build the my-repo/httpd_dockerfile_volume image).

```
      {
        "name": "busybox",
        "image": "busybox",
        "volumesFrom": [
          {
            "sourceContainer": "web"
          }
        ],
        "cpu": 100,
        "memory": 500,
```

```
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date &&
 echo $(date) > /usr/local/apache2/htdocs/host_etc/date && echo $(date)
 > /usr/local/apache2/htdocs/dockerfile_volume/date"
      ],
      "essential": false
    }
  ]
}
```

When this task is run, the two containers mount the volumes, and the `command` in the `busybox` container writes the date and time to a file called `date` in each of the volume folders, which are then visible at the web site displayed by the `web` container.

> **Note**
> Because the `busybox` container runs a quick command and then exits, it needs to be set as `"essential": false` in the container definition to prevent it from stopping the entire task when it exits.

Using the awslogs Log Driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. This enables you to view different logs from your containers in one convenient location, and it prevents your container logs from taking up disk space on your container instances. This topic helps you get started using the `awslogs` log driver in your task definitions.

To send system logs from your Amazon ECS container instances to CloudWatch Logs, see Using CloudWatch Logs with Container Instances (p. 43). For more information about CloudWatch Logs, see Monitoring Log Files in the *Amazon CloudWatch User Guide*.

Topics

- Enabling the awslogs Log Driver on your Container Instances (p. 107)
- Creating Your Log Groups (p. 108)
- Available awslogs Log Driver Options (p. 108)
- Specifying a Log Configuration in your Task Definition (p. 109)
- Viewing awslogs Container Logs in CloudWatch Logs (p. 111)

Enabling the awslogs Log Driver on your Container Instances

Your Amazon ECS container instances require at least version 1.9.0 of the container agent to enable the `awslogs` log driver. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 63).

> **Note**
> If you are not using the Amazon ECS-optimized AMI (with at least version 1.9.0-1 of the `ecs-init` package) for your container instances, you also need to specify that the `awslogs` logging driver is available on the container instance when you start the agent by using the following environment variable in your **docker run** statement or environment variable file. For more information, see Installing the Amazon ECS Container Agent (p. 58).

```
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
```

Your Amazon ECS container instances also require `logs:CreateLogStream` and `logs:PutLogEvents` permission on the IAM role with which you launch your container instances. If you created your Amazon ECS container instance role before `awslogs` log driver support was enabled in Amazon ECS, then you might need to add this permission. If your container instances use the managed IAM policy for container instances, then your container instances should have the correct permissions. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see To check for the `ecsInstanceRole` in the IAM console (p. 181).

Creating Your Log Groups

The `awslogs` log driver can send log streams to existing log groups in CloudWatch Logs, but it cannot create log groups. Before you launch any tasks that use the `awslogs` log driver, you must create the log groups that you intend your containers to use.

As an example, you could have a task with a WordPress container (which uses the `awslogs-wordpress` log group) that is linked to a MySQL container (which uses the `awslogs-mysql` log group). The sections below show how to create these log groups with the AWS CLI and with the CloudWatch console.

Creating a Log Group with the AWS CLI

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information, see the AWS Command Line Interface User Guide.

If you have a working installation of the AWS CLI, you can use it to create your log groups. The command below creates a log group called `awslogs-wordpress` in the `ap-northeast-1` region. Run this command for each log group to create, replacing the log group name with your value and region name to the desired log destination.

```
$ aws logs create-log-group --log-group-name awslogs-wordpress --region ap-northeast-1
```

Creating a Log Group with the CloudWatch Console

The following procedure creates a log group in the CloudWatch console.

To create a log group in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**.
3. Choose **Actions**, **Create log group**.
4. For **Log Group Name**, enter the name of the log group to create.
5. Choose **Create log group** to finish.

Available awslogs Log Driver Options

The `awslogs` log driver supports the following options in Amazon ECS task definitions:

awslogs-region
    Required: Yes

Specify the region to which the `awslogs` log driver should send your Docker logs. You can choose to send all of your logs from clusters in different regions to a single region in CloudWatch Logs so that they are all visible in one location, or you can separate them by region for more granularity. Be sure that the specified log group exists in the region that you specify with this option.

`awslogs-group`

Required: Yes

You must specify a log group to which the `awslogs` log driver will send its log streams. For more information, see Creating Your Log Groups (p. 108).

`awslogs-stream-prefix`

Required: No

The `awslogs-stream-prefix` option allows you to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task to which the container belongs. If you specify a prefix with this option, then the log stream takes the following format:

```
prefix-name/container-name/ecs-task-id
```

If you do not specify a prefix with this option, then the log stream is named after the container ID that is assigned by the Docker daemon on the container instance. Because it is difficult to trace logs back to the container that sent them with just the Docker container ID (which is only available on the container instance), we recommend that you specify a prefix with this option.

For Amazon ECS services, you could use the service name as the prefix, which would allow you to trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task to which the container belongs.

Specifying a Log Configuration in your Task Definition

Before your containers can send logs to CloudWatch, you must specify the `awslogs` log driver for containers in your task definition. This section describes the log configuration for a container to use the `awslogs` log driver. For more information, see Creating a Task Definition (p. 84).

The task definition JSON shown below has a `logConfiguration` object specified for each container; one for the WordPress container that sends logs to a log group called `awslogs-wordpress`, and one for a MySQL container that sends logs to a log group called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```
{
    "containerDefinitions": [
        {
            "name": "wordpress",
            "links": [
                "mysql"
            ],
            "image": "wordpress",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
```

```
                "options": {
                    "awslogs-group": "awslogs-wordpress",
                    "awslogs-region": "ap-northeast-1",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
                }
            ],
            "name": "mysql",
            "image": "mysql",
            "cpu": 10,
            "memory": 500,
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "ap-northeast-1",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            }
        }
    ],
    "family": "awslogs-example"
}
```

In the Amazon ECS console, the log configuration for the `wordpress` container is specified as shown in the image below.
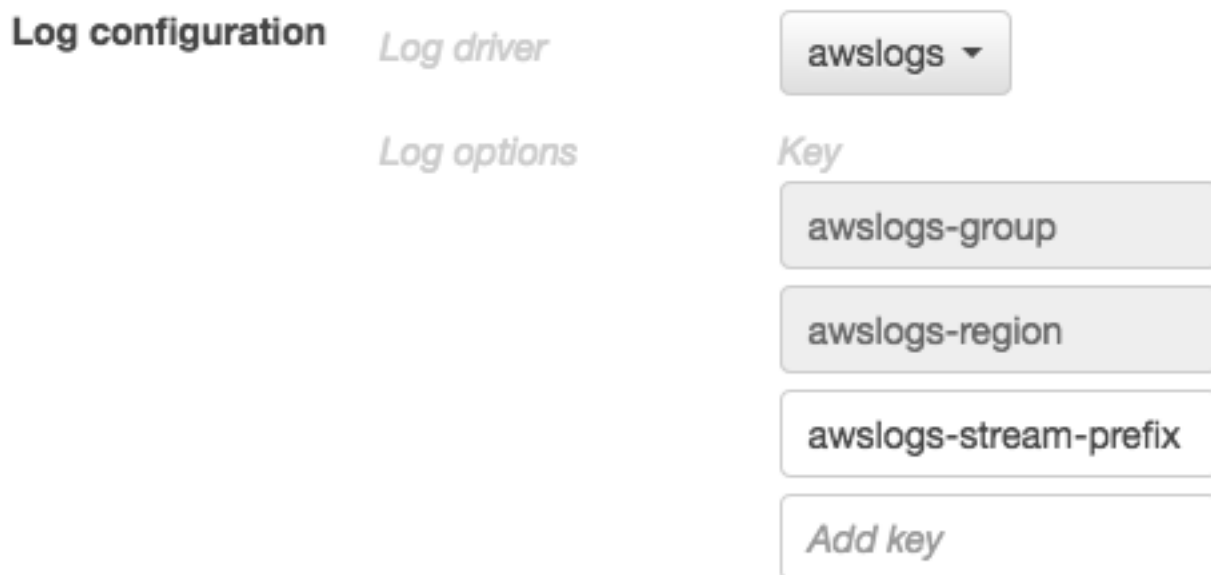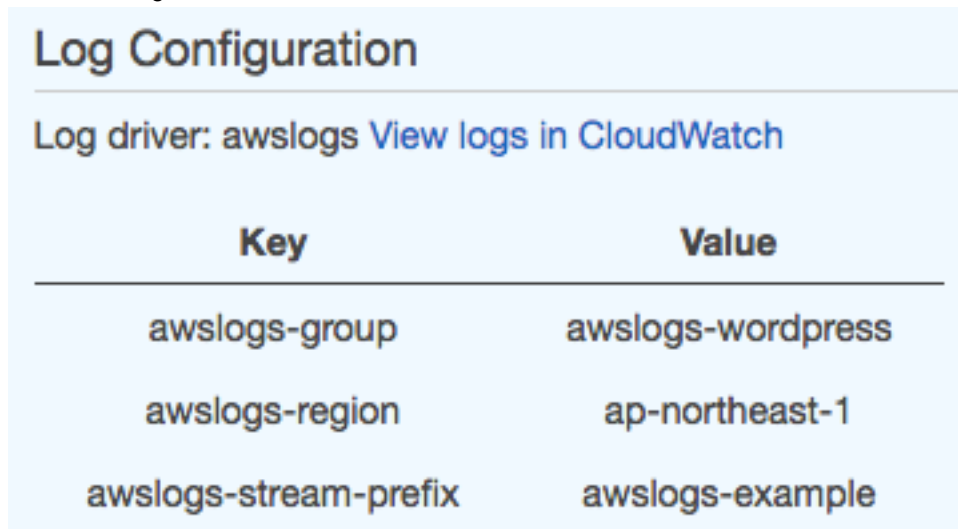
After you have registered a task definition with the `awslogs` log driver in a container definition log configuration, you can run a task or create a service with that task definition to start sending logs to CloudWatch Logs. For more information, see Running Tasks (p. 118) and Creating a Service (p. 142).

Viewing awslogs Container Logs in CloudWatch Logs

After your container instance role has the proper permissions to send logs to CloudWatch Logs, your container agents are updated to at least version 1.9.0, and you have configured and started a task with containers that use the `awslogs` log driver, your configured containers should be sending their log data to CloudWatch Logs. You can view and search these logs in the console.

To view your CloudWatch Logs data for a container from the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, select the cluster that contains the task to view.
3. On the **Cluster:** *cluster_name* page, choose **Tasks** and select the task to view.
4. On the **Task:** *task_id* page, expand the container view by choosing the arrow to the left of the container name.
5. In the **Log Configuration** section, choose **View logs in CloudWatch**, which opens the associated log stream in the CloudWatch console.

To view your CloudWatch Logs data in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**.
3. Select a log group to view. You should see the log groups that you created in Creating Your Log Groups (p. 108).

4. Choose a log stream to view.

Example Task Definitions

Below are some task definition examples that you can use to start creating your own task definitions. For more information, see Task Definition Parameters (p. 88) and Creating a Task Definition (p. 84).

Topics

WordPress and MySQL

The following task definition specifies a WordPress container and a MySQL container that are linked together. These WordPress container exposes the container port 80 on the host port 80. The security group on the container instance would need to open port 80 in order for this WordPress installation to be accessible from a web browser.

For more information about the WordPress container, go to the official WordPress Docker Hub repository at https://registry.hub.docker.com/_/wordpress/. For more information about the MySQL container, go to the official MySQL Docker Hub repository at https://registry.hub.docker.com/_/mysql/.

> **Important**
>
> If you use this task definition with a load balancer, you need to complete the WordPress setup installation through the web interface on the container instance immediately after the container starts. The load balancer health check ping expects a `200` response from the server, but WordPress returns a `301` until the installation is completed. If the load balancer health check fails, the load balancer deregisters the instance.

```
{
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": [
        "mysql"
      ],
      "image": "wordpress",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "memory": 500,
      "cpu": 10
    },
    {
      "environment": [
        {
          "name": "MYSQL_ROOT_PASSWORD",
          "value": "password"
        }
      ],
      "name": "mysql",
      "image": "mysql",
      "cpu": 10,
      "memory": 500,
      "essential": true
    }
  ],
  "family": "hello_world"
```

```
}
```

awslogs Log Driver

The following example demonstrates how to use the `awslogs` log driver in a task definition. The `nginx` container will send its logs to the `ecs-log-streaming` log group in the `us-west-2` region. For more information, see Using the awslogs Log Driver (p. 107).

```
{
  "containerDefinitions": [
    {
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "name": "nginx-container",
      "image": "nginx",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "ecs-log-streaming",
          "awslogs-region": "us-west-2"
        }
      },
      "cpu": 0
    }
  ],
  "family": "example_task_1"
}
```

Amazon ECR Image and Task Definition IAM Role

The following example uses an Amazon ECR image called `aws-nodejs-sample` with the `v1` tag from the `123456789012.dkr.ecr.us-west-2.amazonaws.com` registry. The container in this task will inherit IAM permissions from the `arn:aws:iam::123456789012:role/ AmazonECSTaskS3BucketRole` role. For more information, see IAM Roles for Tasks (p. 186).

```
{
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-
sample:v1",
      "memory": "200",
      "cpu": "10",
      "essential": true
    }
  ],
  "family": "example_task_3",
  "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
```

```
}
```

Entrypoint with Command

The following example demonstrates the syntax for a Docker container that uses an entry point and a command argument. This container will ping `google.com` 4 times and then exit.

```
{
  "containerDefinitions": [
    {
      "memory": 32,
      "essential": true,
      "entryPoint": [
        "ping"
      ],
      "name": "alpine_ping",
      "readonlyRootFilesystem": true,
      "image": "alpine:3.4",
      "command": [
        "-4",
        "google.com"
      ],
      "cpu": 16
    }
  ],
  "family": "example_task_2"
}
```

Deregistering Task Definitions

If you decide that you no longer need a task definition in Amazon ECS, you can deregister the task definition so that it no longer displays in your `ListTaskDefinition` API calls or in the console when you want to run a task or update a service.

When you deregister a task definition, it is immediately marked as `INACTIVE`. Existing tasks and services that reference an `INACTIVE` task definition continue to run without disruption, and existing services that reference an `INACTIVE` task definition can still scale up or down by modifying the service's desired count.

You cannot use an `INACTIVE` task definition to run new tasks or create new services, and you cannot update an existing service to reference an `INACTIVE` task definition (although there may be up to a 10 minute window following deregistration where these restrictions have not yet taken effect).

Use the following procedure to deregister a task definition.

To deregister a task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, choose the region that contains your task definition.
3. In the navigation pane, choose **Task Definitions**.
4. On the **Task Definitions** page, choose the task definition name that contains one or more revisions that you want to deregister.
5. On the **Task Definition name** page, select the box to the left of each task definition revision you want to deregister.

6. Choose **Actions**, and then choose **Deregister**.
7. Verify the information in the **Deregister Task Definition** window, and choose **Deregister** to finish.

Scheduling Amazon ECS Tasks

Amazon EC2 Container Service (Amazon ECS) is a shared state, optimistic concurrency system that provides flexible scheduling capabilities for your tasks and containers. The Amazon ECS schedulers leverage cluster state information provided by the Amazon ECS API to make an appropriate placement decision. Amazon ECS provides the service scheduler (for long-running tasks and applications), and the `RunTask` action (for batch jobs or single run tasks), which place tasks on your cluster for you, as well as the `StartTask` action, which allows you to specify a container instance for the task, so you can integrate with custom, third-party schedulers or use to place a task manually on a specific container instance.

Services

The service scheduler is ideally suited for long running stateless services and applications. The service scheduler ensures that the specified number of tasks are constantly running and reschedules tasks when a task fails (for example, if the underlying container instance fails for some reason). The service scheduler optionally also makes sure that tasks are registered against an Elastic Load Balancing load balancer. You can update your services that are maintained by the service scheduler, such as deploying a new task definition, or changing the running number of desired tasks. For more information, see Services (p. 121).

Running Tasks

The `RunTask` action is ideally suited for processes such as batch jobs that perform work and then stop. `RunTask` randomly distributes tasks across your cluster and tries to minimize the chances that a single instance on your cluster will get a disproportionate number of tasks. For example, you could have a process that calls `RunTask` when work comes into a queue. The task pulls work from the queue, performs the work such as a data transformation, and then exits. For more information, see Running Tasks (p. 118).

The `StartTask` API

In addition to providing a set of default schedulers, Amazon ECS also allows you to write your own schedulers that meet the needs of your business, or leverage third party schedulers. The ECSSchedulerDriver is an open source proof of concept that shows you how can integrate Amazon ECS with third-party schedulers; in this case, with the open source Apache Mesos framework. To write your own scheduler, you can use the Amazon ECS `List` and `Describe` actions to get the state of your cluster and then use the `StartTask` action to place your tasks on the appropriate container instance based on your business and application requirements. The `StartTask` action is available in the AWS CLI, the AWS SDKs, or the Amazon ECS API. For more information, see StartTask in the Amazon EC2 Container Service API Reference.

Topics

-

Running Tasks

Running tasks manually is ideal in certain situations. Perhaps you are developing a task and you are not ready to deploy this task with the service scheduler, or perhaps your task is a one-time or periodic batch job that does not make sense to keep running or restart if it finishes. Use the following procedure to use the default Amazon ECS scheduler to randomly place your task within your cluster.

Note
If you want a specified number of tasks to always remain running or if you want to place your tasks behind a load balancer, you should use the Amazon ECS service scheduler. For more information, see Services (p. 121).

To run a task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region that your cluster is in.
3. In the navigation pane, select **Task Definitions**.
4. On the **Task Definitions** page, choose the task definition that you want to run.
   - To run the latest revision of a task definition shown here, check the box to the left of the name of the task definition that you want to run.
   - To run an earlier revision of a task definition shown here, choose the task definition to view all active revisions, then select the revision to run.
5. Choose **Actions**, and then choose **Run Task**.
6. On the **Run Task** page, select the cluster you would like to use.
7. For **Number of tasks**, enter the number of tasks to launch with this task definition.
8. (Optional) To send command or environment variable overrides to one or more containers in your task definition, or to specify an IAM role task override, complete the following steps:
   a. Choose the **Advanced Options** menu.
   b. On the **Task Role Override** menu, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 186).

      ### Note
      Only roles that have the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For help creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 188).
   c. On the **Container Overrides** menu, select a container to which to send a command or environment variable override.
      - **For a command override:** In the **Command override** field, type the command override to send. If your container definition does not specify an ENTRYPOINT, the format should be a comma-separated list of non-quoted strings. For example:

        ```
        /bin/sh,-c,echo,$DATE
        ```

        If your container definition does specify an ENTRYPOINT (such as **sh,-c**), the format should be an unquoted string, which is surrounded with double quotes and passed as an argument to the ENTRYPOINT command. For example:

        ```
        while true; do echo $DATE > /var/www/html/index.html; sleep 1; done
        ```

- **For environment variable overrides:** Choose **Add Environment Variable**. For **Key**, enter the name of your environment variable. For **Value**, enter the string your environment value should be set to (without surrounding quotes).

The above environment variable override is sent to the container as:

```
MY_ENV_VAR="This variable contains a string."
```

9. Review your task information and choose **Run Task**.

   Note
   If your task moves from PENDING to STOPPED, or if it displays a PENDING status and then disappears from the listed tasks, your task may be stopping due to an error. For more information, see Checking Stopped Tasks for Errors (p. 235) in the troubleshooting section.

Task Life Cycle

When a task is started on a container instance, either manually or as part of a service, it can pass through several states before it finishes on its own or is stopped manually. Some tasks are meant to run as batch jobs that naturally progress through from PENDING to RUNNING to STOPPED. Other tasks, which can be part of a service, are meant to continue running indefinitely, or to be scaled up and down as needed.

When task status changes are requested, such as stopping a task or updating the desired count of a service to scale it up or down, the Amazon ECS container agent tracks these changes as the last known status of the task and the desired status of the task. The flow chart below shows the different paths that task status can take, based on the action that causes the status change.
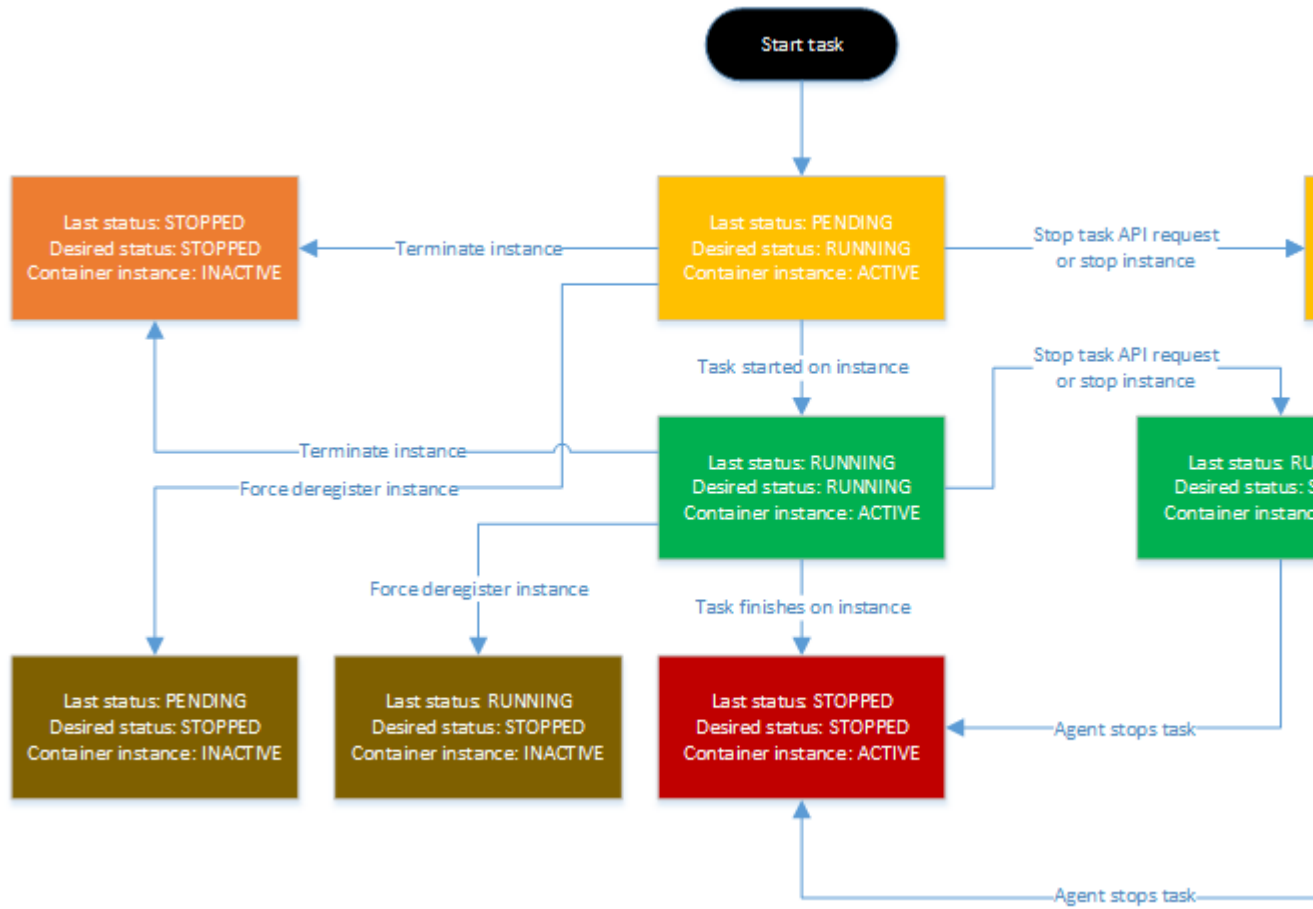
The center path shows the natural progression of a batch job that stops on its own. A persistent task that is not meant to finish would also be on the center path, but it would stop at the RUNNING:RUNNING stage. The paths to the right show what happens at a given state if an API call reaches the agent to stop the task or a container instance. The paths to the left show what happens if the container instance a task is running on is removed, whether by forcefully deregistering it or by terminating the instance.

Services

Amazon ECS allows you to run and maintain a specified number (the "desired count") of instances of a task definition simultaneously in an ECS cluster. This is called a service. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service.

In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

Topics

Service Concepts

- If a task in a service stops, the task is killed and restarted. This process continues until your service reaches the number of desired running tasks.
- You can optionally run your service behind a load balancer. For more information, see Service Load Balancing (p. 124).
- You can optionally specify a deployment configuration for your service. During a deployment (which is triggered by updating the task definition or desired count of a service), the service scheduler uses the minimum healthy percent and maximum percent parameters to determine the deployment strategy. For more information, see Service Definition Parameters (p. 122).
- When the service scheduler launches new tasks, it attempts to balance them across the Availability Zones in your cluster with the following logic:
  - Determine which of the container instances in your cluster can support your service's task definition (for example, they have the required CPU, memory, ports, and container instance attributes).
  - Sort the valid container instances by the fewest number of running tasks for this service in the same Availability Zone as the instance. For example, if zone A has one running service task and

zones B and C each have zero, valid container instances in either zone B or C are considered optimal for placement.

- Place the new service task on a valid container instance in an optimal Availability Zone (based on the previous steps), favoring container instances with the fewest number of running tasks for this service.

Service Definition Parameters

A service definition defines which task definition to use with your service, how many instantiations of that task to run, and which load balancers (if any) to associate with your tasks.

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    }
}
```

Note
You can create the above service definition template with the following AWS CLI command.

```
$ aws ecs create-service --generate-cli-skeleton
```

You can specify the following parameters in a service definition.

`cluster`
    The short name or full Amazon Resource Name (ARN) of the cluster on which to run your service on. If you do not specify a cluster, the default cluster is assumed.

`serviceName`
    The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a region or across multiple regions.

`taskDefinition`
    The `family` and `revision` (`family:revision`) or full ARN of the task definition to run in your service.

`loadBalancers`
    A load balancer object representing the load balancer to use with your service. Currently, you are limited to one load balancer per service. After you create a service, the load balancer name, container name, and container port specified in the service definition are immutable.

For Elastic Load Balancing Classic Load Balancers, this object must contain the load balancer name, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance is registered with the load balancer specified here.

For Elastic Load Balancing Application Load Balancers, this object must contain the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified here.

targetGroupArn
    The full Amazon Resource Name (ARN) of the Elastic Load Balancing target group associated with a service.

loadBalancerName
    The name of the load balancer.

containerName
    The name of the container (as it appears in a container definition) to associate with the load balancer.

containerPort
    The port on the container to associate with the load balancer. This port must correspond to a containerPort in the service's task definition. Your container instances must allow ingress traffic on the hostPort of the port mapping.

desiredCount
    The number of copies of the specified task definition to place and keep running on your cluster.

clientToken
    Unique, case-sensitive identifier you provide to ensure the idempotency of the request. Up to 32 ASCII characters are allowed.

role
    The name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only required if you are using a load balancer with your service.

    If your specified role has a path other than /, then you must either specify the full role ARN (this is recommended) or prefix the role name with the path. For example, if a role with the name bar has a path of /foo/ then you would specify /foo/bar as the role name. For more information, see Friendly Names and Paths in the *IAM User Guide*.

deploymentConfiguration
    Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

maximumPercent
    The maximumPercent parameter represents an upper limit on the number of your service's tasks that are allowed in the RUNNING or PENDING state during a deployment, as a percentage of the desiredCount (rounded down to the nearest integer). This parameter enables you to define the deployment batch size. For example, if your service has a desiredCount of four tasks and a maximumPercent value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for maximumPercent is 200%.

    The maximum number of tasks during a deployment is the desiredCount multiplied by the maximumPercent/100, rounded down to the nearest integer value.

minimumHealthyPercent
    The minimumHealthyPercent represents a lower limit on the number of your service's tasks that must remain in the RUNNING state during a deployment, as a percentage of the desiredCount (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a desiredCount of four tasks and a minimumHealthyPercent of 50%, the scheduler may stop two existing

tasks to free up cluster capacity before starting two new tasks. Tasks for services that *do not* use a load balancer are considered healthy if they are in the RUNNING state; tasks for services that *do* use a load balancer are considered healthy if they are in the RUNNING state and the container instance it is hosted on is reported as healthy by the load balancer. The default value for minimumHealthyPercent is 50% in the console and 100% for the AWS CLI, the AWS SDKs, and the APIs.

The minimum healthy tasks during a deployment is the desiredCount multiplied by the minimumHealthyPercent/100, rounded up to the nearest integer value.

Service Load Balancing

Your Amazon ECS service can optionally be configured to use Elastic Load Balancing to distribute traffic evenly across the tasks in your service.

Elastic Load Balancing provides two types of load balancers: Application Load Balancers and Classic Load Balancers.

Application Load Balancer

An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster. Application Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Application Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the Application Load Balancer Guide.
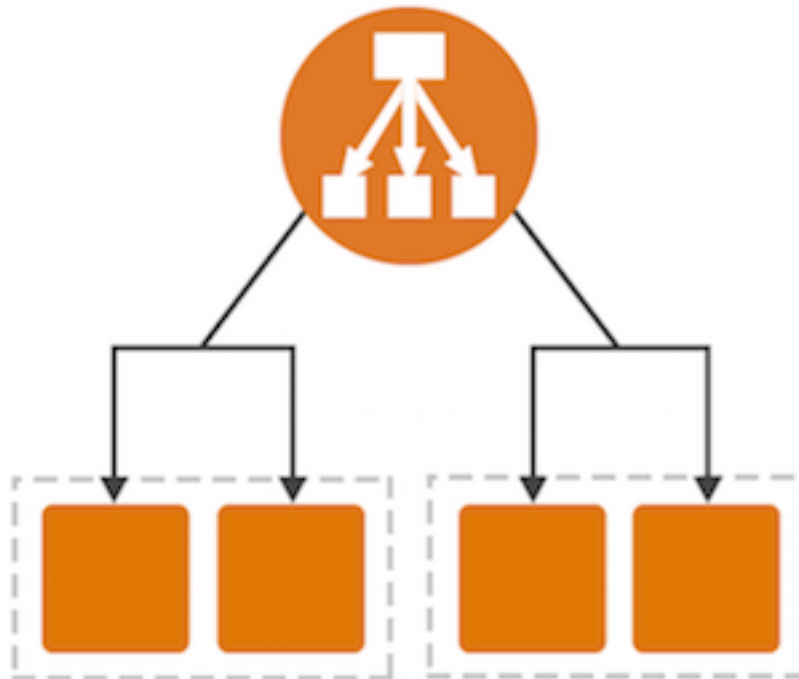
Classic Load Balancer

A Classic Load Balancer makes routing decisions at either the transport layer (TCP/SSL) or the application layer (HTTP/HTTPS). Classic Load Balancers currently require a fixed relationship between the load balancer port and the container instance port. For example, it is possible to map the load balancer port 80 to the container instance port 3030 and the load balancer port 4040 to the container instance port 4040. However, it is not possible to map the load balancer port 80 to port 3030 on one container instance and port 4040 on another container instance. This static mapping requires that your cluster has at least as many container instances as the desired count of a single service that uses a Classic Load Balancer. For more information, see the Classic Load Balancer Guide.

Amazon ECS services can use either type of load balancer. However, Application Load Balancers offer several new features that make them particularly attractive for use with Amazon ECS services:

• Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance).

• Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).

We recommend using Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features. For more information about Elastic Load Balancing and the differences between the two load balancer types, see the Elastic Load Balancing User Guide.

Topics

Load Balancing Concepts

- All of the containers that are launched in a single task definition are always placed on the same container instance. For Classic Load Balancers, you may choose to put multiple containers (in the same task definition) behind the same load balancer by defining multiple host ports in the service definition and adding those listener ports to the load balancer. For example, if a task definition consists of Elasticsearch using port 3030 on the container instance, with Logstash and Kibana using port 4040 on the container instance, the same load balancer can route traffic to Elasticsearch and Kibana through two listeners. For more information, see Listeners for Your Classic Load Balancer in the *Classic Load Balancer Guide*.

  Important
  We do not recommend connecting multiple services to the same Classic Load Balancer. Because entire container instances are registered and deregistered with Classic Load Balancers (and not host and port combinations), this configuration can cause issues if a task from one service stops, causing the entire container instance to be deregistered from the Classic Load Balancer while another task from a different service on the same container instance is still using it. If you want to connect multiple services to a single load balancer (for example, to save costs), we recommend using an Application Load Balancer.

- There is a limit of one load balancer or target group per service.

- Your load balancer subnet configuration must include all subnets that your container instances reside in.

- After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

- If a service's task fails the load balancer health check criteria, the task is killed and restarted. This process continues until your service reaches the number of desired running tasks.

- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting Service Load Balancers (p. 239).

Check the Service Role for your Account

Amazon ECS needs permission to register and deregister container instances with your load balancer when tasks are created and stopped.

In most cases, the Amazon ECS service role is automatically created for you in the Amazon ECS console first run experience. You can use the following procedure to check and see if your account already has an Amazon ECS service role.

To check for the `ecsServiceRole` in the IAM console

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Search the list of roles for `ecsServiceRole`. If the role does not exist, see Amazon ECS Service Scheduler IAM Role (p. 182) to create the role. If the role does exist, select the role to view the attached policies.

4. Choose **Permissions**.

5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.

    a.   Choose **Attach Policy**.

    b.   For **Filter**, type `AmazonEC2ContainerServiceRole` to narrow the available policies to attach.

    c.   Select the box to the left of the **AmazonEC2ContainerServiceRole** policy and choose **Attach Policy**.

6.   Choose **Trust Relationships**, **Edit Trust Relationship**.

7.   Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating a Load Balancer

This section provides a hands-on introduction to using Elastic Load Balancing through the AWS Management Console to use with your Amazon ECS services. In this section, you create an external load balancer that receives public HTTP traffic and routes it to your Amazon ECSinstances.

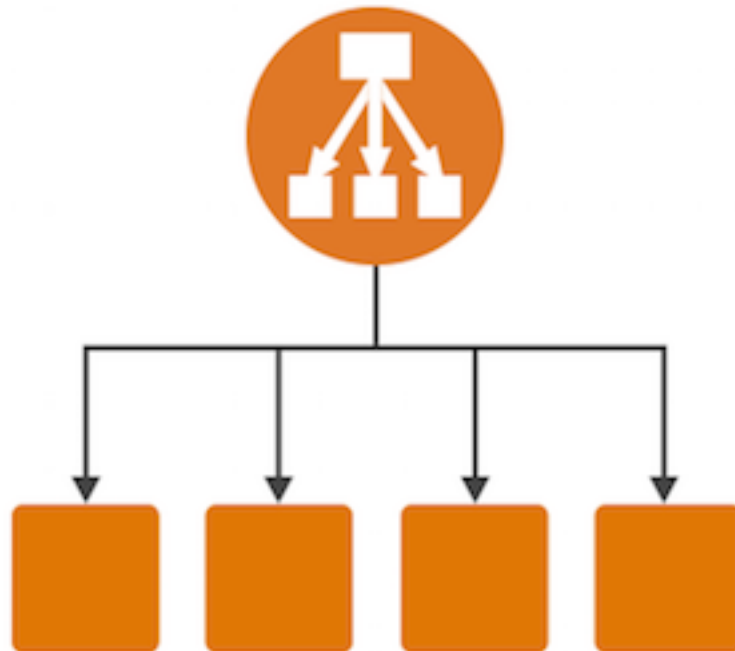Elastic Load Balancing provides two types of load balancers: Application Load Balancers and Classic Load Balancers, and Amazon ECS services can use either type of load balancer. However, Application Load Balancers offer several new features that make them particularly attractive for use with Amazon ECS services:

- Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance).
- Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features. For more information about Elastic Load Balancing and the differences between the two load balancer types, see the Elastic Load Balancing User Guide.

Topics
- Creating an Application Load Balancer (p. 127)
- Creating a Classic Load Balancer (p. 131)

Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and a port for front-end (client to load balancer) connections, and protocol and a port for back-end (load balancer to back-end instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the containers in your tasks on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.

3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.

4. Choose **Create Load Balancer**.

5. On the **Select load balancer type** page, choose **Application Load Balancer** and then choose **Continue**.

6. Complete the **Configure Load Balancer** page as follows:

   a. For **Name**, type a name for your load balancer.

   b. For **Scheme**, an Internet-facing load balancer routes requests from clients over the Internet to targets. An internal load balancer routes requests to targets using private IP addresses.

   c. For **Listeners**, the default is a listener that accepts HTTP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose **Add** to add another listener.

   d. For **VPC**, select the same VPC that you used for the container instances on which you intend to run your service.

   e. For **Available subnets**, select at least two subnets from different Availability Zones, and choose the icon in the **Actions** column. The subnets are moved under **Selected subnets**. Note that you can select only one subnet per Availability Zone. If you select a subnet from an Availability Zone where there is already a selected subnet, this subnet replaces the currently selected subnet for the Availability Zone. Your load balancer subnet configuration must include all subnets that your container instances reside in.

   f. Choose **Next: Configure Security Settings**.

(Optional) Configure Security Settings

If you created a secure listener in the previous step, complete the **Configure Security Settings** page as follows; otherwise, choose **Next: Configure Security Groups**.

To configure security settings

1. If you have a certificate from AWS Certificate Manager, choose **Choose an existing certificate from AWS Certificate Manager (ACM)**, and then choose the certificate from **Certificate name**.

2. If you have already uploaded a certificate using IAM, choose **Choose an existing certificate from AWS Identity and Access Management (IAM)**, and then choose your certificate from **Certificate name**.

3. If you have a certificate ready to upload, choose **Upload a new SSL Certificate to AWS Identity and Access Management (IAM)**. For **Certificate name**, type a name for the certificate. For **Private Key**, copy and paste the contents of the private key file (PEM-encoded). In **Public Key Certificate**, copy and paste the contents of the public key certificate file (PEM-encoded). In

Certificate Chain, copy and paste the contents of the certificate chain file (PEM-encoded), unless you are using a self-signed certificate and it's not important that browsers implicitly accept the certificate.

4. For **Select policy**, keep the one existing predefined security policy.

5. Choose **Next: Configure Security Groups**.

Configure Security Groups

You must assign a security group to your load balancer that allows inbound traffic to the ports that you specified for your listeners. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.

2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your listener to use.

> **Note**
> Later in this topic, you will create a security group rule for your container instances that allows traffic on all ports coming from the security group created here, so that the Application Load Balancer can route traffic to dynamically assigned host ports on your container instances.

Assign a security group:	● Create a **new** security group
	○ Select an **existing** security group
Security group name:	alb-example
Description:	Port 80 for HTTP ECS service

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	
HTTP ⇕	TCP	80	Anywhere ⇕	0.0.0.0/0

Add Rule

3. Choose **Next: Configure Routing** to go to the next page in the wizard.

Configure Routing

In this section, you create a target group for your load balancer and the health check criteria for targets that are registered within that group.

To create a target group and configure health checks

1. For **Target group**, keep the default, **New target group**.

2. For **Name**, type a name for the new target group.

3. Set **Protocol** and **Port** as needed.

4. For **Health checks**, keep the default health check settings.

5. Choose **Next: Register Targets**.

Register Targets

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.

2. Choose **Next: Review** to go to the next page in the wizard.

Review and Create

Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create a Security Group Rule for your Container Instances

After your Application Load Balancer has been created, you must add an inbound rule to your container instance security group that allows traffic from your load balancer to reach the containers.

To allow inbound traffic from your load balancer to your container instances

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. In the left navigation, choose **Security Groups**.

3. Choose the security group that your container instances use. If you created your container instances by using the Amazon ECS first run wizard, this security group may have the description, **ECS Allowed Ports**.

4. Choose the **Inbound** tab, and then choose **Edit**.

5. For **Type**, choose **All traffic**.

6. For **Source**, choose **Custom**, and then type the name of your Application Load Balancer security group that you created in Configure Security Groups (p. 129). This rule allows all traffic from your Application Load Balancer to reach the containers in your tasks that are registered with your load balancer.

7. Choose **Save** to finish.

Create an Amazon ECS Service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see Creating a Service (p. 142).

Creating a Classic Load Balancer

This section walks you through the process of creating a Classic Load Balancer in the AWS Management Console.

Note that you can create your Classic Load Balancer for use with EC2-Classic or a VPC. Some of the tasks described in these procedures apply only to load balancers in a VPC.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and a port for front-end (client to load balancer) connections and a protocol, and protocol and a port for back-end (load balancer to back-end instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the back-end instances on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.

3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.

4. Choose **Create Load Balancer**.

5. On the **Select load balancer type** page, choose **Classic Load Balancer**.

6. For **Load Balancer name**, enter a unique name for your load balancer.

   The load balancer name you choose must be unique within your set of load balancers, must have a maximum of 32 characters, and must only contain alphanumeric characters or hyphens.

7. For **Create LB inside**, select the same network that your container instances are located in: EC2-Classic or a specific VPC.

8. The default values configure an HTTP load balancer that forwards traffic from port 80 at the load balancer to port 80 of your container instances, but you can modify these values for your application. For more information, see Listeners for Your Classic Load Balancer in the *Classic Load Balancer Guide*.

9. [EC2-VPC] To improve the availability of your load balancer, select at least two subnets in different Availability Zones. Your load balancer subnet configuration must include all subnets that your container instances reside in. In the **Select Subnets** section, under **Available Subnets**, select the subnets. The subnets that you select are moved under **Selected Subnets**.

   Note
   If you selected EC2-Classic as your network, or you have a default VPC but did not choose **Enable advanced VPC configuration**, you do not see **Select Subnets**.

Available Subnets				
Actions	**Availability Zone**	**Subnet ID**	**Subnet CIDR**	**Name**
➕	us-west-2c	subnet-cb663da2	10.0.1.0/24	
➕	us-west-2c	subnet-c9663da0	10.0.0.0/24	

Selected Subnets				
Actions	**Availability Zone**	**Subnet ID**	**Subnet CIDR**	**Name**
➖	us-west-2a	subnet-e4f33493	10.0.2.0/24	
➖	us-west-2b	subnet-5264e837	10.0.3.0/24	

10. Choose **Next: Assign Security Groups** to go to the next page in the wizard.

Assign a Security Group to Your Load Balancer in a VPC

If you created your load balancer in a VPC, you must assign it a security group that allows inbound traffic to the ports that you specified for your load balancer and the health checks for your load balancer. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

Note
If you selected EC2-Classic as your network, you do not see this page in the wizard and you can go to the next step. Elastic Load Balancing provides a security group that is assigned to your load balancer for EC2-Classic automatically.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.

2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your load balancer to use. If you specified a different port for the health checks, you must choose **Add Rule** to add a rule that allows inbound traffic to that port as well.

   ### Note
   You should also assign this security group to container instances in your service, or another security group with the same rules.

3. Choose **Next: Configure Security Settings** to go to the next page in the wizard.

(Optional) Configure Security Settings

For this tutorial, you can choose **Next: Configure Health Check** to continue to the next step. For more information about creating a HTTPS load balancer and using additional security features, see HTTPS Load Balancers in the *Classic Load Balancer Guide*.

Configure Health Checks for Your EC2 Instances

Elastic Load Balancing automatically checks the health of the tasks in your service. If Elastic Load Balancing finds an unhealthy task, it stops sending traffic to the instance and reroutes traffic to healthy instances. Amazon ECS stops your unhealthy task and starts another instance of that task.

Note
The following procedure configures an HTTP (port 80) load balancer, but you can modify these values for your application.

To configure a health check for your instances

1. On the **Configure Health Check** page, do the following:

   a. Leave **Ping Protocol** set to its default value of `HTTP`.

   b. Leave **Ping Port** set to its default value of `80`.

   c. For **Ping Path**, replace the default value with a single forward slash ("/"). This tells Elastic Load Balancing to send health check queries to the default home page for your web server, such as `index.html` or `default.html`.

   d. Leave the other fields at their default values.

Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Protocol	HTTP
Ping Port	80
Ping Path	/

2. Choose **Next: Add EC2 Instances** to go to the next page in the wizard.

Load Balancer Instance Registration

Your load balancer distributes traffic between the instances that are registered to it. When you assign your load balancer to an Amazon ECS service, Amazon ECS automatically registers and deregisters container instances when tasks from your service are running on them. Because Amazon ECS handles container instance registration, you do not add container instances to your load balancer at this time.

To skip instance registration and tag the load balancer

1. On the **Add EC2 Instances** page, for **Add Instances to Load Balancer**, ensure that no instances are selected for registration.
2. Leave the other fields at their default values.
3. Choose **Next: Add Tags** to go to the next page in the wizard.

Tag Your Load Balancer (Optional)

You can tag your load balancer, or continue to the next step. Note that you can tag your load balancer later on; for more information, see Tag Your Classic Load Balancer in the *Classic Load Balancer Guide*.

To add tags to your load balancer

1. On the **Add Tags** page, specify a key and a value for the tag.
2. To add another tag, choose **Create Tag** and specify a key and a value for the tag.

3. After you are finished adding tags, choose **Review and Create**.

Create and Verify Your Load Balancer

Before you create the load balancer, review the settings that you selected. After creating the load balancer, you can create a service that uses it to verify that it's sending traffic to your container instances.

To finish creating your load balancer

1. On the **Review** page, check your settings. If you need to make changes to the initial settings, choose the corresponding edit link.
2. Choose **Create** to create your load balancer.
3. After you are notified that your load balancer was created, choose **Close**.

Create an Amazon ECS Service

After your load balancer is created, you can specify it in a service definition when you create a service. For more information, see Creating a Service (p. 142).

Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling is available in all regions that support Amazon ECS.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these service utilization metrics to scale your service up to deal with high demand at peak times, and to scale your service down to reduce costs during periods of low utilization. For more information, see Service Utilization (p. 157).

You can also use CloudWatch metrics published by other services, or custom metrics that are specific to your application. For example, a web service could increase the number of tasks based on Elastic Load Balancing metrics such as `SurgeQueueLength`, and a batch job could increase the number of tasks based on Amazon SQS metrics like `ApproximateNumberOfMessagesVisible`.

You can also use Service Auto Scaling in conjunction with Auto Scaling for Amazon EC2 on your ECS cluster to scale your cluster, and your service, as a result to the demand. For more information, see Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 162).

Service Auto Scaling Required IAM Permissions

Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling. IAM users must have the appropriate permissions for these services before they can use Service Auto Scaling in the AWS Management Console or with the AWS CLI or SDKs. In addition to the standard IAM permissions for creating and updating services, Service Auto Scaling requires the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:*",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The Create Services (p. 197) and Update Services (p. 198) IAM policy examples show the permissions that are required for IAM users to use Service Auto Scaling in the AWS Management Console.

The Application Auto Scaling service needs permission to describe your ECS services and CloudWatch alarms, as well as permissions to modify your service's desired count on your behalf. You must create an IAM role (`ecsAutoscaleRole`) for your ECS services to provide these permissions and then associate that role with your service before it can use Application Auto Scaling. If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p.   ) or update service (p. 147) workflows, and then use the role for any other service later (in the console or with the CLI/SDKs). You can also create the role by following the procedures in Amazon ECS Service Auto Scaling IAM Role (p. 184).

Service Auto Scaling Concepts

- The ECS service scheduler respects the desired count at all times, but as long as you have active scaling policies and alarms on a service, Service Auto Scaling could change a desired count that was manually set by you.
- If a service's desired count is set below its minimum capacity value, and an alarm triggers a scale out activity, Application Auto Scaling scales the desired count up to the minimum capacity value and then continues to scale out as required, based on the scaling policy associated with the alarm. However, a scale in activity will not adjust the desired count, because it is already below the minimum capacity value.
- If a service's desired count is set above its maximum capacity value, and an alarm triggers a scale in activity, Application Auto Scaling scales the desired count down to the maximum capacity value and then continues to scale in as required, based on the scaling policy associated with the alarm. However, a scale out activity will not adjust the desired count, because it is already above the maximum capacity value.
- During scaling activities, the actual running task count in a service is the value that Service Auto Scaling uses as its starting point, as opposed to the desired count, which is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that could not be satisfied, for example, if there are not enough container instance resources to place the additional tasks. If the container instance capacity is available later, the pending scaling activity may succeed, and then further scaling activities can continue after the cool down period.

Amazon ECS Console Experience

The Amazon ECS console's service creation and service update workflows support Service Auto Scaling. The ECS console handles the `ecsAutoscaleRole` and policy creation, provided that the IAM user who is using the console has the permissions described in Service Auto Scaling Required IAM Permissions (p. 134), and that they can create IAM roles and attach policies to them.

When you configure a service to use Service Auto Scaling in the console, your service is automatically registered as a scalable target with Application Auto Scaling so that you can configure scaling policies that scale your service up and down. You can also create and update the scaling policies and CloudWatch alarms that trigger them in the Amazon ECS console.

To create a new ECS service that uses Service Auto Scaling, see Creating a Service (p. 142).

To update an existing service to use Service Auto Scaling, see Updating a Service (p. 147).

AWS CLI and SDK Experience

You can configure Service Auto Scaling by using the AWS CLI or the AWS SDKs, but you must observe the following considerations.

- Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling. For more information about these specific API operations, see the Amazon EC2 Container Service API Reference, the Amazon CloudWatch API Reference, and the Application Auto Scaling API Reference. For more information about the AWS CLI commands for these services, see the ecs, cloudwatch, and application-autoscaling sections of the AWS Command Line Interface Reference.
- Before your service can use Service Auto Scaling, you must register it as a scalable target with the Application Auto Scaling RegisterScalableTarget API operation.
- After your ECS service is registered as a scalable target, you can create scaling policies with the Application Auto Scaling PutScalingPolicy API operation to specify what should happen when your CloudWatch alarms are triggered.
- After you create the scaling policies for your service, you can create the CloudWatch alarms that trigger the scaling events for your service with the CloudWatch PutMetricAlarm API operation.

Tutorial: Service Auto Scaling with CloudWatch Service Utilization Metrics

The following procedures help you to create an Amazon ECS cluster and a service that uses Service Auto Scaling to scale up (and down) using CloudWatch alarms.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these service utilization metrics to scale your service up to deal with high demand at peak times, and to scale your service down to reduce costs during periods of low utilization. For more information, see Service Utilization (p. 157).

In this tutorial, you create a cluster and a service (that runs behind an Elastic Load Balancing load balancer) using the Amazon ECS first run wizard. Then you configure Service Auto Scaling on the service with CloudWatch alarms that use the `CPUUtilization` metric to scale your service up or down, depending on the current application load.

When the CPU utilization of your service rises above 75% (meaning that more than 75% of the CPU that is reserved for the service is being used), the scale out alarm triggers Service Auto Scaling to add another task to your service to help out with the increased load. Conversely, when the CPU utilization of your service drops below 25%, the scale in alarm triggers a decrease in the service's desired count to free up those cluster resources for other tasks and services.

Prerequisites

This tutorial assumes that you have an AWS account and an IAM administrative user with permissions to perform all of the actions described within, and an Amazon EC2 key pair in the current region. If

you do not have these resources, or your are not sure, you can create them by following the steps in
Setting Up with Amazon ECS (p. 8).

Your Amazon ECS container instances also require `ecs:StartTelemetrySession` permission on
the IAM role that you launch your container instances with. If you created your Amazon ECS container
instance role before CloudWatch metrics were available for Amazon ECS, then you might need to
add this permission. For information about checking your Amazon ECS container instance role and
attaching the managed IAM policy for container instances, see To check for the `ecsInstanceRole` in
the IAM console (p. 181).

Step 1: Create a Cluster and a Service

After you have enabled CloudWatch metrics for your clusters and services, you can create a cluster
and service using the Amazon ECS first run wizard. The first run wizard takes care of creating the
necessary IAM roles and policies for this tutorial, an Auto Scaling group for your container instances,
and it creates a service that runs behind a load balancer. The wizard also makes the later clean up
process much easier, because you can delete the entire AWS CloudFormation stack in one step.

For this tutorial, you create a cluster called `service-autoscaling` and a service called `sample-webapp`.

To create your cluster and service

1. Open the Amazon ECS console first run wizard at https://console.aws.amazon.com/ecs/home#/
    firstRun.
2. By default, you are given the option to create an image repository and push an image to Amazon
    ECR.
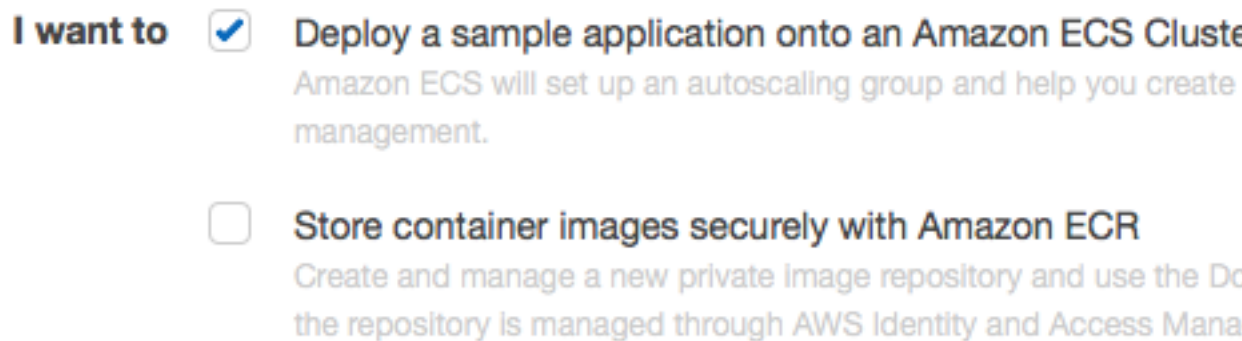
    For this tutorial, you will not use Amazon ECR, so be sure to clear the lower option. Choose
    Continue to proceed.
3. On the **Create a task definition** page, leave all of the default options and choose **Next step**.
4. On the **Configure service** page, for **Container name: host port**, choose **simple-app:80**.

    > **Important**
    > Elastic Load Balancing load balancers do incur cost while they exist in your AWS
    > resources. For more information, see Elastic Load Balancing Pricing.
5. For **Select IAM role for service**, choose an existing Amazon ECS service (`ecsServiceRole`)
    role that you have already created, or choose **Create new role** to create the required IAM role for
    your service.
6. The remaining default values here are set up for the sample application, so leave them as they are
    and choose **Next step**.
7. On the **Configure cluster** page, enter the following information:

    a. For **Cluster name**, type `service-autoscaling`.

b. For instance type, choose any available instance type. The default `t2.micro` works fine for this tutorial.

c. For **Number of instances**, enter the number of instances to launch into your cluster. For the purposes of this tutorial, two instances are sufficient.

> **Important**
> Your AWS account incurs the standard Amazon EC2 usage fees for these instances from the time that you launch the instances until you terminate them (which is the final task of this tutorial), even if they remain idle.

d. (Optional) For **Key pair**, choose a key pair to use for SSH access to your instances. This is not required, but it can be useful for diagnostic purposes if you need to troubleshoot your instances later.

e. For **Container instance IAM role**, choose an existing Amazon ECS container instance (`ecsInstanceRole`) role that you have already created, or choose **Create new role** to create the required IAM role for your container instances.

f. Choose **Review and Launch** to proceed. Review your configurations and choose **Launch instance & run service** to finish.

You are directed to a **Launch Status** page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).

8. When your cluster and service are created, choose **View service** to view your new service.

Step 2: Configure Service Auto Scaling

Now that you have launched a cluster and created a service in that cluster that is running behind a load balancer, you can configure Service Auto Scaling by creating scaling policies to scale your service up and down in response to CloudWatch alarms.

To configure basic Service Auto Scaling parameters

1. On the **Service: sample-webapp** page, your service configuration should look similar to the image below (although the task definition revision and load balancer name will likely be different). Choose **Update** to update your new service.

Service : sample-webapp

Details

Cluster	service-autoscaling
Status	ACTIVE
Task Definition	console-sample-app-static:6
Desired count	1
Pending count	0
Running count	1
Service Role	ecsServiceRole

Load

Loa

E(
F
SMA

Deplo

Mini

2. On the **Update service** page, choose **Configure Service Auto Scaling**.

3. For **Service Auto Scaling**, choose **Configure Service Auto Scaling to adjust your service's desired count**.

Service Auto Scaling   ○   Do not adjust the service's desired count

   ●   Configure Service Auto Scaling to adjust yo
        service's desired count

4. For **Minimum number of tasks**, enter 1 for the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted below this amount.

5. For **Desired number of tasks**, this field is pre-populated with the value you entered earlier. This value must be between the minimum and maximum number of tasks specified on this page. Leave this value at 1.

6. For **Maximum number of tasks**, enter 2 for the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted above this amount.

7. For **IAM role for Service Auto Scaling**, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose **Create new role** and the role is created for you. For future reference, the role that is created for you is called `ecsAutoscaleRole`. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 184).

To configure scaling policies for your service

These steps will help you create scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale out alarm to increase the desired count of your service, and a scale in alarm to decrease the desired count of your service.

1. For **Policy name**, enter `ScaleOutPolicy`

2. For **Execute policy when**, choose **Create new alarm**.

   a. For **Alarm name**, enter `sample-webapp-cpu-gt-75`.

   b. For **ECS service metric**, choose **CPUUtilization**.

   c. For **Alarm threshold**, enter the following information to match the image below. This causes the CloudWatch alarm to trigger when the service's CPU utilization is greater than 75% for one minute.

   d. Choose **Save** to save your alarm.

3. For **Scaling action**, enter the following information to match the image below. This causes your service's desired count to increase by 1 task when the alarm is triggered.

4. For **Cooldown period**, enter 60 for the number of seconds between scaling actions and choose **Save** to save your `ScaleOutPolicy`.

5. After you return to the **Service Auto Scaling (optional)** page, choose **Add scaling policy** to configure your `ScaleInPolicy`.

6. For **Policy name**, enter `ScaleInPolicy`

7. For **Execute policy when**, choose **Create new alarm**.

   a. For **Alarm name**, enter `sample-webapp-cpu-lt-25`.

   b. For **ECS service metric**, choose **CPUUtilization**.

   c. For **Alarm threshold**, enter the following information to match the image below. This causes the CloudWatch alarm to trigger when the service's CPU utilization is less than 25% for one minute.

Alarm threshold: Average ▼ of CPUUtilization < ▼ 25

for 1 consecutive periods of 1 minute ▼

     d.   Choose **Save** to save your alarm.

8.   For **Scaling action**, enter the following information to match the image below. This causes your service's desired count to decrease by 1 task when the alarm is triggered.

Scaling action: Remove ▼ 1 tasks ▼ when 25

9.   For **Cooldown period**, enter 60 for the number of seconds between scaling actions and choose **Save** to save your `ScaleOutPolicy`.

10.   After you return to the **Service Auto Scaling (optional)** page, choose **Update Service** to finish your Service Auto Scaling configuration.

11.   When your service status is finished updating, choose **View Service**.

Step 3: Trigger a Scaling Activity

After your service is configured with Service Auto Scaling, you can trigger a scaling activity by pushing your service's CPU utilization into the `ALARM` state. Because the example in this tutorial is a web application that is running behind a load balancer, you can send thousands of HTTP requests to your service (using the ApacheBench utility) to spike the service CPU utilization above our threshold amount. This spike should trigger the alarm, which in turn triggers a scaling activity to add one task to your service.

After the ApacheBench utility finishes the requests, the service CPU utilization should drop below your 25% threshold, triggering a scale in activity that returns the service's desired count to 1.

To trigger a scaling activity for your service

1.   From your service's main view page in the console, choose the load balancer name to view its details in the Amazon EC2 console. You need the load balancer's DNS name, which should look something like this: `EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com`.

2.   Use the ApacheBench (**ab**) utility to make thousands of HTTP requests to your load balancer in a short period of time.

       Note

       This command is installed by default on Mac OSX, and it is available for many Linux distributions, as well. For example, you can install **ab** on Amazon Linux with the following command:

```
$ sudo yum install -y httpd24-tools
```

       Run the following command, substituting your load balancer's DNS name.

```
$ ab -n 100000 -c 1000 http://EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-
east-1.elb.amazonaws.com/
```

3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

4. Choose **Alarms** in the left navigation pane.

5. Wait for your **ab** HTTP requests to trigger the scale out alarm in the CloudWatch console. You should see your Amazon ECS service scale out and add 1 task to your service's desired count.

6. Shortly after your **ab** HTTP requests complete (between 1 and 2 minutes), your scale in alarm should trigger and the scale in policy reduces your service's desired count back to 1.

Step 5: Cleaning Up

When you have completed this tutorial, you may choose to keep your cluster, Auto Scaling group, load balancer, and EC2 instances. However, if you are not actively using these resources, you should consider cleaning them up so that your account does not incur unnecessary charges.

To delete your cluster and CloudWatch alarms

1. In the Amazon ECS console, switch to **Clusters** in the left navigation pane.

2. On the **Clusters** page, choose the **x** in the upper right hand corner of the **service-autoscaling** cluster to delete the cluster.

3. Review and choose **Delete** to confirm your cluster deletion. It may take a few minutes for the cluster AWS CloudFormation stack to finish cleaning up.

4. In the CloudWatch console **Alarms** view, select the alarms that begin with **sample-webapp-cpu-** and then choose **Delete** to delete the alarms.

5. Choose **Yes, Delete** to confirm your alarm deletion.

Creating a Service

When you create an Amazon ECS service, you specify several parameters that define what makes up your service and how it should behave. These parameters create a service definition. Use the following procedure to create a service.

> **Note**
> If you want to use an Elastic Load Balancing load balancer to distribute traffic across the containers in your service, see Service Load Balancing (p. 124) to choose a load balancer type and create the required resources in advance.
> You must verify that your container instances can receive traffic from your load balancers.
> You can allow traffic to all ports on your container instances from your load balancer's security group to ensure that traffic can reach any containers that use dynamically assigned ports.

Configuring Basic Service Parameters

All services require some basic configuration parameters that define the service, such as the task definition to use, which cluster the service should run on, how many tasks should be placed for the service, and so on; this is called the service definition. For more information about the parameters defined in a service definition, see Service Definition Parameters (p. 122).

This section covers creating a service with the basic service definition parameters that are required; when you have configured these parameters, you can create your service or move on to the next sections for optional service definition configuration, such as configuring your service to use a load balancer.

To configure the basic service definition parameters

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. On the navigation bar, select the region that your cluster is in.

3. In the navigation pane, choose **Task Definitions**.

4. On the **Task Definitions** page, choose the name of the task definition from which to create your service.

5. On the **Task Definition name** page, choose the revision of the task definition from which to create your service.

6. Review the task definition, and choose **Create Service**.

7. On the **Create Service** page, for **Cluster**, choose the cluster in which to create your service.

8. For **Service name**, enter a unique name for your service.

9. For **Number of tasks**, enter the number of tasks to launch and maintain on your cluster.

    Note
    If your task definition uses static host port mappings on your container instances, then you need at least one container instance with the specified port available in your cluster for each task in your service. This restriction does not apply if your task definition uses dynamic host port mappings. For more information, see `portMappings` (p.   ) in the Task Definition Parameters (p. 88) topic.

10. (Optional) You can specify deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

    *   **Minimum healthy percent**: Specify a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the service's desired number of tasks (rounded up to the nearest integer). For example, if your service has a desired number of four tasks and a minimum healthy percent of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that do not use a load balancer are considered healthy if they are in the `RUNNING` state; tasks for services that do use a load balancer are considered healthy if they are in the `RUNNING` state and the container instance it is hosted on is reported as healthy by the load balancer. The default value for minimum healthy percent is 50% in the console, and 100% with the AWS CLI or SDKs.

    *   **Maximum percent**: Specify an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the service's desired number of tasks (rounded down to the nearest integer). For example, if your service has a desired number of four tasks and a maximum percent value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for maximum percent is 200%.

11. If you do not want to run your service behind a load balancer or configure your service to use Service Auto Scaling, then you can proceed to Review and Create Your Service (p. 147). Otherwise, proceed to the next sections.

(Optional) Configuring Your Service to Use a Load Balancer

If you have an available Elastic Load Balancing load balancer configured, you can attach it to your service with the following procedures, or you can configure a new load balancer in the Amazon EC2 console; for more information see Creating a Load Balancer (p. 127).

Note
You must create your Elastic Load Balancing load balancer resources prior to following these procedures.

First, you must choose the load balancer type to use with your service. Then you can configure your
service to work with the load balancer.

To choose a load balancer type

1. If you have not done so already, follow the basic service creation procedures in Configuring Basic
   Service Parameters (p. 142).

2. On the **Create Service** page, choose **Configure ELB**.

3. Choose the load balancer type to use with your service:

   Application Load Balancer
   > Allows containers to use dynamic host port mapping (multiple tasks allowed per container
   > instance). Multiple services can use the same listener port on a single load balancer with rule-
   > based routing and paths.

   Classic Load Balancer
   > Requires static host port mappings (only one task allowed per container instance); rule-based
   > routing and paths are not supported.

   We recommend that you use Application Load Balancers for your Amazon ECS services so that
   you can take advantage of the advanced features available to them.

4. For **Select IAM role for service**, choose **Create new role** to create a new role for your service, or
   select an existing IAM role to use for your service (by default, this is `ecsServiceRole`).

   > **Important**
   > If you choose to use an existing `ecsServiceRole` IAM role, you must verify that the
   > role has the proper permissions to use Application Load Balancers and Classic Load
   > Balancers, as shown in Amazon ECS Service Scheduler IAM Role (p. 182).

5. For **ELB Name**, choose the name of the load balancer to use with your service. Only load
   balancers that correspond to the load balancer type you selected earlier are visible here.

6. The following steps differ based on the load balancer type for your service. If you've
   chosen an Application Load Balancer, follow the steps in To configure an Application Load
   Balancer (p. 144). If you've chosen a Classic Load Balancer, follow the steps in To configure a
   Classic Load Balancer (p. 145).

To configure an Application Load Balancer

1. For **Select a Container**, choose the container and port combination from your task definition that
   your load balancer should distribute traffic to, and choose **Add to ELB**.

2. For **Listener port**, choose the listener port and protocol of the listener that you created in Creating
   an Application Load Balancer (p. 127) (if applicable), or choose **create new** to create a new
   listener and then enter a port number and choose a port protocol in **Listener protocol**.

3. For **Target group name**, choose the target group that you created in Creating an Application Load
   Balancer (p. 127) (if applicable), or choose **create new** to create a new target group.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:

   • For **Target group name**, enter a name for your target group.

   • For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.

   • For **Path pattern**, if your listener does not have any existing rules, the default path pattern (`/`) is
     used. If your listener already has a default rule, then you must enter a path pattern that matches
     traffic that you want to have sent to your service's target group. For example, if your service is a
     web application called `web-app`, and you want traffic that matches `http://`*`my-elb-url`*`/web-`
     `app` to route to your service, then you would enter `/web-app*` as your path pattern. For more
     information, see ListenerRules in the *Application Load Balancer Guide*.

- For **Health check path**, enter the path to which the load balancer should send health check pings.

5. When you are finished configuring your Application Load Balancer, choose **Save** to save your configuration and proceed to Review and Create Your Service (p. 147).

To configure a Classic Load Balancer

1. The **Health check port**, **Health check protocol**, and **Health check path** fields are all pre-populated with the values you configured in Creating a Classic Load Balancer (p. 131) (if applicable). You can update these settings in the Amazon EC2 console.

2. For **Container for ELB health check**, choose the container to send health checks.

3. When you are finished configuring your Classic Load Balancer, choose **Save** to save your configuration and proceed to Review and Create Your Service (p. 147).

(Optional) Configuring Your Service to Use Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. For more information see Service Auto Scaling (p. 134).

To configure basic Service Auto Scaling parameters

1. If you have not done so already, follow the basic service creation procedures in Configuring Basic Service Parameters (p. 142).

2. On the **Create Service** page, choose **Configure Service Auto Scaling**.

3. On the **Service Auto Scaling** page, select **Configure Service Auto Scaling to adjust your service's desired count**.

4. For **Minimum number of tasks**, enter the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted below this amount.

5. For **Desired number of tasks**, this field is pre-populated with the value you entered earlier. You can change your service's desired count at this time, but this value must be between the minimum and maximum number of tasks specified on this page.

6. For **Maximum number of tasks**, enter the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted above this amount.

7. For **IAM role for Service Auto Scaling**, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose **Create new role** and the role will be created for you. For future reference, the role that is created for you is called `ecsAutoscaleRole`. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 184).

To configure scaling policies for your service

These steps will help you create scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a **Scale out** alarm to increase the desired count of your service, and a **Scale in** alarm to decrease the desired count of your service.

1. For **Policy name**, enter a descriptive name for your policy, or use the default policy name that is already entered.

2. For **Execute policy when**, select the CloudWatch alarm that you want to use to scale your service up or down.

You can use an existing CloudWatch alarm that you have previously created, or you can choose to create a new alarm. The **Create new alarm** workflow allows you to create CloudWatch alarms that are based on the `CPUUtilization` and `MemoryUtilization` of the service that you are creating. To use other metrics, you can create your alarm in the CloudWatch console and then return to this wizard to choose that alarm.

3. (Optional) If you've chosen to create a new alarm, complete the following steps.

   a. For **Alarm name**, enter a descriptive name for your alarm. For example, if your alarm should trigger when your service CPU utilization exceeds 75%, you could call the alarm *service_name*-cpu-gt-75.

   b. For **ECS service metric**, choose the service metric to use for your alarm. For more information about these service utilization metrics, see Service Utilization (p. 157).

   c. For **Alarm threshold**, enter the following information to configure your alarm:

      • Choose the CloudWatch statistic for your alarm (the default value of **Average** works in many cases). For more information, see Statistics in the *Amazon CloudWatch User Guide*.

      • Choose the comparison operator for your alarm and enter the value that the comparison operator checks against (for example, > and 75).

      • Enter the number of consecutive periods before the alarm is triggered and the period length. For example, a 2 consecutive periods of 5 minutes would take 10 minutes before the alarm triggered. Because your Amazon ECS tasks can scale up and down quickly, you should consider using a low number of consecutive periods and a short period duration to react to alarms as soon as possible.

   d. Choose **Save** to save your alarm.

4. For **Scaling action**, enter the following information to configure how your service responds to the alarm:

   • Choose whether to add to, subtract from, or set a specific desired count for your service.

   • If you chose to add or subtract tasks, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is triggered. If you chose to set the desired count, enter the desired count that your service should be set to when the scaling action is triggered.

   • (Optional) If you chose to add or subtract tasks, choose whether the previous value is used as an integer or a percent value of the existing desired count.

   • Enter the lower boundary of your step scaling adjustment. By default, for your first scaling action, this value is the metric amount where your alarm is triggered. For example, the following scaling action adds 100% of the existing desired count when the CPU utilization is greater than 75%.
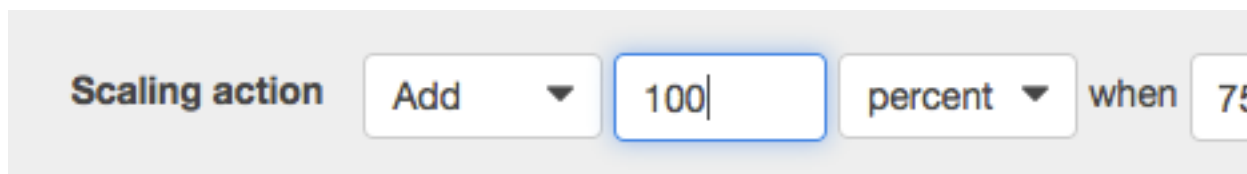
5. (Optional) You can repeat Step 4 (p. 146) to configure multiple scaling actions for a single alarm (for example, to add 1 task if CPU utilization is between 75-85%, and to add 2 tasks if CPU utilization is greater than 85%).

6. (Optional) If you chose to add or subtract a percentage of the existing desired count, enter a minimum increment value for **Add tasks in increments of *N* task(s)**.

7. For **Cooldown period**, enter the number of seconds between scaling actions.

8. Repeat Step 1 (p. 145) through Step 7 (p. 146) for the **Scale in** policy and choose **Save** to save your Service Auto Scaling configuration.

Review and Create Your Service

After you have configured your basic service definition parameters and optionally configured your service to use a load balancer, you can review your configuration and then choose **Create Service** to finish creating your service.

Note
After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

Updating a Service

You can update a running service to change the number of tasks that are maintained by a service or which task definition is used by the tasks. If you have an application that needs more capacity, you can scale up your service to use more of your container instances (as long as they are available). If you have unused capacity that you would like to scale down, you can reduce the number of desired tasks in your service and free up resources.

If you have updated the Docker image of your application, you can create a new task definition with that image and deploy it to your service. The service scheduler uses the minimum healthy percent and maximum percent parameters (in the service's deployment configuration) to determine the deployment strategy.

The minimum healthy percent represents a lower limit on the number of your service's tasks that must remain in the RUNNING state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a desired number of four tasks and a minimum healthy percent of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that *do not* use a load balancer are considered healthy if they are in the RUNNING state; tasks for services that *do* use a load balancer are considered healthy if they are in the RUNNING state and the container instance it is hosted on is reported as healthy by the load balancer. The default value for minimum healthy percent is 50% in the console and 100% for the AWS CLI, the AWS SDKs, and the APIs.

The maximum percent parameter represents an upper limit on the number of your service's tasks that are allowed in the RUNNING or PENDING state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer). This parameter enables you to define the deployment batch size. For example, if your service has a desired number of four tasks and a maximum percent value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for maximum percent is 200%.

When the service scheduler replaces a task during an update, if a load balancer is used by the service, the service first removes the task from the load balancer and waits for the connections to drain. Then the equivalent of **docker stop** is issued to the containers running in the task. This results in a SIGTERM signal and a 30-second timeout, after which SIGKILL is sent and the containers are forcibly stopped. If the container handles the SIGTERM signal gracefully and exits within 30 seconds from receiving it, no SIGKILL signal is sent. The service scheduler starts and stops tasks as defined by your minimum healthy percent and maximum percent settings.

Important
If you are changing the ports used by containers in a task definition, you may need to update your container instance security groups to work with the updated ports.

If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service.

Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To update a running service

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the region that your cluster is in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the name of the cluster that your service resides in.
5. On the **Cluster:** *name* page, choose **Services**.
6. Check the box to the left of the service to update and choose **Update**.
7. On the **Update Service** page, your service information is pre-populated. Change the task definition, deployment configuration, or number of desired tasks (or any combination of these).
8. (Optional) You can use Service Auto Scaling to scale your service up and down automatically in response to CloudWatch alarms.

    a. Under **Optional configurations**, choose **Configure Service Auto Scaling**.
    b. Proceed to Step 3 (p. 145) of (Optional) Configuring Your Service to Use Service Auto Scaling (p. 145).
    c. Complete the steps in that section and then return here.
9. Choose **Update Service** to finish and update your service.

Deleting a Service

You can delete a service if you have no running tasks in it and the desired task count is zero. If the service is actively maintaining tasks, you cannot delete it, and you must update the service to a desired task count of zero. For more information, see Updating a Service (p. 147).

> **Note**
> When you delete a service, if there are still running tasks that require cleanup, the service status moves from `ACTIVE` to `DRAINING`, and the service is no longer visible in the console or in `ListServices` API operations. After the tasks have stopped, then the service status moves from `DRAINING` to `INACTIVE`. Services in the `DRAINING` or `INACTIVE` status can still be viewed with `DescribeServices` API operations; however, in the future, `INACTIVE` services may be cleaned up and purged from Amazon ECS record keeping, and `DescribeServices` API operations on those services will return a `ServiceNotFoundException` error.

Use the following procedure to delete an empty service.

To delete an empty service

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the region that your cluster is in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the name of the cluster that your service resides in.
5. On the **Cluster :** *name* page, choose **Services**.

6. Check the box to the left of the service to update and choose **Delete**.

    Note
    Your service must have zero desired or running tasks before it can be deleted.

7. Choose **Yes, Delete** to confirm your service deletion.

Amazon ECR Repositories

Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. Amazon ECR provides a secure, scalable, and reliable registry. Amazon ECR supports private Docker repositories with resource-based permissions using AWS IAM so that specific users or Amazon EC2 instances can access repositories and images. Developers can use the Docker CLI to author and manage images.

Note
Amazon ECR is available in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
EU (Frankfurt)	eu-central-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

For more information on how to create repositories, push and pull images from Amazon ECR, and set access controls on your repositories, see the Amazon EC2 Container Registry User Guide.

Using Amazon ECR Images with Amazon ECS

You can use your Amazon ECR images with Amazon ECS, but you need to satisfy some prerequisites:

- Your container instances must be using at least version 1.7.0 of the Amazon ECS container agent. The latest version of the Amazon ECS-optimized AMI supports Amazon ECR images in task definitions. For more information, including the latest Amazon ECS-optimized AMI IDs, see Amazon ECS Container Agent Versions (p. 61).
- The Amazon ECS container instance role (`ecsInstanceRole`) that you use with your container instances must possess the following IAM policy permissions for Amazon ECR.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

If you use the `AmazonEC2ContainerServiceforEC2Role` managed policy for your container instances, then your role has the proper permissions. To check that your role supports Amazon ECR, see Amazon ECS Container Instance IAM Role (p. 180).

- In your Amazon ECS task definitions, make sure that you are using the full `registry/repository:tag` naming for your Amazon ECR images. For example, *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest*.

Amazon ECS CloudWatch Metrics

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Topics

Enabling CloudWatch Metrics

Your Amazon ECS container instances require at least version 1.4.0 of the container agent to enable CloudWatch metrics; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 63).

If you are starting your agent manually (for example, if you are not using the Amazon ECS-optimized AMI for your container instances), be sure to add volume mounts for the `cgroup` virtual file system and the `execdriver` path. For more information about an example run command with these volume mounts, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs) (p. 67).

Your Amazon ECS container instances also require `ecs:StartTelemetrySession` permission on the IAM role that you launch your container instances with. If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, then you might need to add this permission. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see To check for the `ecsInstanceRole` in the IAM console (p. 181).

Note

You can disable CloudWatch metrics collection by setting `ECS_DISABLE_METRICS=true` in your Amazon ECS container agent configuration. For more information, see Amazon ECS Container Agent Configuration (p. 69).

Available Metrics and Dimensions

The metrics and dimensions that Amazon ECS sends to Amazon CloudWatch are listed below.

Amazon ECS Metrics

Amazon ECS provides metrics for you to monitor the CPU and memory reservation and utilization across your cluster as a whole, and the CPU and memory utilization on the services in your clusters.

Amazon ECS sends the following metrics to CloudWatch every minute.

Metric	Description
`CPUReservation`	The percentage of CPU units that are reserved by running tasks in the cluster.
	Cluster CPU reservation (this metric can only be filtered by `ClusterName`) is measured as the total CPU units that are reserved by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster.
	Valid Dimensions: `ClusterName, ServiceName`
	Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples.
	Unit: Percent
`CPUUtilization`	The percentage of CPU units that are used in the cluster or service.
	Cluster CPU utilization (metrics that are filtered by `ClusterName` without `ServiceName`) is measured as the total CPU units in use by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster.
	Service CPU utilization (metrics that are filtered by `ClusterName` and `ServiceName`) is measured as the total CPU units in use by the tasks that belong to the service, divided by the total number of CPU units that are reserved for the tasks that belong to the service.
	Valid Dimensions: `ClusterName, ServiceName`
	Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples.
	Unit: Percent

Metric	Description
`MemoryReservation`	The percentage of memory that is reserved by running tasks in the cluster. Cluster memory reservation (this metric can only be filtered by `ClusterName`) is measured as the total memory that is reserved by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. Valid Dimensions: `ClusterName`, `ServiceName` Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples. Unit: Percent
`MemoryUtilization`	The percentage of memory that is used in the cluster or service. Cluster memory utilization (metrics that are filtered by `ClusterName` without `ServiceName`) is measured as the total memory in use by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. Service memory utilization (metrics that are filtered by `ClusterName` and `ServiceName`) is measured as the total memory in use by the tasks that belong to the service, divided by the total memory that is reserved for the tasks that belong to the service. Valid Dimensions: `ClusterName`, `ServiceName` Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples. Unit: Percent

Dimensions for Amazon ECS Metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions:

Dimension	Description
`ClusterName`	This dimension filters the data you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName`.
`ServiceName`	This dimension filters the data you request for all resources in a specified service within a specified cluster.

Cluster Reservation

Cluster reservation metrics are measured as the percentage of CPU and memory that is reserved by all Amazon ECS tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active container instance in the cluster.

```
                                         (Total CPU units reserved by tasks in
 cluster) x 100
Cluster CPU reservation =
  -------------------------------------------------------------
                              (Total CPU units registered by container instances
 in cluster)
```

```
                                          (Total MiB of memory reserved by tasks
 in cluster x 100)
Cluster memory reservation =
  -----------------------------------------------------------------
                              (Total MiB of memory registered by container
 instances in cluster)
```

When you run a task in a cluster, Amazon ECS parses its task definition and reserves the aggregate CPU units and MiB of memory that is specified in its container definitions. Each minute, Amazon ECS calculates the number of CPU units and MiB of memory that are currently reserved for each task that is running in the cluster. The total amount of CPU and memory reserved for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster.

For example, a cluster has two active container instances registered, a `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If a task definition reserves 1,024 CPU units and 2,048 MiB of memory, and ten tasks are started with this task definition on this cluster (and no other tasks are currently running), a total of 10,240 CPU units and 20,480 MiB of memory are reserved, which is reported to CloudWatch as 55% CPU reservation and 60% memory reservation for the cluster.

The illustration below shows the total registered CPU units in a cluster and what their reservation and utilization means to existing tasks and new task placement. The lower (Reserved, utilized) and center (Reserved, not utilized) blocks represent the total CPU units that are reserved for the existing tasks that are running on the cluster, or the `CPUReservation` CloudWatch metric. The lower block represents the reserved CPU units that the running tasks are actually using on the cluster, or the `CPUUtilization` CloudWatch metric. The upper block represents CPU units that are not reserved by existing tasks; these CPU units are available for new task placement. Existing tasks can utilize these unreserved CPU units as well, if their need for CPU resources increases. For more information, see the `cpu` (p.   ) task definition parameter documentation.

Aggregate cluster CPU units
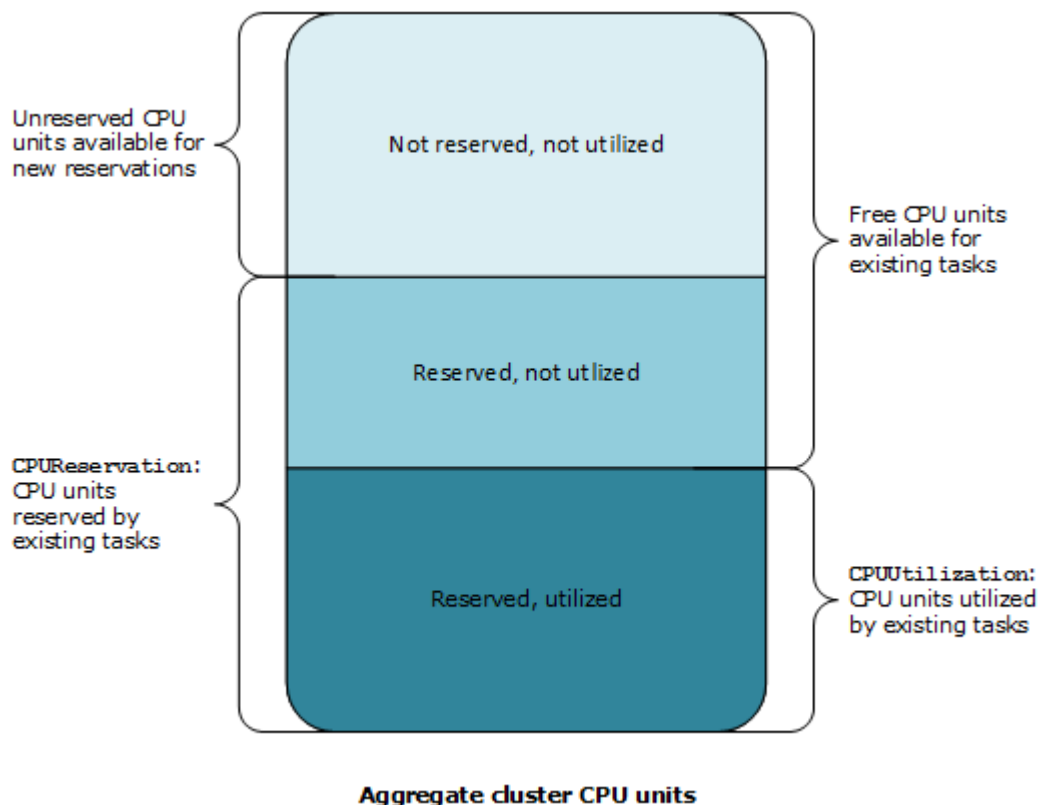
Cluster Utilization

Cluster utilization is measured as the percentage of CPU and memory that is used by all Amazon ECS tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active container instance in the cluster.

```
                                  (Total CPU units used by tasks in cluster)
 x 100
Cluster CPU utilization =
 --------------------------------------------------------------
                              (Total CPU units registered by container instances
 in cluster)
```

```
                                  (Total MiB of memory used by tasks in
 cluster x 100)
Cluster memory utilization =
 ----------------------------------------------------------------
                              (Total MiB of memory registered by container
 instances in cluster)
```

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster.

For example, a cluster has two active container instances registered, a `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If ten tasks are running on this cluster that each consume 1,024 CPU units and 2,048 MiB of memory, a total of 10,240 CPU units and 20,480 MiB of memory are utilized on the cluster, which is reported to CloudWatch as 55% CPU utilization and 60% memory utilization for the cluster.

Service Utilization

Service utilization is measured as the percentage of CPU and memory that is used by the Amazon ECS tasks that belong to a service on a cluster when compared to the CPU and memory that is defined in the service's task definition.

```
                                              (Total CPU units used by tasks in
 service) x 100
Service CPU utilization =
  ---------------------------------------------------------------------------
                               (Total CPU units reserved in task definition) x
 (number of tasks in service)
```

```
                                              (Total MiB of memory used by tasks
 in service) x 100
Service memory utilization =
  ------------------------------------------------------------------------------
                               (Total MiB of memory reserved in task
 definition) x (number of tasks in service)
```

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks owned by the service that are running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total resources that are reserved for the service in the service's task definition.

For example, the task definition for a service reserves a total of 512 CPU units and 1,024 MiB of memory for all of its containers. The service has a desired count of 1 running task, the service is running on a cluster with 1 `c4.large` container instance (with 2,048 CPU units and 3,768 MiB of memory), and there are no other tasks running on the cluster. Although the task has 512 CPU units reserved, because it is the only running task on a container instance with 2,048 CPU units, it has the ability to use up to four times the reserved amount (2,048 / 512); however, the memory reservation of 1,024 MiB is a hard limit and it cannot be exceeded, so service memory utilization cannot exceed 100%.

If this task is performing CPU-intensive work during a period and using all 2,048 of the available CPU units and 512 MiB of memory, then the service reports 400% CPU utilization and 50% memory utilization. If the task is idle and using 128 CPU units and 128 MiB of memory, then the service reports 25% CPU utilization and 12.5% memory utilization.

Service RUNNING Task Count

You can use CloudWatch metrics to view the number of tasks in your services that are in the `RUNNING` state. For example, you can set a CloudWatch alarm for this metric to alert you if the number of running tasks in your service falls below a specified value.

To view the number of running tasks in a service

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. Under the **Metrics** section of the navigation pane, choose **ECS**.

3. Scroll down to the **ECS > ClusterName,ServiceName** table and choose any metric (either `CPUUtilization` or `MemoryUtilization`) that corresponds to the service to view running tasks in.

4. Change the period to **1 Minute**.

5. Change the statistic to **Data Samples**. The value displayed indicates the number of `RUNNING` tasks in the service.
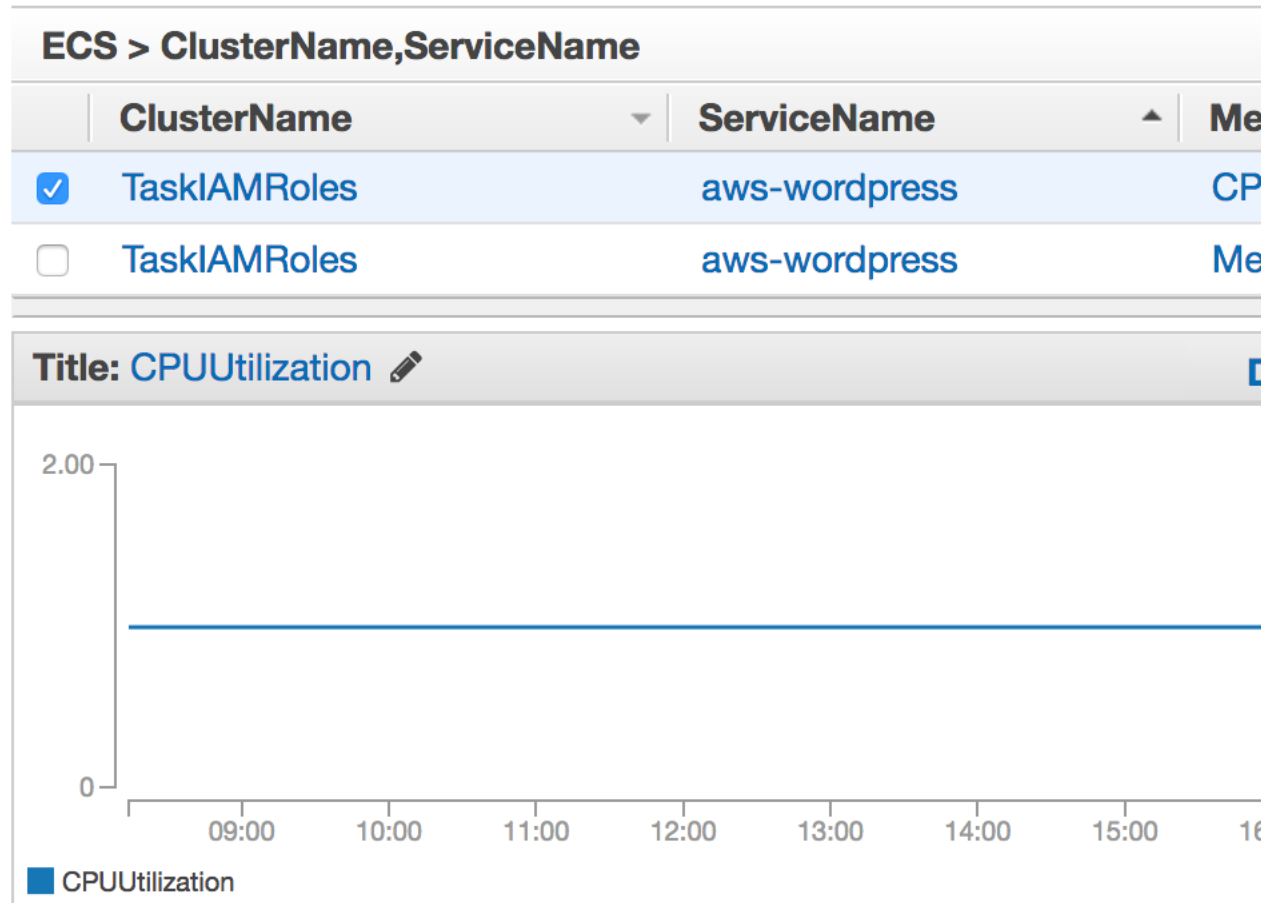
Viewing Amazon ECS Metrics

After you have enabled CloudWatch metrics for Amazon ECS, you can view those metrics in both the Amazon ECS and CloudWatch consoles. The Amazon ECS console provides a 24-hour maximum, minimum, and average view of your cluster and service metrics, while the CloudWatch console provides a fine-grained and customizable display of your resources, as well as the number of running tasks in a service.

Topics

Viewing Cluster Metrics in the Amazon ECS Console

Cluster and service metrics are available in the Amazon ECS console. The view provided for cluster metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information about cluster metrics, see Cluster Reservation (p. 155) and Cluster Utilization (p. 156).

To view cluster metrics in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster to view metrics with.
3. On the **Cluster:** *cluster-name* page, choose the **Metrics** tab to view cluster metrics.
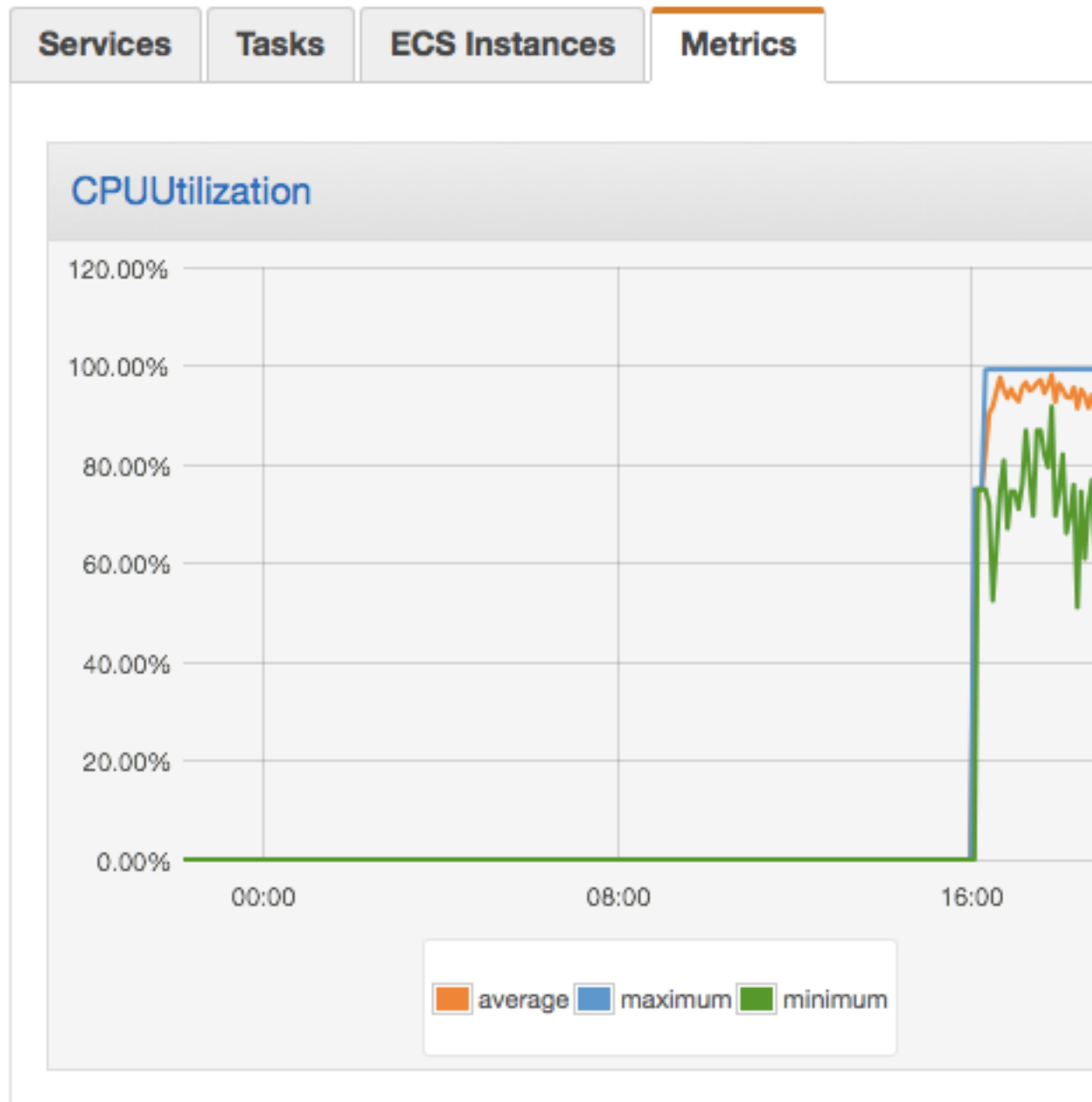
Viewing Service Metrics in the Amazon ECS Console

Service CPU and memory utilization metrics are available in the Amazon ECS console. The view provided for service metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information about service utilization metrics, see Service Utilization (p. 157).

To view service metrics in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. Choose the cluster that contains the service to view metrics with.

3. On the **Cluster:** *cluster-name* page, choose the **Services** tab to view the services in that cluster.

4. Choose the service to view metrics with.

5. On the **Service:** *service-name* page, choose the **Metrics** tab to view service metrics.
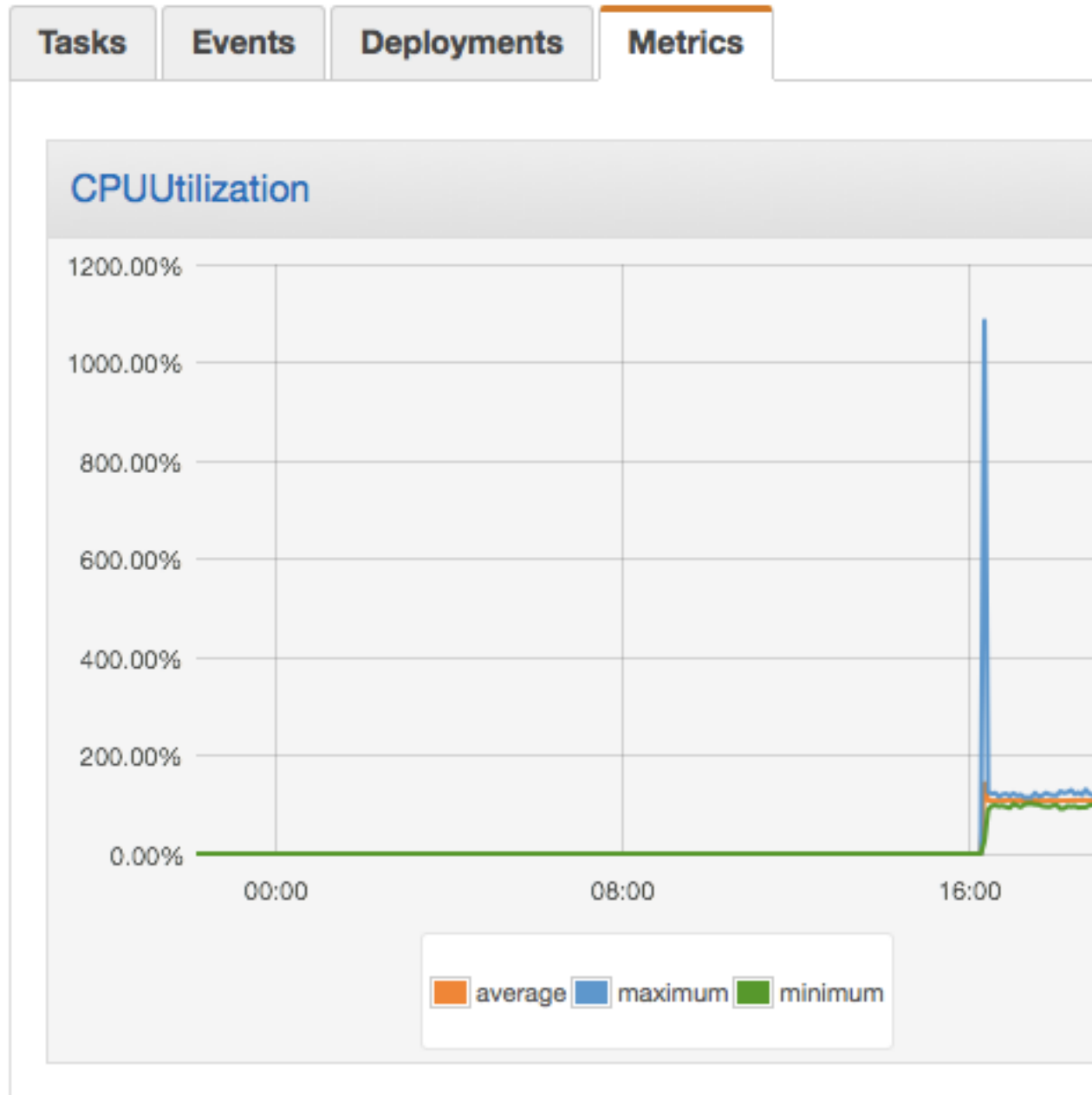
Viewing Amazon ECS Metrics in the CloudWatch Console

Amazon ECS cluster and service metrics can also be viewed in the CloudWatch console. The CloudWatch console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs. You can view Cluster Reservation (p. 155), Cluster Utilization (p. 156), Service Utilization (p. 157), and the Service RUNNING Task Count (p. 157). For more information about CloudWatch, see the Amazon CloudWatch User Guide.

To view metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the **Metrics** section in the left navigation, choose **ECS**.
3. Choose the metrics to view. Cluster metrics are scoped as **ECS > ClusterName** and service utilization metrics are scoped as **ECS > ClusterName, ServiceName**. The example below shows cluster CPU and memory utilization.
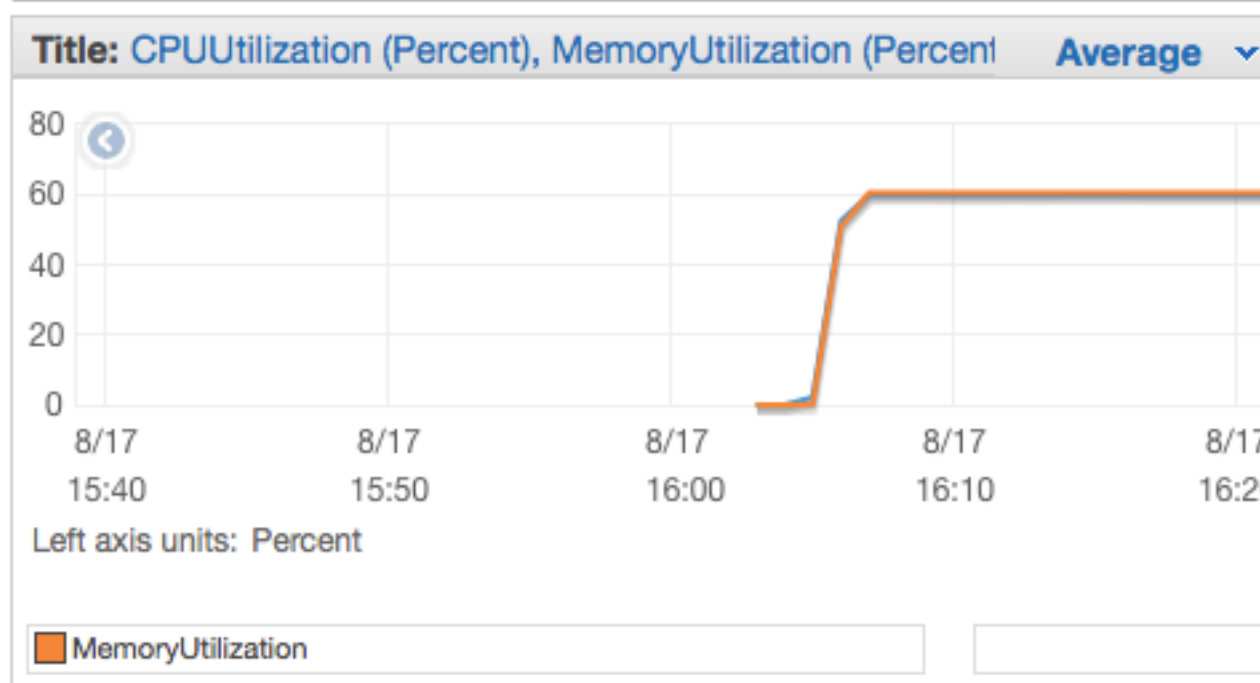
Tutorial: Scaling Container Instances with CloudWatch Alarms

The following procedures help you to create an Auto Scaling group for an Amazon ECS cluster that you can scale up (and down) using CloudWatch alarms.

Depending on the Amazon EC2 instance types you use in your clusters, and quantity of container instances you have in a cluster, your tasks have a limited amount of resources that they can use when they are run. ECS monitors the resources available in the cluster to work with the schedulers to place tasks. If your cluster runs low on any of these resources, such as memory, you will eventually be unable to launch more tasks until you add more container instances, reduce the number of desired tasks in a service, or stop some of the running tasks in your cluster to free up the constrained resource.

In this tutorial, you create a CloudWatch alarm using the `MemoryReservation` metric for your cluster. When the memory reservation of your cluster rises above 75% (meaning that only 25% of the memory in your cluster is available to for new tasks to reserve), the alarm triggers the Auto Scaling group to add another instance and provide more resources for your tasks and services.

Prerequisites

This tutorial assumes that you have enabled CloudWatch metrics for your clusters and services. Metrics are not available until the clusters and services send the metrics to CloudWatch, and you cannot create CloudWatch alarms for metrics that do not exist yet.

Your Amazon ECS container instances require at least version 1.4.0 of the container agent to enable CloudWatch metrics. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 63).

Your Amazon ECS container instances also require `ecs:StartTelemetrySession` permission on the IAM role that you launch your container instances with. If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, then you might need to add this permission. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see To check for the `ecsInstanceRole` in the IAM console (p. 181).

Step 1: Create a CloudWatch Alarm for a Metric

After you have enabled CloudWatch metrics for your clusters and services, and the metrics for your cluster are visible in the CloudWatch console, you can set alarms on the metrics. For more information, see Creating Amazon CloudWatch Alarms in the *Amazon CloudWatch User Guide*.

For this tutorial, you create an alarm on the cluster `MemoryReservation` metric to alert when the cluster's memory reservation is above 75%.

To create a CloudWatch alarm on a metric

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the left navigation, choose **Alarms**.
3. Choose **Create Alarm**.
4. In the **CloudWatch Metrics by Category** section, choose **ECS > ClusterName**.
5. On the **Modify Alarm** page, choose the `MemoryReservation` metric for the default cluster and choose **Next**.
6. In the **Alarm Threshold** section, enter a name and description for your alarm.

    *   **Name:** `memory-above-75-pct`
    *   **Description:** `Cluster memory reservation above 75%`

7. Set the threshold and time period requirement to `MemoryReservation` greater than 75% for 1 period.

Alarm Threshold

Provide the details and threshold for your alarm. Use the graph on the right to help set the appropriate threshold.

Name: memory-above-75-pct

Description: Cluster memory reservation above 75%

Whenever: MemoryReservation

is: > 75

for: 1 consecutive period(s)

8. (Optional) Configure a notification to send when the alarm is triggered. You can also choose to delete the notification if you don't want to configure one now.

9. Choose **Create Alarm**. Now you can use this alarm to trigger your Auto Scaling group to add a container instance when the memory reservation is above 75%.

10. (Optional) You can also create another alarm that triggers when the memory reservation is below 25%, which you can use to remove a container instance from your Auto Scaling group.

Step 2: Create a Launch Configuration for an Auto Scaling Group

Now that you have enabled CloudWatch metrics and created an alarm based on one of those metrics, you can create a launch configuration and an Auto Scaling group for your cluster. For more information and other configuration options, see the Auto Scaling User Guide.

To create an Auto Scaling launch configuration

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. On the left navigation, choose **Auto Scaling Groups**.

3. On the **Welcome to Auto Scaling** page, choose **Create Auto Scaling Group**.

4. On the **Create Auto Scaling Group** page, choose **Create launch configuration**.

5. On the **Choose AMI** step of the **Create Auto Scaling Group** wizard, choose **Community AMIs**.

6. Choose the ECS-optimized AMI for your Auto Scaling group.

   To use the Amazon ECS-optimized AMI, type **amazon-ecs-optimized** in the **Search community AMIs** field and press the **Enter** key. Choose **Select** next to the **amzn-ami-2016.09.b-amazon-ecs-optimized** AMI. The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI Name	AMI ID	EC2 console link
us-east-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-eca289fb	Launch instance
us-east-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-446f3521	Launch instance

Region	AMI Name	AMI ID	EC2 console link
us-west-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-9fadf8ff	Launch instance
us-west-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-7abc111a	Launch instance
eu-west-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-a1491ad2	Launch instance
eu-central-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-54f5303b	Launch instance
ap-northeast-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-9cd57ffd	Launch instance
ap-southeast-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-a900a3ca	Launch instance
ap-southeast-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-5781be34	Launch instance

7. On the **Choose Instance Type** step of the **Create Auto Scaling Group** wizard, choose an instance type for your Auto Scaling group and choose **Next: Configure details**.

8. On the **Configure details** step of the **Create Auto Scaling Group** wizard, enter the following information. The other fields are optional. For more information, see Creating Launch Configurations in the *Auto Scaling User Guide*.

   • **Name:** Enter a name for your launch configuration.

   • **IAM role:** Select the ecsInstanceRole for your container instances. If you do not have this role configured, see Amazon ECS Container Instance IAM Role (p. 180).

   • **IP Address Type:** Choose the IP address type option that you want for your container instances. If you want external traffic to be able to reach your containers, choose **Assign a public IP address to every instance.**

9. (Optional) If you have configuration information that you want to pass to your container instances with EC2 user data, choose **Advanced Details** and enter your user data in the **User data** field. For more information, see Amazon ECS Container Agent Configuration (p. 69).

10. Choose **Next: Add Storage**.

11. On the **Add Storage** step of the **Create Auto Scaling Group** wizard, make any storage configuration changes you need for your instances and choose **Next: Configure Security Group**.

12. On the **Configure Security Group** step of the **Create Auto Scaling Group** wizard, select an existing security group that meets the needs of your containers, or create a new security group and choose **Review**.

13. Review your launch configuration and choose **Create launch configuration**.

14. Select a private key to use for connecting to your instances with SSH and choose **Create launch configuration** to finish and move on to creating an Auto Scaling group with your new launch configuration.

Step 3: Create an Auto Scaling Group for your Cluster

After the launch configuration is complete, continue with the following procedure to create an Auto Scaling group that uses your launch configuration.

To create an Auto Scaling group

1. On the **Configure Auto Scaling group details** step of the **Create Auto Scaling Group** wizard, enter the following information and choose **Next: Configure scaling policies**.

    *   **Group name:** Enter a name for your Auto Scaling group.
    *   **Group size:** Specify the number of container instances your Auto Scaling group should start with.
    *   **Network:** Choose a VPC to launch your container instances into.
    *   **Subnet:** Choose the subnets you would like to launch your container instances into. For a highly available cluster, we recommend that you enable all of the subnets in the region.

2. On the **Configure scaling policies** step of the **Create Auto Scaling Group** wizard, choose **Use scaling policies to adjust the capacity of this group**.

3. Enter the minimum and maximum number of container instances for your Auto Scaling group.

4. In the **Increase Group Size** section, enter the following information.

    *   **Execute policy when:** Choose the `memory-above-75-pct` CloudWatch alarm you configured earlier.
    *   **Take the action:** Enter the number of instances you would like to add to your cluster when the alarm is triggered.

5. If you configured an alarm to trigger a group size reduction, set that alarm in the **Decrease Group Size** section and specify how many instances to remove if that alarm is triggered. Otherwise, collapse the **Decrease Group Size** section by clicking the **X** in the upper-right-hand corner of the section.

    Note
    If you configure your Auto Scaling group to remove container instances, any tasks running on the removed container instances are killed. If your tasks are running as part of a service, Amazon ECS restarts those tasks on another instance if the required resources are available (CPU, memory, ports); however, tasks that were started manually will are not restarted automatically.

6. Choose **Review** to review your Auto Scaling group and then choose **Create Auto Scaling Group** to finish.

Step 4: Verify and Test your Auto Scaling Group

Now that you've created your Auto Scaling group, you should be able to see your instances launching in the Amazon EC2 console **Instances** page. These instances should register into your Amazon ECS cluster as well after they launch.

To test that your Auto Scaling group is configured properly, you can create some tasks that consume a considerable amount of memory and start launching them into your cluster. After your cluster exceeds the 75% memory reservation from the CloudWatch alarm for the specified number of periods, you should see a new instance launch in the EC2 console.

Step 5: Cleaning Up

When you have completed this tutorial, you may choose to keep your Auto Scaling group and Amazon EC2 instances in service for your cluster. However, if you are not actively using these resources, you should consider cleaning them up so your account does not incur unnecessary charges. You can delete your Auto Scaling group to terminate the Amazon EC2 instances within it, but your launch configuration remains intact and you can create a new Auto Scaling group with it later if you choose.

To delete your Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the left navigation, choose **Auto Scaling Groups**.
3. Choose the Auto Scaling group you created for this tutorial.
4. Choose **Actions** and then choose **Delete**.
5. Choose **Yes, Delete** to delete your Auto Scaling group.

Amazon ECS IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify Amazon ECS resources, or perform tasks using the Amazon ECS API. (This means that they also can't do so using the Amazon ECS console or the AWS CLI.) To allow IAM users to create or modify resources and perform tasks, you must create IAM policies that grant IAM users permission to use the specific resources and API actions they'll need, and then attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more general information about IAM policies, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Likewise, Amazon ECS container instances make calls to the Amazon ECS and Amazon EC2 APIs on your behalf, so they need to authenticate with your credentials. This authentication is accomplished by creating an IAM role for your container instances and associating that role with your container instances when you launch them. For more information, see Amazon ECS Container Instance IAM Role (p. 180). If you use an Elastic Load Balancing load balancer with your Amazon ECS services, calls to the Amazon EC2 and Elastic Load Balancing APIs are made on your behalf to register and deregister container instances with your load balancers. For more information, see Amazon ECS Service Scheduler IAM Role (p. 182). For more general information about IAM roles, see IAM Roles in the *IAM User Guide*.

Getting Started

An IAM policy must grant or deny permission to use one or more Amazon ECS actions. It must also specify the resources that can be used with the action, which can be all resources, or in some cases, specific resources. The policy can also include conditions that you apply to the resource.

Amazon ECS partially supports resource-level permissions. This means that for some Amazon ECS API actions, you cannot specify which resource a user is allowed to work with for that action; instead, you have to allow users to work with all resources for that action.

Topics

Policy Structure

The following topics explain the structure of an IAM policy.

Topics

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

There are various elements that make up a statement:

- **Effect:** The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.

- **Action**: The *action* is the specific API action for which you are granting or denying permission. To learn about specifying *action*, see Actions for Amazon ECS (p. 170).

- **Resource**: The resource that's affected by the action. Some Amazon ECS API actions allow you to include specific resources in your policy that can be created or modified by the action. To specify a resource in the statement, you need to use its Amazon Resource Name (ARN). For more information about specifying the *arn* value, see Amazon Resource Names for Amazon ECS (p. 170). For more information about which API actions support which ARNs, see Supported Resource-Level

Permissions for Amazon ECS API Actions (p. 173). If the API action does not support ARNs, use the * wildcard to specify that all resources can be affected by the action.

- **Condition**: Conditions are optional. They can be used to control when your policy will be in effect. For more information about specifying conditions for Amazon ECS, see Condition Keys for Amazon ECS (p. 171).

For more information about example IAM policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 174).

Actions for Amazon ECS

In an IAM policy statement, you can specify any API action from any service that supports IAM. For Amazon ECS, use the following prefix with the name of the API action: `ecs:`. For example: `ecs:RunTask` and `ecs:CreateCluster`.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["ecs:action1", "ecs:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Describe" as follows:

```
"Action": "ecs:Describe*"
```

To specify all Amazon ECS API actions, use the * wildcard as follows:

```
"Action": "ecs:*"
```

For a list of Amazon ECS actions, see Actions in the *Amazon EC2 Container Service API Reference*.

Amazon Resource Names for Amazon ECS

Each IAM policy statement applies to the resources that you specify using their ARNs.

> **Important**
> Currently, not all API actions support individual ARNs; we'll add support for additional API actions and ARNs for additional Amazon ECS resources later. For information about which ARNs you can use with which Amazon ECS API actions, as well as supported condition keys for each ARN, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 173).

An ARN has the following general syntax:

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

service
    The service (for example, `ecs`).
region
    The region for the resource (for example, `us-east-1`).
account
    The AWS account ID, with no hyphens (for example, `123456789012`).
resourceType
    The type of resource (for example, `instance`).

resourcePath
> A path that identifies the resource. You can use the * wildcard in your paths.

For example, you can indicate a specific cluster (`default`) in your statement using its ARN as follows:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/default"
```

You can also specify all clusters that belong to a specific account by using the * wildcard as follows:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

To specify all resources, or if a specific API action does not support ARNs, use the * wildcard in the `Resource` element as follows:

```
"Resource": "*"
```

The following table describes the ARNs for each type of resource used by the Amazon ECS API actions.

Resource Type	ARN
All Amazon ECS resources	arn:aws:ecs:*
All Amazon ECS resources owned by the specified account in the specified region	arn:aws:ecs:*region*:*account*:*
Cluster	arn:aws:ecs:*region*:*account*:cluster/*cluster-name*
Container instance	arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*
Task definition	arn:aws:ecs:*region*:*account*:task-definition-family-name:*task-definition-revision-number*
Service	arn:aws:ecs:*region*:*account*:service/*service-name*
Task	arn:aws:ecs:*region*:*account*:task/*task-id*
Container	arn:aws:ecs:*region*:*account*:container/*container-id*

Many Amazon ECS API actions accept multiple resources. To specify multiple resources in a single statement, separate their ARNs with commas, as follows:

```
"Resource": ["arn1", "arn2"]
```

For more general information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the *Amazon Web Services General Reference*.

Condition Keys for Amazon ECS

In a policy statement, you can optionally specify conditions that control when it is in effect. Each condition contains one or more key-value pairs. Condition keys are not case-sensitive. We've defined AWS-wide condition keys, plus additional service-specific condition keys.

If you specify multiple conditions, or multiple keys in a single condition, we evaluate them using a logical AND operation. If you specify a single condition with multiple values for one key, we evaluate the condition using a logical OR operation. For permission to be granted, all conditions must be met.

You can also use placeholders when you specify conditions. For more information, see Policy Variables in the *IAM User Guide*.

Amazon ECS implements the AWS-wide condition keys (see Available Keys), plus the following service-specific condition keys. (We'll add support for additional service-specific condition keys for Amazon ECS later.)

Condition Key	Key/Value Pair	Evaluation Types
ecs:cluster	"ecs:cluster":"*cluster-arn*" Where *cluster-arn* is the ARN for the Amazon ECS cluster	ARN, Null
ecs:container-instances	"ecs:container-instances":"*container-instance-arns*" Where *container-instance-arns* is one or more container instance ARNs.	ARN, Null

For information about which condition keys you can use with which Amazon ECS resources, on an action-by-action basis, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 173). For example policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 174).

Checking that Users Have the Required Permissions

After you've created an IAM policy, we recommend that you check whether it grants users the permissions to use the particular API actions and resources they need before you put the policy into production.

First, create an IAM user for testing purposes, and then attach the IAM policy that you created to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

> **Note**
> You can also test your policies with the IAM Policy Simulator. For more information on the policy simulator, see Working with the IAM Policy Simulator in the *IAM User Guide*.

If the action that you are testing creates or modifies a resource, you should make the request using the `DryRun` parameter (or run the AWS CLI command with the `--dry-run` option). In this case, the call completes the authorization check, but does not complete the operation. For example, you can check whether the user can terminate a particular instance without actually terminating it. If the test user has the required permissions, the request returns `DryRunOperation`; otherwise, it returns `UnauthorizedOperation`.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed and retest until you get the desired results.

> **Important**
> It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the `DecodeAuthorizationMessage` action. For more

information, see DecodeAuthorizationMessage in the *AWS Security Token Service API Reference*, and decode-authorization-message in the *AWS Command Line Interface Reference*.

Supported Resource-Level Permissions for Amazon ECS API Actions

Resource-level permissions refers to the ability to specify which resources users are allowed to perform actions on. Amazon ECS has partial support for resource-level permissions. This means that for certain Amazon ECS actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use. For example, you can grant users permission to launch instances, but only of a specific type, and only using a specific AMI.

The following table describes the Amazon ECS API actions that currently support resource-level permissions, as well as the supported resources, resource ARNs, and condition keys for each action.

Important
If an Amazon ECS API action is not listed in this table, then it does not support resource-level permissions. If an Amazon ECS API action does not support resource-level permissions, you can grant users permission to use the action, but you have to specify a * for the resource element of your policy statement.

API action	Resource	Condition keys
DeleteCluster	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
DeregisterContainerInstance	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
DescribeClusters	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster1*, arn:aws:ecs:*region*:*account*:cluster/*my-cluster2*	N/A
DescribeContainerInstances	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id1*, arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id2*	ecs:cluster
DescribeTasks	Task arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252a*, arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252b*	ecs:cluster
ListContainerInstances	Cluster	N/A

API action	Resource	Condition keys
	arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	
ListTasks	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
Poll	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
RegisterContainerInstance	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
RunTask	Task definition arn:aws:ecs:*region*:*account*:task-definition/*hello_world:8*	ecs:cluster
StartTask	Task definition arn:aws:ecs:*region*:*account*:task-definition/*hello_world:8*	ecs:cluster ecs:container-instances
StartTelemetrySession	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
StopTask	Task arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252a*	ecs:cluster
SubmitContainerStateChange	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
SubmitTaskStateChange	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
UpdateContainerAgent	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster

Creating Amazon ECS IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have
access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more general information about IAM policies, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

To create an IAM policy for a user

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies** and then choose **Create Policy**.

3. In the **Create Policy** section, choose **Select** next to **Create Your Own Policy**.

4. In the **Policy Name** field, type your own unique name, such as `AmazonECSUserPolicy`.

5. In the **Policy Document** field, paste the policy to apply to the user. For example policies, see Amazon ECS IAM Policy Examples (p. 190).

6. Choose **Create Policy** to finish.

To attach an IAM policy to a user

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Users** and then choose the user you would like to attach the policy to.

3. In the **Permissions** tab, choose **Attach Policy**.

4. In the **Attach Policy** section, select the custom policy you created in the previous procedure and then choose **Attach Policy**.

Managed Policies

Amazon ECS and Amazon ECR provide several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own polices.

Topics

- Amazon ECS Managed Policies (p. 175)
- Amazon ECR Managed Policies (p. 178)

Amazon ECS Managed Policies

Amazon ECS provides several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over Amazon ECS resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own polices. For more information about each API operation mentioned in these policies, see Actions in the *Amazon EC2 Container Service API Reference*.

Topics

- AmazonEC2ContainerServiceFullAccess (p. 176)
- AmazonEC2ContainerServiceforEC2Role (p. 176)
- AmazonEC2ContainerServiceRole (p. 177)
- AmazonEC2ContainerServiceAutoscaleRole (p. 177)

- AmazonEC2ContainerServiceTaskRole (p. 178)

AmazonEC2ContainerServiceFullAccess

This policy allows full administrator access to Amazon ECS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:Describe*",
        "autoscaling:UpdateAutoScalingGroup",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStack*",
        "cloudformation:UpdateStack",
        "cloudwatch:GetMetricStatistics",
        "ec2:Describe*",
        "elasticloadbalancing:*",
        "ecs:*",
        "iam:ListInstanceProfiles",
        "iam:ListRoles",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

AmazonEC2ContainerServiceforEC2Role

This policy allows Amazon ECS container instances to make calls to AWS on your behalf. For more information, see Amazon ECS Container Instance IAM Role (p. 180).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
```

```
        ],
        "Resource": "*"
      }
    ]
}
```

AmazonEC2ContainerServiceRole

This policy allows Elastic Load Balancing load balancers to register and deregister Amazon ECS container instances on your behalf. For more information, see Amazon ECS Service Scheduler IAM Role (p. 182).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:Describe*",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets"
      ],
      "Resource": "*"
    }
  ]
}
```

AmazonEC2ContainerServiceAutoscaleRole

This policy allows Application Auto Scaling to scale your Amazon ECS service's desired count up and down in response to CloudWatch alarms on your behalf. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 184).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1456535218000",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Stmt1456535243000",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
```

```
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AmazonEC2ContainerServiceTaskRole

This policy allows containers in your Amazon ECS tasks to make calls to the AWS APIs on your behalf. For more information, see **Amazon EC2 Container Service Task Role** (p. 186).

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

Amazon ECR Managed Policies

Amazon ECR provides several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over Amazon ECR resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own polices. For more information about each API operation mentioned in these policies, see Actions in the *Amazon EC2 Container Registry API Reference*.

Topics

AmazonEC2ContainerRegistryFullAccess

This policy allows full administrator access to Amazon ECR.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:*"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

AmazonEC2ContainerRegistryPowerUser

This policy allows power user access to Amazon ECR, which allows read and write access to repositories, but does not allow users to delete repositories or change the policy documents applied to them.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetRepositoryPolicy",
        "ecr:DescribeRepositories",
        "ecr:ListImages",
        "ecr:BatchGetImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload",
        "ecr:PutImage"
      ],
      "Resource": "*"
    }
  ]
}
```

AmazonEC2ContainerRegistryReadOnly

This policy allows read-only access to Amazon ECR, such as the ability to list repositories and the images within the repositories, and also to pull images from Amazon ECR with the Docker CLI.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetRepositoryPolicy",
        "ecr:DescribeRepositories",
        "ecr:ListImages",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon ECS Container Instance IAM Role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you. Before you can launch container instances and register them into a cluster, you must create an IAM role for those container instances to use when they are launched. This requirement applies to container instances launched with the Amazon ECS-optimized AMI provided by Amazon, or with any other instances that you intend to run the agent on.

> **Important**
> Containers that are running on your container instances have access to all of the permissions that are supplied to the container instance role through instance metadata. We recommend that you limit the permissions in your container instance role to the minimal list of permissions provided in the managed `AmazonEC2ContainerServiceforEC2Role` policy shown below. If the containers in your tasks need extra permissions that are not listed here, we recommend providing those tasks with their own IAM roles. For more information, see IAM Roles for Tasks (p. 186).
> You can prevent containers on the `docker0` bridge from accessing the permissions supplied to the container instance role (while still allowing the permissions that are provided by IAM Roles for Tasks (p. 186)) by running the following **iptables** command on your container instances; however, containers will not be able to query instance metadata with this rule in effect. Note that this command assumes the default Docker bridge configuration and it will not work for containers that use the `host` network mode. For more information, see Network Mode (p. 89).

```
iptables --insert FORWARD 1 --in-interface docker+ --destination
 169.254.169.254/32 --jump DROP
```

The `AmazonEC2ContainerServiceforEC2Role` policy is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Note
The `ecs:CreateCluster` line in the above policy is optional, provided that the cluster you intend to register your container instance into already exists. If the cluster does not already exist, the agent must have permission to create it, or you can create the cluster with the **create-cluster** command prior to launching your container instance.
If you omit the `ecs:CreateCluster` line, the Amazon ECS container agent will not be able to create clusters, including the default cluster.

The `ecs:Poll` line in the above policy is used to grant the agent permission to connect with the Amazon ECS service to report status and get commands.

The Amazon ECS instance role is automatically created for you in the console first-run experience; however, you should manually attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. You can use the following procedure to check and see if your account already has the Amazon ECS instance role and to attach the managed IAM policy if needed.

To check for the `ecsInstanceRole` in the IAM console

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Search the list of roles for `ecsInstanceRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose the **Permissions** tab.

5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceforEC2Role** managed policy is attached to the role. If the policy is attached, your Amazon ECS instance role is properly configured. If not, follow the substeps below to attach the policy.

    a. Choose **Attach Policy**.

    b. In the **Filter** box, type **AmazonEC2ContainerServiceforEC2Role** to narrow the available policies to attach.

    c. Check the box to the left of the **AmazonEC2ContainerServiceforEC2Role** policy and choose **Attach Policy**.

6. Choose the **Trust Relationships** tab, and **Edit Trust Relationship**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create the `ecsInstanceRole` IAM role for your container instances

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create New Role**.
3. In the **Role Name** field, type `ecsInstanceRole` to name the role, and then choose **Next Step**.
4. In the **Select Role Type** section, choose **Select** next to the **Amazon EC2 Role for EC2 Container Service** role.
5. In the **Attach Policy** section, select the **AmazonEC2ContainerServiceforEC2Role** policy and then choose **Next Step**.
6. Review your role information and then choose **Create Role** to finish.

Adding Amazon S3 Read-only Access to your Container Instance Role

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch time. You can store a copy of your `ecs.config` file in a private bucket, use Amazon EC2 user data to install the AWS CLI and then copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

For more information about creating an `ecs.config` file, storing it in Amazon S3, and launching instances with this configuration, see Storing Container Instance Configuration in Amazon S3 (p. 74).

To allow Amazon S3 read-only access for your container instance role

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose the IAM role you use for your container instances (this role is likely titled `ecsInstanceRole`). For more information, see Amazon ECS Container Instance IAM Role (p. 180).
4. Choose the **Permissions** tab.
5. Under **Managed Policies**, choose **Attach Policy**.
6. On the **Attach Policy** page, type `S3` into the **Filter** field to narrow the policy results.
7. Check the box to the left of the **AmazonS3ReadOnlyAccess** policy and click **Attach Policy**.

   Note
   This policy allows read-only access to all Amazon S3 resources. For more restrictive bucket policy examples, see Bucket Policy Examples in the Amazon Simple Storage Service Developer Guide.

Amazon ECS Service Scheduler IAM Role

The Amazon ECS service scheduler makes calls to the Amazon EC2 and Elastic Load Balancing APIs on your behalf to register and deregister container instances with your load balancers. Before you can attach a load balancer to an Amazon ECS service, you must create an IAM role for your services to use before you start them. This requirement applies to any Amazon ECS service that you plan to use with a load balancer.

In most cases, the Amazon ECS service role is created for you automatically in the console first-run experience. You can use the following procedure to check if your account already has the Amazon ECS service role.

The `AmazonEC2ContainerServiceRole` policy is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:Describe*",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

The `ec2:AuthorizeSecurityGroupIngress` rule is reserved for future use. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To check for the `ecsServiceRole` in the IAM console

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Search the list of roles for `ecsServiceRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose the **Permissions** tab.

5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.

    a. Choose **Attach Policy**.

    b. In the **Filter** box, type **AmazonEC2ContainerServiceRole** to narrow the available policies to attach.

    c. Check the box to the left of the **AmazonEC2ContainerServiceRole** policy and choose **Attach Policy**.

6. Choose the **Trust Relationships** tab, and **Edit Trust Relationship**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
```

```
            "Service": "ecs.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
  ]
}
```

To create an IAM role for your service scheduler load balancers

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles** and then choose **Create New Role**.

3. In the **Role Name** field, type `ecsServiceRole` to name the role, and then choose **Next Step**.

4. In the **Select Role Type** section, scroll down and choose **Select** next to the **Amazon EC2 Container Service Role** service role.

5. In the **Attach Policy** section, select the **AmazonEC2ContainerServiceRole** policy and then choose **Next Step**.

6. Review your role information and then choose **Create Role** to finish.

Amazon ECS Service Auto Scaling IAM Role

Before you can use Service Auto Scaling with Amazon ECS, the Application Auto Scaling service needs permission to describe your CloudWatch alarms and registered services, as well as permission to update your Amazon ECS service's desired count on your behalf. These permissions are provided by the Service Auto Scaling IAM role (`ecsAutoscaleRole`).

> **Note**
> IAM users also require permissions to use Service Auto Scaling; these permissions are described in Service Auto Scaling Required IAM Permissions (p. 134). If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p.   ) or update service (p. 147) workflows, and then use the role for any other service later (in the console or with the CLI/SDKs).

You can use the following procedure to check and see if your account already has Service Auto Scaling.

The `AmazonEC2ContainerServiceAutoscaleRole` policy is shown below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1456535218000",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
```

```
        },
        {
            "Sid": "Stmt1456535243000",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

To check for the Service Auto Scaling role in the IAM console

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Search the list of roles for `ecsAutoscaleRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose the **Permissions** tab.

5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceAutoscaleRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.

   a. Choose **Attach Policy**.

   b. For **Filter**, type `AmazonEC2ContainerServiceAutoscaleRole` to narrow the available policies to attach.

   c. Select the box to the left of the **AmazonEC2ContainerAutoscaleRole** policy and choose **Attach Policy**.

6. Choose **Trust Relationships**, **Edit Trust Relationship**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "application-autoscaling.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create an IAM role for Service Auto Scaling

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles** and then choose **Create New Role**.

3. In the **Role Name** field, type `ecsAutoscaleRole` to name the role, and then choose **Next Step**.
4. In the **Select Role Type** section, scroll down and choose **Select** next to the **Amazon EC2 Container Service Autoscale Role** service role.
5. In the **Attach Policy** section, select the **AmazonEC2ContainerServiceAutoscaleRole** policy and then choose **Next Step**.
6. Review your role information and then choose **Create Role** to finish.

Amazon EC2 Container Service Task Role

Before you can use IAM roles for tasks , Amazon ECS needs permission to make calls to the AWS APIs on your behalf. These permissions are provided by the Amazon EC2 Container Service Task Role.

You can create a task IAM role for each task definition that needs permission to call AWS APIs. You simply create an IAM policy that defines which permissions your task should have, and then attach that policy to a role that uses the Amazon EC2 Container Service Task Role managed policy. For more information, see Creating an IAM Role and Policy for your Tasks (p. 188).

The Amazon EC2 Container Service Task Role trust relationship is shown below.

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

IAM Roles for Tasks

With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or `RunTask` API operation. The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

> **Important**
> Containers that are running on your container instances have access to all of the permissions that are supplied to the container instance role. We recommend that you limit the permissions in your container instance role to the minimal list of permissions shown in Amazon ECS Container Instance IAM Role (p. 180).
> You can prevent containers on the `docker0` bridge from accessing the permissions supplied to the container instance role (while still allowing the permissions that are provided by IAM roles for tasks) by running the following **iptables** command on your container instances; however, containers will not be able to query instance metadata with this rule in effect. Note

that this command assumes the default Docker bridge configuration and it will not work for containers that use the `host` network mode. For more information, see Network Mode (p. 89).

```
iptables --insert FORWARD 1 --in-interface docker+ --destination
 169.254.169.254/32 --jump DROP
```

You define the IAM role to use in your task definitions, or you can use a `taskRoleArn` override when running a task manually with the `RunTask` API operation. The Amazon ECS agent receives a payload message for starting the task with additional fields that contain the role credentials. The Amazon ECS agent sets the task's UUID as an identification token and updates its internal credential cache so that the identification token for the task points to the role credentials that are received in the payload. The Amazon ECS agent populates the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable in the `Env` object (available with the **docker inspect *container_id*** command) for all containers that belong to this task with the following relative URI: */credential_provider_version*/credentials?id=*task_UUID*.

From inside the container, you can query the credentials with the following command:

```
$ curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
{
    "AccessKeyId": "ACCESS_KEY_ID",
    "Expiration": "EXPIRATION_DATE",
    "RoleArn": "TASK_ROLE_ARN",
    "SecretAccessKey": "SECRET_ACCESS_KEY",
    "Token": "SECURITY_TOKEN_STRING"
}
```

If your container instance is using at least version 1.11.0 of the container agent and a supported version of the AWS CLI or SDKs, then the SDK client will see that the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` variable is available, and it will use the provided credentials to make calls to the AWS APIs. For more information, see Enabling Task IAM Roles on your Container Instances (p. 188) and Using a Supported AWS SDK (p. 189).

Each time the credential provider is used, the request is logged locally on the host container instance at `/var/log/ecs/audit.log.YYYY-MM-DD-HH`. For more information, see IAM Roles for Tasks Credential Audit Log (p. 242).

Topics

Benefits of Using IAM Roles for Tasks

- **Credential Isolation:** A container can only retrieve credentials for the IAM role that is defined in the task definition to which it belongs; a container never has access to credentials that are intended for another container that belongs to another task.

- **Authorization:** Unauthorized containers cannot access IAM role credentials defined for other tasks.

- **Auditability:** Access and event logging is available through CloudTrail to ensure retrospective auditing. Task credentials have a context of `taskArn` that is attached to the session, so CloudTrail logs show which task is using which role.

Enabling Task IAM Roles on your Container Instances

Your Amazon ECS container instances require at least version 1.11.0 of the container agent to enable task IAM roles; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 63). If you are using the Amazon ECS-optimized AMI, your instance needs at least 1.11.0-1 of the `ecs-init` package. If your container instances are launched from version 2016.03.e or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see Amazon ECS-optimized AMI (p. 34).

If you are not using the Amazon ECS-optimized AMI for your container instances, be sure to add the `ECS_ENABLE_TASK_IAM_ROLE=true` environment variable and the `--net=host` option to your **docker run** command that starts the agent. For an example run command, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs) (p. 67). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

```
sysctl -w net.ipv4.conf.all.route_localnet=1
iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --
to-destination 127.0.0.1:51679
iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j
 REDIRECT --to-ports 51679
```

Creating an IAM Role and Policy for your Tasks

You must create an IAM policy for your tasks to use that specifies the permissions that you would like the containers in your tasks to have. You have several ways to create a new IAM permission policy. You can copy a complete AWS managed policy that already does some of what you're looking for and then customize it to your specific requirements. For more information, see Creating a New Policy in the *IAM User Guide*.

You must also create a role for your tasks to use before you can specify it in your task definitions. You can create the role using the **Amazon EC2 Container Service Task Role** managed role policy. Then you can attach your specific IAM policy to the role that gives the containers in your task the permissions you desire. The procedures below describe how to do this.

If you have multiple task definitions or services that require IAM permissions, you should consider creating a role for each specific task definition or service with the minimum required permissions for the tasks to operate so that you can minimize the access that you provide for each task.

To create an IAM policy for your tasks

In this example, we create a policy to allow read-only access to an Amazon S3 bucket. You could store database credentials or other secrets in this bucket, and the containers in your task can read the credentials from the bucket and load them into your application.

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies** and then choose **Create Policy**.

3. In the **Create Policy** section, choose **Select** next to **Create Your Own Policy**.

4. In the **Policy Name** field, type your own unique name, such as `AmazonECSTaskS3BucketPolicy`.

5. In the **Policy Document** field, paste the policy to apply to your tasks. The example below allows permission to the *my-task-secrets-bucket* Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1465589882000",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-task-secrets-bucket/*"
      ]
    }
  ]
}
```

6. Choose **Create Policy** to finish.

To create an IAM role for your tasks

1. Open the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles** and then choose **Create New Role**.

3. In the **Role Name** field, enter a name for your role. For this example, type `AmazonECSTaskS3BucketRole` to name the role, and then choose **Next Step**.

4. In the **Select Role Type** section, choose **Select** next to the **Amazon EC2 Container Service Task Role** role.

5. In the **Attach Policy** section, select the policy you want to use for your tasks (in this example `AmazonECSTaskS3BucketPolicy`, and then choose **Next Step**.

6. Review your role information and then choose **Create Role** to finish.

Using a Supported AWS SDK

Support for IAM roles for tasks was added to the AWS SDKs on July 13th, 2016, so the containers in your tasks must use an AWS SDK version that was created on or after that date. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at Tools for Amazon Web Services when you are building your containers.

The following AWS SDK versions and above support IAM roles for tasks:

- AWS CLI: 1.10.47
- C++: 0.12.19
- CoreCLR: 3.2.6-beta
- Go: 1.2.5
- Java: 1.11.16
- .NET: 3.1.6
- Node.js: 2.4.7

- PHP: 3.18.28
- Python (`botocore`): 1.4.37
- Python (Boto3): 1.4.0

  Note

  The `botocore` module provides the low-level core functionality for Boto3, and each version of Boto3 supports a range of `botocore` module versions. For Boto3 support of IAM roles for tasks, you must ensure that your underlying `botocore` module is at least the minimum version shown above.

- Ruby: 2.3.22

Specifying an IAM Role for your Tasks

After you have created a role and attached a policy to that role, you can run tasks that assume the role. You have several options to do this:

- Specify an IAM role for your tasks in the task definition. You can create a new task definition or a new revision of an existing task definition and specify the role you created previously. If you use the console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter. For more information, see .

  Note

  This option is required if you want to use IAM task roles in an Amazon ECS service.
- Specify an IAM task role override when running a task. You can specify an IAM task role override when running a task. If you use the console to run your task, choose **Advanced Options** and then choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter in the `overrides` JSON object. For more information, see .

Amazon ECS IAM Policy Examples

The following examples show policy statements that you could use to control the permissions that IAM users have to Amazon ECS.

Topics

Amazon ECS First Run Wizard

The Amazon ECS first run wizard simplifies the process of creating a cluster and running your tasks and services. However, users require permissions to many API operations from multiple AWS services

to complete the wizard. The policy below shows the required permissions to complete the Amazon ECS first run wizard.

Note

If you want to create an Amazon ECR repository in the first run wizard, tag and push an image to that repository, and use that image in an Amazon ECS task definition, then your user also needs the permissions listed in the `AmazonEC2ContainerRegistryFullAccess` managed policy. For more information, see Amazon ECR Managed Policies (p. 178).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CreateAutoScalingGroup",
        "autoscaling:CreateLaunchConfiguration",
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:DeleteLaunchConfiguration",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeAutoScalingNotificationTypes",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeTags",
        "autoscaling:DescribeTriggers",
        "autoscaling:UpdateAutoScalingGroup",
        "cloudformation:CreateStack",
        "cloudformation:DescribeStack*",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateKeyPair",
        "ec2:CreateNetworkInterface",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVpc",
        "ec2:DeleteInternetGateway",
        "ec2:DeleteRoute",
        "ec2:DeleteRouteTable",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSubnet",
        "ec2:DeleteTags",
        "ec2:DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeNetworkInterface",
```

```
          "ec2:DescribeRouteTables",
          "ec2:DescribeSecurityGroups",
          "ec2:DescribeSubnets",
          "ec2:DescribeTags",
          "ec2:DescribeVpcAttribute",
          "ec2:DescribeVpcs",
          "ec2:DetachInternetGateway",
          "ec2:DisassociateRouteTable",
          "ec2:ModifyVpcAttribute",
          "ec2:RunInstances",
          "ec2:TerminateInstances",
          "ecr:*",
          "ecs:*",
          "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
          "elasticloadbalancing:AttachLoadBalancerToSubnets",
          "elasticloadbalancing:ConfigureHealthCheck",
          "elasticloadbalancing:CreateLoadBalancer",
          "elasticloadbalancing:DeleteLoadBalancer",
          "elasticloadbalancing:DeleteLoadBalancerListeners",
          "elasticloadbalancing:DeleteLoadBalancerPolicy",
          "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
          "elasticloadbalancing:DescribeInstanceHealth",
          "elasticloadbalancing:DescribeLoadBalancerAttributes",
          "elasticloadbalancing:DescribeLoadBalancerPolicies",
          "elasticloadbalancing:DescribeLoadBalancerPolicyTypes",
          "elasticloadbalancing:DescribeLoadBalancers",
          "elasticloadbalancing:ModifyLoadBalancerAttributes",
          "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
          "iam:AttachRolePolicy",
          "iam:CreateRole",
          "iam:GetPolicy",
          "iam:GetPolicyVersion",
          "iam:GetRole",
          "iam:ListAttachedRolePolicies",
          "iam:ListInstanceProfiles",
          "iam:ListRoles",
          "iam:ListGroups",
          "iam:ListUsers",
          "iam:CreateInstanceProfile",
          "iam:AddRoleToInstanceProfile",
          "iam:ListInstanceProfilesForRole"
        ],
        "Resource": "*"
      }
    ]
}
```

Clusters

The following IAM policy allows permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to * for all resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following IAM policy allows permission to describe and delete a specific cluster. The
`DescribeCluster` and `DeleteCluster` actions accept cluster ARNs as resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeCluster",
        "ecs:DeleteCluster"
      ],
      "Resource": [
        "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/<cluster_name>"
      ]
    }
  ]
}
```

The following IAM policy can be attached to a user or group that would only allow that user or group to
perform operations on a specific cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecs:Describe*",
        "ecs:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "ecs:DeleteCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:ListContainerInstances",
        "ecs:RegisterContainerInstance",
        "ecs:SubmitContainerStateChange",
        "ecs:SubmitTaskStateChange"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
    },
    {
```

```
      "Action": [
        "ecs:DescribeContainerInstances",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:UpdateContainerAgent",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:RunTask"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/
default"
        }
      }
    }
  ]
}
```

Container Instances

Container instance registration is handled by the Amazon ECS agent, but there may be times where you want to allow a user to deregister an instance manually from a cluster. Perhaps the container instance was accidentally registered to the wrong cluster, or the instance was terminated with tasks still running on it.

The following IAM policy allows a user to list and deregister container instances in a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DeregisterContainerInstance",
        "ecs:ListContainerInstances"
      ],
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
      ]
    }
  ]
}
```

The following IAM policy allows a user to describe a specified container instance in a specified cluster. To open this permission up to all container instances in a cluster, you can replace the container instance UUID with *.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeContainerInstance"
```

```
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<container_instance_UUID>"
      ]
    }
  ]
}
```

Task Definitions

Task definition IAM policies do not support resource-level permissions, but the following IAM policy allows a user to register, list, and describe task definitions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:RegisterTaskDefinition",
        "ecs:ListTaskDefinitions",
        "ecs:DescribeTaskDefinition"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Run Tasks

The resources for `RunTask` are task definitions. To limit which clusters a user can run task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to run any revision of a specific task definition on a specific cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:RunTask"
      ],
      "Condition": {
```

```
      "ArnEquals": {
        "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs::<region>:<aws_account_id>:task-definition/
<task_family>:*"
      ]
    }
  ]
}
```

Start Tasks

The resources for `StartTask` are task definitions. To limit which clusters and container instances a user can start task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to start any revision of a specific task definition on a specific cluster and specific container instance:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:StartTask"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>",
          "ecs:container-instances" : [
            "arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<container_instance_UUID>"
          ]
        }
      },
      "Resource": [
        "arn:aws:ecs::<region>:<aws_account_id>:task-definition/
<task_family>:*"
      ]
    }
  ]
}
```

List and Describe Tasks

The following IAM policy allows a user to list tasks for a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "ecs:ListTasks"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"
        }
      },
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following IAM policy allows a user to describe a specified task in a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeTask"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task/<task_UUID>"
      ]
    }
  ]
}
```

Create Services

The following IAM policy allows a user to create Amazon ECS services in the AWS Management
Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
```

```
            "ecs:List*",
            "ecs:Describe*",
            "ecs:CreateService",
            "elasticloadbalancing:Describe*",
            "iam:AttachRolePolicy",
            "iam:CreateRole",
            "iam:GetPolicy",
            "iam:GetPolicyVersion",
            "iam:GetRole",
            "iam:ListAttachedRolePolicies",
            "iam:ListRoles",
            "iam:ListGroups",
            "iam:ListUsers"
        ],
        "Resource": [
          "*"
        ]
      }
    ]
}
```

Update Services

The following IAM policy allows a user to update Amazon ECS services in the AWS Management
Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs:List*",
        "ecs:Describe*",
        "ecs:UpdateService",
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:ListRoles",
        "iam:ListGroups",
        "iam:ListUsers"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Using the Amazon ECS Command Line Interface

The Amazon EC2 Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose, a popular open-source tool for defining and running multi-container applications. Use the CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

> **Note**
> The source code for the Amazon ECS CLI is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently provide support for running modified copies of this software.

Topics

Installing the Amazon ECS CLI

Follow these instructions to install the Amazon ECS CLI on your Mac OSX or Linux system.

> **Note**
> The Amazon ECS CLI is not available for Windows systems at this time.

To install the Amazon ECS CLI

1. Download the binary.

    - For Mac OSX:

```
$ sudo curl -o /usr/local/bin/ecs-cli https://s3.amazonaws.com/amazon-
  ecs-cli/ecs-cli-darwin-amd64-latest
```

- For Linux systems:

```
$ sudo curl -o /usr/local/bin/ecs-cli https://s3.amazonaws.com/amazon-
  ecs-cli/ecs-cli-linux-amd64-latest
```

2. (Optional) Verify the downloaded binary with the MD5 sum provided.
   - For Mac OSX: https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-darwin-amd64-latest.md5
   - For Linux systems: https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-linux-amd64-latest.md5
3. Apply execute permissions to the binary.

```
$ sudo chmod +x /usr/local/bin/ecs-cli
```

4. Verify that the CLI is working properly.

```
$ ecs-cli --version
ecs-cli version 0.4.1 (e27df48)
```

After you have installed the CLI, proceed to Configuring the Amazon ECS CLI (p. 200).

Configuring the Amazon ECS CLI

The Amazon ECS CLI requires some basic configuration information before you can use it, such as your AWS credentials, the AWS region in which to create your cluster, and the name of the Amazon ECS cluster to use with the **ecs-cli configure** command. These settings are stored in the `~/.ecs/config` file.

AWS Credentials

The Amazon ECS CLI requires your AWS credentials to make calls to AWS APIs on your behalf (for more information, see Managing Access Keys for your AWS Account in the *Amazon Web Services General Reference*). You can configure your AWS credentials in several ways:

- You can set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables. When you run **ecs-cli configure**, the values of those variables are stored in the Amazon ECS CLI configuration file.
- You can use a named profile from the `~/.aws/credentials` file on your system, if you have previously configured your AWS credentials there for another tool, such as the AWS CLI. You can follow the Quick Configuration instructions in the *AWS Command Line Interface User Guide* to set up a `default` profile if you have not already done so. You can then pass this named profile as `--profile default` when you run the **ecs-cli configure** command.
- You can pass credentials directly on the command line with the `--access-key` and `--secret-key` options.

To configure the Amazon ECS CLI

- Configure the CLI with the following command, substituting `us-west-2` with your desired AWS region, `ecs-cli-demo` with the name of an existing Amazon ECS cluster or a new cluster to use, and the `$AWS_ACCESS_KEY_ID` and `$AWS_SECRET_ACCESS_KEY` environment variables with your AWS credentials.

```
$ ecs-cli configure --region us-west-2 --access-key $AWS_ACCESS_KEY_ID --
secret-key $AWS_SECRET_ACCESS_KEY --cluster ecs-cli-demo
INFO[0000] Saved ECS CLI configuration for cluster (ecs-cli-demo)
```

After you have installed and configured the CLI, you can try the Amazon ECS CLI Tutorial (p. 201). For more information, see the ECS CLI Command Line Reference (p. 206).

Amazon ECS CLI Tutorial

This simple tutorial shows a few of the different commands and capabilities of the Amazon ECS CLI. Before you can start this tutorial, you must install and configure the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 199).

Topics

Step 1: Create your Cluster

The first action you should take is to create a cluster of Amazon ECS container instances that you can launch your containers on with the **ecs-cli up** command. There are many options that you can choose to configure your cluster with this command, but most of them are optional. In this example, you create a simple cluster of two t2.medium container instances that use the id_rsa key pair for SSH access (substitute your own key pair here).

By default, the security group created for your container instances opens port 80 for inbound traffic. You can use the --port option to specify a different port to open, or if you have more complicated security group requirements, you can specify an existing security group to use with the --security-group option.

```
$ ecs-cli up --keypair id_rsa --capability-iam --size 2 --instance-type
 t2.medium
INFO[0000] Created cluster                                 cluster=ecs-cli-demo
INFO[0000] Waiting for your cluster resources to be created
INFO[0001] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0061] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0121] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0182] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
```

This command may take a few minutes to complete as your resources are created. Now that you have a cluster, you can create a Docker compose file and deploy it.

Step 2: Create a Compose File

For this step, create a simple Docker compose file that creates a WordPress application consisting of a web server and a MySQL database. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1 and 2.

The following parameters are supported in compose files for the Amazon ECS CLI:

- `command`
- `cpu_shares`
- `dns`
- `dns_search`
- `entrypoint`
- `environment`: If an environment variable value is not specified in the compose file, but it exists in the shell environment, the shell environment variable value is passed to the task definition that is created for any associated tasks or services.

    Important
    We do not recommend using plain text environment variables for sensitive information, such as credential data.

- `env_file`

    Important
    We do not recommend using plain text environment variables for sensitive information, such as credential data.

- `extra_hosts`
- `hostname`
- `image`
- `labels`
- `links`
- `log_driver`
- `log_opt`
- `mem_limit` (in bytes)
- `ports`
- `privileged`
- `read_only`
- `security_opt`
- `ulimits`
- `user`
- `volumes`
- `volumes_from`
- `working_dir`

    Important
    The `build` directive is not supported at this time.

For more information about Docker compose file syntax, see the Compose file reference in the Docker documentation.

Here is the compose file, which you can call `hello-world.yml`. Each container has 100 CPU units and 500 MiB of memory. The `wordpress` container exposes port 80 to the container instance for inbound traffic to the web server.

```
version: '2'
services:
  wordpress:
    image: wordpress
    cpu_shares: 100
    mem_limit: 524288000
    ports:
      - "80:80"
    links:
      - mysql
  mysql:
    image: mysql
    cpu_shares: 100
    mem_limit: 524288000
    environment:
      MYSQL_ROOT_PASSWORD: password
```

Step 3: Deploy the Compose File to a Cluster

After you create the compose file, you can deploy it to your cluster with the **ecs-cli compose up** command. By default, the command looks for a file called docker-compose.yml in the current directory, but you can specify a different file with the `--file` option. By default, the resources created by this command have the current directory in the title, but you can override that with the `--project-name` *project_name* option.

```
$ ecs-cli compose --file hello-world.yml up
INFO[0000] Using ECS task definition
 TaskDefinition=ecscompose-docker-compose:2
INFO[0000] Starting container...                       container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress
INFO[0000] Starting container...                       container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql
INFO[0000] Describe ECS container status               container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress desiredStatus=RUNNING
 lastStatus=PENDING taskDefinition=ecscompose-docker-compose:2
INFO[0000] Describe ECS container status               container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql desiredStatus=RUNNING lastStatus=PENDING
 taskDefinition=ecscompose-docker-compose:2
INFO[0054] Started container...                        container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress desiredStatus=RUNNING
 lastStatus=RUNNING taskDefinition=ecscompose-docker-compose:2
INFO[0054] Started container...                        container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql desiredStatus=RUNNING lastStatus=RUNNING
 taskDefinition=ecscompose-docker-compose:2
```

Step 4: View the Running Containers on a Cluster

After you deploy the compose file, you can view the containers that are running on your cluster with the **ecs-cli ps** command.

```
$ ecs-cli ps
Name                                              State    Ports
     TaskDefinition
340488e0-a307-4322-b41c-99f1b70e97f9/wordpress  RUNNING  52.89.204.137:80-
>80/tcp  ecscompose-docker-compose:2
```

```
340488e0-a307-4322-b41c-99f1b70e97f9/mysql        RUNNING
     ecscompose-docker-compose:2
```

In the above example, you can see the `wordpress` and `mysql` containers from your compose file, and also the IP address and port of the web server. If you point a web browser to that address, you should see the WordPress installation wizard.

Step 5: Scale the Tasks on a Cluster

You can scale your task count up so you could have more instances of your application with the **ecs-cli compose scale** command. In this example, you can increase the count of your application to two.

```
$ ecs-cli compose --file hello-world.yml scale 2
```

Now you should see two more containers in your cluster.

```
$ ecs-cli ps
Name                                           State    Ports
     TaskDefinition
340488e0-a307-4322-b41c-99f1b70e97f9/wordpress  RUNNING  52.89.204.137:80-
>80/tcp  ecscompose-docker-compose:2
340488e0-a307-4322-b41c-99f1b70e97f9/mysql       RUNNING
     ecscompose-docker-compose:2
f80d82d5-3724-4f2f-86b1-5ee5891ce995/mysql       RUNNING
     ecscompose-docker-compose:2
f80d82d5-3724-4f2f-86b1-5ee5891ce995/wordpress  RUNNING  52.89.205.89:80->80/
tcp    ecscompose-docker-compose:2
```

Step 6: Create an ECS Service from a Compose File

Now that you know that your containers work properly, you can make sure that they are replaced if they fail or stop. You can do this by creating a service from your compose file with the **ecs-cli compose service up** command. This command creates a task definition from the latest compose file (if it does not already exist) and creates an ECS service with it, with a desired count of 1.

Before starting your service, stop the containers from your compose file with the **ecs-cli compose down** command so that you have an empty cluster to work with.

```
$ ecs-cli compose --file hello-world.yml down
INFO[0000] Stopping container...                        container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress
INFO[0000] Stopping container...                        container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql
INFO[0000] Stopping container...
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/mysql
INFO[0000] Stopping container...
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/wordpress
INFO[0000] Describe ECS container status
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/mysql desiredStatus=STOPPED
 lastStatus=RUNNING taskDefinition=ecscompose-docker-compose:2
INFO[0000] Describe ECS container status
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/wordpress
 desiredStatus=STOPPED lastStatus=RUNNING taskDefinition=ecscompose-docker-
compose:2
```

```
INFO[0000] Describe ECS container status                 container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress desiredStatus=STOPPED
 lastStatus=RUNNING taskDefinition=ecscompose-docker-compose:2
INFO[0000] Describe ECS container status                 container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql desiredStatus=STOPPED lastStatus=RUNNING
 taskDefinition=ecscompose-docker-compose:2
INFO[0006] Stopped container...                          container=340488e0-
a307-4322-b41c-99f1b70e97f9/wordpress desiredStatus=STOPPED
 lastStatus=STOPPED taskDefinition=ecscompose-docker-compose:2
INFO[0006] Stopped container...                          container=340488e0-
a307-4322-b41c-99f1b70e97f9/mysql desiredStatus=STOPPED lastStatus=STOPPED
 taskDefinition=ecscompose-docker-compose:2
INFO[0006] Stopped container...
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/mysql desiredStatus=STOPPED
 lastStatus=STOPPED taskDefinition=ecscompose-docker-compose:2
INFO[0006] Stopped container...
 container=f80d82d5-3724-4f2f-86b1-5ee5891ce995/wordpress
 desiredStatus=STOPPED lastStatus=STOPPED taskDefinition=ecscompose-docker-
compose:2
```

Now you can create your service.

```
$ ecs-cli compose --file hello-world.yml service up
INFO[0000] Using ECS task definition
 TaskDefinition=ecscompose-docker-compose:2
INFO[0000] Created an ECS Service
 serviceName=ecscompose-service-docker-compose taskDefinition=ecscompose-
docker-compose:2
INFO[0000] Updated ECS service successfully              desiredCount=1
 serviceName=ecscompose-service-docker-compose
INFO[0000] Describe ECS Service status                   desiredCount=1
 runningCount=0 serviceName=ecscompose-service-docker-compose
INFO[0015] ECS Service has reached a stable state        desiredCount=1
 runningCount=1 serviceName=ecscompose-service-docker-compose
```

Step 7: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

```
$ ecs-cli compose --file hello-world.yml service rm
INFO[0000] Updated ECS service successfully              desiredCount=0
 serviceName=ecscompose-service-docker-compose
INFO[0000] Describe ECS Service status                   desiredCount=0
 runningCount=1 serviceName=ecscompose-service-docker-compose
INFO[0015] ECS Service has reached a stable state        desiredCount=0
 runningCount=0 serviceName=ecscompose-service-docker-compose
INFO[0015] Deleted ECS service                           service=ecscompose-
service-docker-compose
INFO[0015] ECS Service has reached a stable state        desiredCount=0
 runningCount=0 serviceName=ecscompose-service-docker-compose
```

Now, take down your cluster, which cleans up the resources that you created earlier with **ecs-cli up**.

```
$ ecs-cli down --force
```

```
INFO[0000] Waiting for your cluster resources to be deleted
INFO[0000] Cloudformation stack status
 stackStatus=DELETE_IN_PROGRESS
INFO[0061] Cloudformation stack status
 stackStatus=DELETE_IN_PROGRESS
INFO[0121] Deleted cluster                                    cluster=ecs-cli-demo
```

ECS CLI Command Line Reference

The following commands are available in the Amazon ECS CLI. Help text for each command is available by appending the `--help` option to the final command argument; for example, help text for **ecs-cli compose service up** is displayed with the following command:

```
$ ecs-cli compose service up --help
```

Available Commands

ecs-cli

Description

The Amazon EC2 Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose, a popular open-source tool for defining and running multi-container applications.

For a quick walkthrough of the ECS CLI, see the Amazon ECS CLI Tutorial (p. 201).

Help text is available for each individual subcommand with **ecs-cli *subcommand* --help**.

Syntax

ecs-cli [--version] [*subcommand*] [--help]

Options

Name	Description
--version, -v	Prints the version information for the Amazon ECS CLI. Required: No

Name	Description
`--help, -h`	Show the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli** command supports the following subcommands:

configure
Configures your AWS credentials, the AWS region to use, and the Amazon ECS cluster name to use with the Amazon ECS CLI. For more information, see ecs-cli configure (p. 207).

up
Creates the ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster. For more information, see ecs-cli up (p. 210).

down
Deletes the CloudFormation stack that was created by **ecs-cli up** and the associated resources. For more information, see ecs-cli down (p. 213).

scale
Modifies the number of container instances in an ECS cluster. For more information, see ecs-cli scale (p. 214).

ps
Lists all of the running containers in an ECS cluster. For more information, see ecs-cli ps (p. 215).

license
Prints the `LICENSE` files for the ECS CLI and its dependencies. For more information, see ecs-cli license (p. 216).

compose
Executes **docker-compose**–style commands on an ECS cluster. For more information, see ecs-cli compose (p. 216).

help
Shows the help text for the specified command.

ecs-cli configure

Description

Configures your AWS credentials, the AWS region to use, resource creation prefixes, and the ECS cluster name to use with the Amazon ECS CLI. The resulting configuration is stored in the `~/.ecs/config` file.

Each time you run the **ecs-cli configure** command, the configuration values in `~/.ecs/config` are replaced with the values from the latest command (and if existing configuration parameters are not specified with their associated option flags or environment variables, they are removed or replaced with the default values).

Syntax

```
ecs-cli configure [--region region] [--access-key aws_access_key_id] [--
secret-key aws_secret_access_key] [--profile profile_name] --cluster
```

cluster_name [--compose-project-name-prefix *prefix*] [--compose-service-name-
prefix *prefix*] [--cfn-stack-name-prefix *prefix*] [--help]

Options

Name	Description
--region, -r *region*	Specifies the AWS region to use. If the AWS_REGION environment variable is set when **ecs-cli configure** is run, then the AWS region is set to the value of that environment variable. Type: String Required: No
--access-key *aws_access_key_id*	Specifies the AWS access key to use. If the AWS_ACCESS_KEY_ID environment variable is set when **ecs-cli configure** is run, then the AWS access key ID is set to the value of that environment variable. Type: String Required: No
--secret-key *aws_secret_access_key*	Specifies the AWS secret key to use. If the AWS_SECRET_ACCESS_KEY environment variable is set when **ecs-cli configure** is run, then the AWS secret access key is set to the value of that environment variable. Type: String Required: No
--profile, -p *profile_name*	Specifies your AWS credentials with an existing named profile from ~/.aws/credentials. If the AWS_PROFILE environment variable is set when **ecs-cli configure** is run, then the AWS named profile is set to the value of that environment variable. Type: String Required: No
--cluster, -c *cluster_name*	Specifies the ECS cluster name to use. If the cluster does not exist, it is created when you try to add resources to it with the **ecs-cli up** command. Type: String Required: Yes
--compose-project-name-prefix *prefix*	Specifies the prefix to add to an ECS task definition that is registered from a compose file. You can specify an empty string (--compose-project-name-prefix " ") with this option to omit the default prefix. **Important** This prefix is used to name and later manage resources created by the ECS CLI. Resources that are created with a prefix are only addressable

Name	Description
	from the ECS CLI if the configured prefix matches the prefix that was used when the resource was created. Before you change the prefix value, you should consider the effects on any active resources.
	Type: String
	Default: `ecscompose-`
	Required: No
`--compose-service-name-prefix` *prefix*	Specifies the prefix to add to an ECS service that is created from a compose file. You can specify an empty string (`--compose-service-name-prefix ""`) with this option to omit the default prefix.
	Important This prefix is used to name and later manage resources created by the ECS CLI. Resources that are created with a prefix are only addressable from the ECS CLI if the configured prefix matches the prefix that was used when the resource was created. Before you change the prefix value, you should consider the effects on any active resources.
	Type: String
	Default: `ecscompose-service-`
	Required: No
`--cfn-stack-name-prefix` *prefix*	Specifies the prefix to add to the AWS CloudFormation stack that is created on **ecs-cli up**. You can specify an empty string (`--cfn-stack-name-prefix ""`) with this option to omit the default prefix.
	Important This prefix is used to name and later manage resources created by the ECS CLI. Resources that are created with a prefix are only addressable from the ECS CLI if the configured prefix matches the prefix that was used when the resource was created. Before you change the prefix value, you should consider the effects on any active resources.
	Type: String
	Default: `amazon-ecs-cli-setup-`
	Required: No
`--help, -h`	Show the help text for the specified command.
	Required: No

Examples

Example

This example configures the ECS CLI to create and/or use a cluster called `ecs-cli` in the `us-west-2` region.

```
$ ecs-cli configure --region us-west-2 --access-key $AWS_ACCESS_KEY_ID --
secret-key $AWS_SECRET_ACCESS_KEY --cluster ecs-cli
INFO[0000] Saved ECS CLI configuration for cluster (ecs-cli)
```

Example

This example configures the ECS CLI to create and/or use a cluster called `ecs-cli` in the `us-west-2` region and omit the default ECS CLI prefixes on future resource creation.

> **Note**
> Any existing resources, such as task definitions, services, or AWS CloudFormation stacks, that were created with the default prefixes will not be addressable from the ECS CLI until the configured prefix matches the prefix that was used when the resource was created.

```
$ ecs-cli configure --region us-west-2 --access-key $AWS_ACCESS_KEY_ID --
secret-key $AWS_SECRET_ACCESS_KEY --cluster ecs-cli --compose-project-name-
prefix "" --compose-service-name-prefix "" --cfn-stack-name-prefix ""
INFO[0000] Saved ECS CLI configuration for cluster (ecs-cli)
```

ecs-cli up

Description

Create the ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster.

This command creates a new AWS CloudFormation stack called `amazon-ecs-cli-setup-cluster_name`; you can view the progress of the stack creation in the AWS Management Console.

Syntax

```
ecs-cli up --keypair keypair_name --capability-iam [--size n] [--azs
availability_zone_1,availability_zone_2] [--security-group security_group_id]
[--cidr ip_range] [--port port_number] [--subnets subnet_1,subnet_2] [--vpc
vpc_id] [--instance-type instance_type] [--image-id ami_id] [--help]
```

Options

Name	Description
`--verbose, --debug`	Provide more verbose output for debugging purposes. Required: No
`--keypair keypair_name`	Specifies the name of an existing Amazon EC2 key pair to enable SSH access to the EC2 instances in your cluster.

Name	Description
	For help creating a key pair, see Setting Up with Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.
	Type: String
	Required: Yes
`--capability-iam`	Acknowledges that this command may create IAM resources.
	Required: Yes
`--size` *n*	Specifies the number of instances to launch and register to the cluster.
	Type: Integer
	Default: 1
	Required: No
`--azs` *availability_zone_1,availability_zone_2*	Specifies a comma-separated list of 2 VPC Availability Zones in which to create subnets (these zones must have the `available` status). This option is recommended if you do not specify a VPC ID with the `--vpc` option.
	Warning
	Leaving this option blank can result in failure to launch container instances if an unavailable zone is chosen at random.
	Type: String
	Required: No
`--security-group` *security_group_id*	Specifies an existing security group to associate with your container instances. If you do not specify a security group here, then a new one is created.
	For more information, see Security Groups in the *Amazon EC2 User Guide for Linux Instances*.
	Required: No
`--cidr` *ip_range*	Specifies a CIDR/IP range for the security group to use for container instances in your cluster.
	Note
	This parameter is ignored if an existing security group is specified with the `--security-group` option.
	Type: CIDR/IP range
	Default: `0.0.0.0/0`
	Required: No

Name	Description
--port *port_number*	Specifies a port to open on the security group to use for container instances in your cluster. **Note** This parameter is ignored if an existing security group is specified with the `--security-group` option. Type: Integer Default: `80` Required: No
--subnets *subnet_1,subnet_2*	Specifies a comma-separated list of existing VPC Subnet IDs in which to launch your container instances. Type: String Required: This option is required if you specify a VPC with the `--vpc` option.
--vpc *vpc_id*	Specifies the ID of an existing VPC in which to launch your container instances. If you specify a VPC ID, you must specify a list of existing subnets in that VPC with the `--subnets` option. If you do not specify a VPC ID, a new VPC is created with two subnets. Type: String Required: No
--instance-type *instance_type*	Specifies the EC2 instance type for your container instances. For more information on EC2 instance types, see Amazon EC2 Instances. Type: String Default: `t2.micro` Required: No
--image-id *ami_id*	Specifies the EC2 AMI ID to use for your container instances. Type: String Default: The latest Amazon ECS-optimized AMI for the specified region. Required: No
--help, -h	Show the help text for the specified command. Required: No

Examples

Example

This example brings up a cluster of 4 `c4.large` instances and configures them to use the EC2 keypair called `id_rsa`.

```
$ ecs-cli up --keypair id_rsa --capability-iam --size 4 --instance-type
 c4.large
INFO[0000] Created cluster                                 cluster=ecs-cli
INFO[0000] Waiting for your cluster resources to be created
INFO[0001] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0061] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0121] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0181] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
```

ecs-cli down

Description

Deletes the CloudFormation stack that was created by **ecs-cli up** and the associated resources. The `--force` option is required.

> **Note**
> The ECS CLI can only manage tasks, services, and container instances that were created with the ECS CLI. To manage tasks, services, and container instances that were not created by the ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

The **ecs-cli down** command attempts to delete the cluster specified in `~/.ecs/config`. However, if there are any active services (even with a desired count of 0) or registered container instances in your cluster that were not created by **ecs-cli up** (for example, if you used an existing ECS cluster with registered container instances, such as the default cluster), the cluster is not deleted and the services and pre-existing container instances remain active.

If you have remaining services or container instances in your cluster that you would like to remove, you can follow the procedures in to remove them and then delete your cluster.

Syntax

```
ecs-cli down --force [--help]
```

Options

Name	Description
`--force, -f`	Acknowledges that this command permanently deletes resources. Required: Yes

Name	Description
`--help, -h`	Show the help text for the specified command. Required: No

Examples

Example

This example brings up a cluster of 4 `c4.large` instances and configures them to use the EC2 keypair called `id_rsa`.

```
$ ecs-cli down --force
INFO[0001] Waiting for your cluster resources to be deleted
INFO[0001] Cloudformation stack status
 stackStatus=DELETE_IN_PROGRESS
INFO[0062] Cloudformation stack status
 stackStatus=DELETE_IN_PROGRESS
INFO[0123] Cloudformation stack status
 stackStatus=DELETE_IN_PROGRESS
INFO[0154] Deleted cluster
```

ecs-cli scale

Description

Modifies the number of container instances in your cluster. This command changes the desired and maximum instance count in the Auto Scaling group created by the **ecs-cli up** command. You can use this command to scale up (increase the number of instances) or scale down (decrease the number of instances) your cluster.

> **Note**
> The ECS CLI can only manage tasks, services, and container instances that were created with the ECS CLI. To manage tasks, services, and container instances that were not created by the ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

Syntax

ecs-cli scale --capability-iam --size *n* [--help]

Options

Name	Description
`--capability-iam`	Acknowledges that this command may create IAM resources. Required: Yes
`--size *n*`	Specifies the number of instances to maintain in your cluster. Type: Integer Required: Yes

Name	Description
`--help, -h`	Show the help text for the specified command. Required: No

Examples

Example

This example scales the current cluster to 2 container instances.

```
$ ecs-cli scale --capability-iam --size 2
INFO[0001] Waiting for your cluster resources to be updated
INFO[0001] Cloudformation stack status
 stackStatus=UPDATE_IN_PROGRESS
```

ecs-cli ps

Description

Lists all of the running containers in your ECS cluster.

Syntax

`ecs-cli ps [--help]`

Options

Name	Description
`--help, -h`	Show the help text for the specified command. Required: No

Examples

Example

This example shows the containers that are running in the cluster.

```
$ ecs-cli ps
Name                                            State     Ports
      TaskDefinition
595deba7-16a1-4aaf-9b27-e152eba03ccc/wordpress  RUNNING   52.33.62.24:80->80/
tcp    ecscompose-hello-world:3
595deba7-16a1-4aaf-9b27-e152eba03ccc/mysql      RUNNING
      ecscompose-hello-world:3
7fc0a2a4-9202-47d2-8b06-4463286b63de/mysql      RUNNING
      ecscompose-hello-world:3
7fc0a2a4-9202-47d2-8b06-4463286b63de/wordpress  RUNNING   52.32.232.166:80-
>80/tcp   ecscompose-hello-world:3
```

ecs-cli license

Description

Prints the `LICENSE` files for the ECS CLI and its dependencies.

Syntax

`ecs-cli license [--help]`

Options

Name	Description
`--help, -h`	Show the help text for the specified command.
	Required: No

Examples

Example

This example prints the license files.

```
$ ecs-cli license
Copyright 2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"). You may not
 use this file except in compliance with the
License. A copy of the License is located at

    http://aws.amazon.com/apache2.0/

or in the "license" file accompanying this file. This file is distributed on
 an "AS IS" BASIS, WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND, either express or implied. See the License for the
 specific language governing permissions
and limitations under the License.
...
```

ecs-cli compose

Description

Manage Amazon ECS tasks with **docker-compose**-style commands on an ECS cluster.

> **Note**
>
> If you want to create Amazon ECS services with the ECS CLI, see ecs-cli compose service (p. 219).

The **ecs-cli compose** command works with a Docker compose file to create task definitions and manage tasks. At this time, the latest version of the Amazon ECS CLI supports Docker compose file syntax versions 1 and 2. By default, the command looks for a compose file in the current directory, called `docker-compose.yml`; however, you can specify a different file name or path to a compose file with the `--file` option. This is especially useful for managing tasks and services from multiple compose files at a time with the ECS CLI.

The **ecs-cli compose** command uses a project name with the task definitions and services it creates. When the CLI creates a task definition from a compose file, the task definition is called `ecscompose-`*`project-name`*. When the CLI creates a service from a compose file, the service is called `ecscompose-service-`*`project-name`*. By default, the project name is the name of the current working directory. However, you can specify your own project name with the `--project-name` option.

> **Note**
> The ECS CLI can only manage tasks, services, and container instances that were created with the ECS CLI. To manage tasks, services, and container instances that were not created by the ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

The following parameters are supported in compose files for the Amazon ECS CLI:

- `command`
- `cpu_shares`
- `dns`
- `dns_search`
- `entrypoint`
- `environment`: If an environment variable value is not specified in the compose file, but it exists in the shell environment, the shell environment variable value is passed to the task definition that is created for any associated tasks or services.

  > **Important**
  > We do not recommend using plain text environment variables for sensitive information, such as credential data.

- `env_file`

  > **Important**
  > We do not recommend using plain text environment variables for sensitive information, such as credential data.

- `extra_hosts`
- `hostname`
- `image`
- `labels`
- `links`
- `log_driver`
- `log_opt`
- `mem_limit` (in bytes)
- `ports`
- `privileged`
- `read_only`
- `security_opt`
- `ulimits`
- `user`
- `volumes`
- `volumes_from`
- `working_dir`

> **Important**
> The `build` directive is not supported at this time.

For more information about Docker compose file syntax, see the Compose file reference in the Docker documentation.

Syntax

```
ecs-cli compose [--verbose] [--file compose-file] [--project-name project-
name] [subcommand] [arguments] [--help]
```

Options

Name	Description
`--verbose, --debug`	Increase the verbosity of command output to aid in diagnostics. Required: No
`--file, -f compose-file`	Specifies the Docker compose file to use. At this time, the latest version of the Amazon ECS CLI supports Docker compose file syntax versions 1 and 2. If the `COMPOSE_FILE` environment variable is set when **ecs-cli compose** is run, then the Docker compose file is set to the value of that environment variable. Type: String Default: `./docker-compose.yml` Required: No
`--project-name, -p project-name`	Specifies the project name to use. If the `COMPOSE_PROJECT_NAME` environment variable is set when **ecs-cli compose** is run, then the project name is set to the value of that environment variable. Type: String Default: The current directory name. Required: No
`--help, -h`	Show the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli compose** command supports the following subcommands:

create
Creates an ECS task definition from your compose file.

start
Starts a single task from the task definition created from your compose file.

up
Creates an ECS task definition from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of **create** and **start**)

ps
Lists all the containers in your cluster that were started by the compose project.

scale *n*
Scales the number of running tasks to the specified count.

run [*containerName*] [*command*] ...

Starts all containers overriding commands with the supplied one-off commands for the containers.

stop

Stops all the running tasks created by the compose project.

service [*subcommand*]

Creates an ECS service from your compose file. For more information, see ecs-cli compose service (p. 219).

help

Shows the help text for the specified command.

Examples

Example 1

This example creates a task definition with the project name `hello-world` from the `hello-world.yml` compose file.

```
$ ecs-cli compose --project-name hello-world --file hello-world.yml create
INFO[0000] Using ECS task definition
 TaskDefinition=ecscompose-hello-world:5
```

ecs-cli compose service

Description

Manage Amazon ECS services with **docker-compose**-style commands on an ECS cluster.

> **Note**
> If you want run tasks Amazon ECS with the ECS CLI instead of creating services, see ecs-cli compose (p. 216).

The **ecs-cli compose service** command works with a Docker compose file to create task definitions and manage services. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1 and 2. By default, the command looks for a compose file in the current directory, called `docker-compose.yml`; however, you can specify a different file name or path to a compose file with the `--file` option. This is especially useful for managing tasks and services from multiple compose files at a time with the ECS CLI.

The **ecs-cli compose service** command uses a project name with the task definitions and services it creates. When the CLI creates a task definition from a compose file, the task definition is called `ecscompose-`*project-name*. When the CLI creates a service from a compose file, the service is called `ecscompose-service-`*project-name*. By default, the project name is the name of the current working directory. However, you can specify your own project name with the `--project-name` option.

> **Note**
> The ECS CLI can only manage tasks, services, and container instances that were created with the ECS CLI. To manage tasks, services, and container instances that were not created by the ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

The following parameters are supported in compose files for the Amazon ECS CLI:

- `command`
- `cpu_shares`
- `dns`

- `dns_search`
- `entrypoint`
- `environment`: If an environment variable value is not specified in the compose file, but it exists in the shell environment, the shell environment variable value is passed to the task definition that is created for any associated tasks or services.

  Important
  We do not recommend using plain text environment variables for sensitive information, such as credential data.
- `env_file`

  Important
  We do not recommend using plain text environment variables for sensitive information, such as credential data.
- `extra_hosts`
- `hostname`
- `image`
- `labels`
- `links`
- `log_driver`
- `log_opt`
- `mem_limit` (in bytes)
- `ports`
- `privileged`
- `read_only`
- `security_opt`
- `ulimits`
- `user`
- `volumes`
- `volumes_from`
- `working_dir`

  Important
  The `build` directive is not supported at this time.

For more information about Docker compose file syntax, see the Compose file reference in the Docker documentation.

Syntax

```
ecs-cli compose [--verbose] [--file compose-file] [--project-name project-name] service [subcommand] [arguments] [--help]
```

Options

Name	Description
`--verbose, --debug`	Increase the verbosity of command output to aid in diagnostics. Required: No

Name	Description
`--file, -f` *`compose-file`*	Specifies the Docker compose file to use. At this time, the latest version of the Amazon ECS CLI supports Docker compose file syntax versions 1 and 2. If the `COMPOSE_FILE` environment variable is set when **ecs-cli compose** is run, then the Docker compose file is set to the value of that environment variable. Type: String Default: `./docker-compose.yml` Required: No
`--project-name, -p` *`project-name`*	Specifies the project name to use. If the `COMPOSE_PROJECT_NAME` environment variable is set when **ecs-cli compose** is run, then the project name is set to the value of that environment variable. Type: String Default: The current directory name. Required: No
`--help, -h`	Show the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli compose service** command supports the following subcommands and arguments:

create [--deployment-max-percent *n*] [--deployment-min-healthy-percent *n*]
Creates an ECS service from your compose file. The service is created with a desired count of `0`, so no containers are started by this command.

The `--deployment-max-percent` option specifies the upper limit (as a percentage of the service's `desiredCount`) of the number of running tasks that can be running in a service during a deployment (the default value is `200`). The `--deployment-min-healthy-percent` option specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a deployment (the default value is `100`). For more information, see `maximumPercent` (p. 123) and `minimumHealthyPercent` (p. 123).

start
Starts one copy of each of the containers on the created ECS service. This command updates the desired count of the service to `1`.

up [--deployment-max-percent *n*] [--deployment-min-healthy-percent *n*]
Creates an ECS service from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of **create** and **start**). This command updates the desired count of the service to `1`.

The `--deployment-max-percent` option specifies the upper limit (as a percentage of the service's `desiredCount`) of the number of running tasks that can be running in a service during a deployment (the default value is `200`). The `--deployment-min-healthy-percent` option specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a

deployment (the default value is `100`). For more information, see `maximumPercent` (p. 123) and `minimumHealthyPercent` (p. 123).

ps

Lists all the containers in your cluster that belong to the service created with the compose project.

scale [--deployment-max-percent *n*] [--deployment-min-healthy-percent *n*] *n*

Scales the desired count of the service to the specified count.

The `--deployment-max-percent` option specifies the upper limit (as a percentage of the service's `desiredCount`) of the number of running tasks that can be running in a service during a deployment (the default value is `200`). The `--deployment-min-healthy-percent` option specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a deployment (the default value is `100`). For more information, see `maximumPercent` (p. 123) and `minimumHealthyPercent` (p. 123).

stop

Stops the running tasks that belong to the service created with the compose project. This command updates the desired count of the service to `0`.

rm

Updates the desired count of the service to `0` and then deletes the service.

help

Shows the help text for the specified command.

Examples

Example 1

This example brings up an Amazon ECS service with the project name `hello-world` from the `hello-world.yml` compose file.

```
$ ecs-cli compose --project-name hello-world --file hello-world.yml service
 up
INFO[0001] Using ECS task definition
 TaskDefinition=ecscompose-hello-world:3
INFO[0001] Created an ECS Service
 serviceName=ecscompose-service-hello-world taskDefinition=ecscompose-hello-
world:3
INFO[0002] Updated ECS service successfully              desiredCount=1
 serviceName=ecscompose-service-hello-world
INFO[0002] Describe ECS Service status                   desiredCount=1
 runningCount=0 serviceName=ecscompose-service-hello-world
INFO[0033] Describe ECS Service status                   desiredCount=1
 runningCount=0 serviceName=ecscompose-service-hello-world
INFO[0063] Describe ECS Service status                   desiredCount=1
 runningCount=0 serviceName=ecscompose-service-hello-world
INFO[0093] Describe ECS Service status                   desiredCount=1
 runningCount=0 serviceName=ecscompose-service-hello-world
INFO[0108] ECS Service has reached a stable state        desiredCount=1
 runningCount=1 serviceName=ecscompose-service-hello-world
```

Example 2

This example scales the service created by the `hello-world` project to a desired count of 2.

```
$ ecs-cli compose --project-name hello-world --file hello-world.yml service
 scale 2
```

```
INFO[0001] Updated ECS service successfully             desiredCount=2
 serviceName=ecscompose-service-hello-world
INFO[0001] Describe ECS Service status                  desiredCount=2
 runningCount=1 serviceName=ecscompose-service-hello-world
INFO[0032] Describe ECS Service status                  desiredCount=2
 runningCount=1 serviceName=ecscompose-service-hello-world
INFO[0063] ECS Service has reached a stable state       desiredCount=2
 runningCount=2 serviceName=ecscompose-service-hello-world
```

Example 3

This example scales the service created by the `hello-world` project to a desired count of 0 and then deletes the service.

```
$ ecs-cli compose --project-name hello-world --file hello-world.yml service
 rm
INFO[0000] Updated ECS service successfully             desiredCount=0
 serviceName=ecscompose-service-hello-world
INFO[0000] Describe ECS Service status                  desiredCount=0
 runningCount=2 serviceName=ecscompose-service-hello-world
INFO[0016] ECS Service has reached a stable state       desiredCount=0
 runningCount=0 serviceName=ecscompose-service-hello-world
INFO[0016] Deleted ECS service                          service=ecscompose-
service-hello-world
INFO[0016] ECS Service has reached a stable state       desiredCount=0
 runningCount=0 serviceName=ecscompose-service-hello-world
```

Using the AWS CLI with Amazon ECS

The following steps will help you set up a cluster, register a task definition, run a task, and perform other common scenarios in Amazon ECS with the AWS CLI.

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information on the AWS CLI, see http://aws.amazon.com/cli/.

For more information on the other tools available for managing your AWS resources, including the different AWS SDKs, IDE toolkits, and the Windows PowerShell command line tools, see http://aws.amazon.com/tools/.

Step 1: (Optional) Create a Cluster

By default, your account receives a `default` cluster when you launch your first container instance.

Note
The benefit of using the `default` cluster that is provided for you is that you don't have to specify the `--cluster` *cluster_name* option in the following commands. If you do create your own non-default cluster, you need to specify `--cluster` *cluster_name* for each command that you intend to use with that cluster.

However, you can create your own cluster with a unique name with the following command.

```
$ aws ecs create-cluster --cluster-name MyCluster
{
    "cluster": {
        "clusterName": "MyCluster",
        "status": "ACTIVE",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"
    }
}
```

Step 2: Launch an Instance with the Amazon ECS AMI

You must have an ECS container instance in your cluster before you can run tasks on it. If you do not already have any container instances in your cluster, see Launching an Amazon ECS Container Instance (p. 39) for more information. The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI Name	AMI ID	EC2 console link
us-east-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-eca289fb	Launch instance
us-east-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-446f3521	Launch instance
us-west-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-9fadf8ff	Launch instance
us-west-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-7abc111a	Launch instance
eu-west-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-a1491ad2	Launch instance
eu-central-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-54f5303b	Launch instance
ap-northeast-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-9cd57ffd	Launch instance
ap-southeast-1	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-a900a3ca	Launch instance
ap-southeast-2	**amzn-ami-2016.09.b-amazon-ecs-optimized**	ami-5781be34	Launch instance

Step 3: List Container Instances

Within a few minutes of launching your container instance, the Amazon ECS agent registers the instance with your default cluster. You can list the container instances in a cluster by running the following command:

```
$ aws ecs list-container-instances --cluster default
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/container_instance_ID"
    ]
}
```

Step 4: Describe your Container Instance

After you have the ARN or ID of a container instance, you can use the **describe-container-instances** command to get valuable information on the instance, such as remaining and registered CPU and memory resources.

```
$ aws ecs describe-container-instances --cluster default --container-
instances container_instance_ID
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
```

```
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": true,
            "containerInstanceArn": "arn:aws:ecs:us-
west-2:aws_account_id:container-instance/container_instance_ID",
            "pendingTasksCount": 0,
            "remainingResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "runningTasksCount": 0,
            "attributes": [
                {
                    "name": "com.amazonaws.ecs.capability.privileged-
container"
                },
                {
```

```
                         "name": "com.amazonaws.ecs.capability.docker-remote-
api.1.17"
                },
                {
                         "name": "com.amazonaws.ecs.capability.docker-remote-
api.1.18"
                },
                {
                         "name": "com.amazonaws.ecs.capability.docker-remote-
api.1.19"
                },
                {
                         "name": "com.amazonaws.ecs.capability.logging-
driver.json-file"
                },
                {
                         "name": "com.amazonaws.ecs.capability.logging-
driver.syslog"
                }
            ],
            "versionInfo": {
                "agentVersion": "1.5.0",
                "agentHash": "b197edd",
                "dockerVersion": "DockerVersion: 1.7.1"
            }
        }
    ]
}
```

You can also find the EC2 instance ID that you can use to monitor the instance in the Amazon EC2 console or with the **aws ec2 describe-instances --instance-id *instance_id*** command.

Step 5: Register a Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses a `busybox` image from Docker Hub and simply sleeps for 360 seconds. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 82).

```
{
  "containerDefinitions": [
    {
      "name": "sleep",
      "image": "busybox",
      "cpu": 10,
      "command": [
        "sleep",
        "360"
      ],
      "memory": 10,
      "essential": true
    }
  ],
  "family": "sleep360"
}
```

The above example JSON can be passed to the AWS CLI in two ways: you can save the task definition JSON as a file and pass it with the `--cli-input-json file://`*`path_to_file.json`* option, or you can escape the quotation marks in the JSON and pass the JSON container definitions on the command line as in the below example. If you choose to pass the container definitions on the command line, your command additionally requires a `--family` parameter that is used to keep multiple versions of your task definition associated with each other.

To use a JSON file for container definitions:

```
$ aws ecs register-task-definition --cli-input-json file://$HOME/tasks/
sleep360.json
```

To use a JSON string for container definitions:

```
$ aws ecs register-task-definition --family sleep360 --container-definitions
 "[{\"name\":\"sleep\",\"image\":\"busybox\",\"cpu\":10,\"command\":[\"sleep
\",\"360\"],\"memory\":10,\"essential\":true}]"
```

The **register-task-definition** returns a description of the task definition after it completes its registration.

```
{
    "taskDefinition": {
        "volumes": [],
        "taskDefinitionArn": "arn:aws:ec2:us-east-1:aws_account_id:task-
definition/sleep360:1",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "sleep",
                "mountPoints": [],
                "image": "busybox",
                "cpu": 10,
                "portMappings": [],
                "command": [
                    "sleep",
                    "360"
                ],
                "memory": 10,
                "essential": true,
                "volumesFrom": []
            }
        ],
        "family": "sleep360",
        "revision": 1
    }
}
```

Step 6: List Task Definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **run-task** or **start-task**.

```
$ aws ecs list-task-definitions
{
    "taskDefinitionArns": [
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep300:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep300:2",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]
}
```

Step 7: Run a Task

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can run the registered task in your cluster. For this example, you place a single instance of the sleep360:1 task definition in your default cluster.

```
$ aws ecs run-task --cluster default --task-definition sleep360:1 --count 1
{
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
            "overrides": {
                "containerOverrides": [
                    {
                        "name": "sleep"
                    }
                ]
            },
            "lastStatus": "PENDING",
            "containerInstanceArn": "arn:aws:ecs:us-
east-1:aws_account_id:container-instance/container_instance_ID",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/
default",
            "desiredStatus": "RUNNING",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-
definition/sleep360:1",
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-
east-1:aws_account_id:container/container_ID",
                    "taskArn": "arn:aws:ecs:us-
east-1:aws_account_id:task/task_ID",
                    "lastStatus": "PENDING",
                    "name": "sleep"
                }
            ]
        }
    ]
}
```

Step 8: List Tasks

List the tasks for your cluster. You should see the task that you ran in the previous section. You can take the task ID or the full ARN that is returned from this command and use it to describe the task later.

```
$ aws ecs list-tasks --cluster default
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
    ]
}
```

Step 9: Describe the Running Task

Describe the task using the task ID retrieved earlier to get more information about the task.

```
$ aws ecs describe-tasks --cluster default --task task_ID
{
    "failures": [],
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
            "overrides": {
                "containerOverrides": [
                    {
                        "name": "sleep"
                    }
                ]
            },
            "lastStatus": "RUNNING",
            "containerInstanceArn": "arn:aws:ecs:us-
east-1:aws_account_id:container-instance/container_instance_ID",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/
default",
            "desiredStatus": "RUNNING",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-
definition/sleep360:1",
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-
east-1:aws_account_id:container/container_ID",
                    "taskArn": "arn:aws:ecs:us-
east-1:aws_account_id:task/task_ID",
                    "lastStatus": "RUNNING",
                    "name": "sleep",
                    "networkBindings": []
                }
            ]
        }
    ]
}
```

Amazon ECS Service Limits

The following table provides the default limits for Amazon ECS for an AWS account which can be changed. For more information, see AWS Service Limits in the *Amazon Web Services General Reference*.

Resource	Default Limit
Number of clusters per region, per account	1000
Number of container instances per cluster	1000
Number of services per cluster	500

The following table provides other limitations for Amazon ECS that cannot be changed.

Resource	Default Limit
Number of load balancers per service	1
Number of tasks per service (the desired count)	1000
Number of tasks launched (count) per **run-task**	10
Number of container instances per **start-task**	10
Throttle on container instance registration rate	1 per second / 60 max per minute
Task definition size limit	32 KiB
Task definition max containers	10
Throttle on task definition registration rate	1 per second / 60 max per minute

Logging Amazon ECS API Calls By Using AWS CloudTrail

Amazon ECS is integrated with AWS CloudTrail, a service that captures API calls made by or on behalf of Amazon ECS in your AWS account and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the Amazon ECS console or from the Amazon ECS API. Using the information collected by CloudTrail, you can determine what request was made to Amazon ECS, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon ECS Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to Amazon ECS actions are tracked in log files. Amazon ECS records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

All of the Amazon ECS actions are logged and are documented in the Amazon EC2 Container Service API Reference. For example, calls to the **CreateService**, **RunTask**, and **RegisterContainerInstance** actions generate entries in the CloudTrail log files.

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the **userIdentity** field in the CloudTrail Event Reference.

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 life cycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see Configuring Amazon SNS Notifications.

You can also aggregate Amazon ECS log files from multiple AWS regions and multiple AWS accounts into a single S3 bucket. For more information, see Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket.

Understanding Amazon ECS Log File Entries

CloudTrail log files can contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

Amazon ECS Troubleshooting

You may need to troubleshoot issues with your load balancers, tasks, services, or container instances. This chapter helps you find diagnostic information from the Amazon ECS container agent, the Docker daemon on the container instance, and the service event log in the Amazon ECS console.

Topics

Checking Stopped Tasks for Errors

If you have trouble starting a task (for example, you run the task and the task displays a `PENDING` status and then disappears) your task might be stopping because of an error. You can view errors like this in the Amazon ECS console by displaying the stopped task and inspecting it for error messages.

To check stopped tasks for errors

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, choose the cluster in which your stopped task resides.
3. On the **Cluster :** *clustername* page, choose the **Tasks** tab to view your tasks.
4. In the **Desired task status** table header, choose **Stopped** to view stopped tasks, and then choose the stopped task you want to inspect. The most recent stopped tasks are listed first.
5. In the **Details** section, inspect the **Stopped reason** field to see the reason the task was stopped.

Details

Cluster	default
Container Instance	dd3599e9-2ca6-40f4-9da5-a0bb10408260
EC2 instance id	i-83c6ab47
Task Definition	curler:4
Last status	STOPPED
Desired status	STOPPED
Created at	2015-11-20 13:31:01 -0800
Stopped at	2015-11-20 13:31:03 -0800
Stopped reason	Essential container in task exited

Some possible reasons and their explanations are listed below:

Task failed ELB health checks in (elb elb-name)
> The current task failed the ELB health check for the load balancer that is associated with the task's service. For more information, see Troubleshooting Service Load Balancers (p. 239).

Scaling activity initiated by (deployment deployment-id)
> When you reduce the desired count of a stable service, some tasks need to be stopped in order to reach the desired number. Tasks that are stopped by downscaling services have this stopped reason.

Host EC2 (instance *id*) stopped/terminated
> If you stop or terminate a container instance with running tasks, then the tasks are given this stopped reason.

Container instance deregistration forced by user
> If you force the deregistration of a container instance with running tasks, then the tasks are given this stopped reason.

Essential container in task exited
> Containers marked as `essential` in task definitions cause a task to stop if they exit or die. When an essential container exiting is the cause of a stopped task, the Step 6 (p. 236) can provide more diagnostic information as to why the container stopped.

6. If you have a container that has stopped, expand the container and inspect the **Status reason** row to see what caused the task state to change.

Containers

Name	Container Id	Status
▼ curler	3f871451-c9f1-4d6f-a…	STOPPED (CannotPull(

Details

Status reason CannotPullContainerError: Error: image tutum/bogus:late

Command ["/usr/bin/watch","curl","-v","http://amazon-ecs-200477

In the previous example, the container image name cannot be found. This can happen if you misspell the image name.

If this inspection does not provide enough information, you can connect to the container instance with SSH and inspect the Docker container locally. For more information, see Inspect Docker Containers (p. 246).

Service Event Messages

If you are troubleshooting a problem with a service, the first place you should check for diagnostic information is the service event log.

To check the service event log in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.

2. On the **Clusters** page, choose the cluster in which your service resides.

3. On the **Cluster :** *clustername* page, choose the service that you would like to inspect.

4. On the **Service :** *servicename* page, choose the **Events** tab.

5. Examine the **Message** column for errors or other helpful information.

(service *service-name*) was unable to place a task because the resources could not be found.

In the above image, this service could not find the available resources to add another task. The possible causes for this are:

Not enough ports
> If your task uses fixed host port mapping (for example, your task uses port 80 on the host for a web server), you must have at least one container instance per task, because only one container can use a single host port at a time. You should add container instances to your cluster or reduce your number of desired tasks.

Not enough memory
> If your task definition specifies 1000 MiB of memory, and the container instances in your cluster each have 1024 MiB of memory, you can only run one copy of this task per container instance. You can experiment with less memory in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Not enough CPU
> A container instance has 1,024 CPU units for every CPU core. If your task definition specifies 1,000 CPU units, and the container instances in your cluster each have 1,024 CPU units, you can only run one copy of this task per container instance. You can experiment with less CPU units in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Container instance missing required attribute
> Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the `ECS_AVAILABLE_LOGGING_DRIVERS` agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you do not have any available container instances that can satisfy this requirement, the task cannot be placed. For more information on which attributes are required for specific task definition parameters and agent configuration variables, see Task Definition Parameters (p. 88) and Amazon ECS Container Agent Configuration (p. 69).

(service *service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* encountered error "AGENT".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs; for more information, see Amazon ECS Container Agent Log (p. 242). You should also verify that the agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following commands:

```
[ec2-user ~]$ sudo stop ecs
ecs stop/waiting
[ec2-user ~]$ sudo start ecs
ecs start/running, process 26119
```

(service *service-name*) (instance *instance-id*) is unhealthy in (elb *elb-name*) due to (reason Instance has failed at least the UnhealthyThreshold number of health checks consecutively.)

This service is registered with a load balancer and the load balancer health checks are failing. For more information, see Troubleshooting Service Load Balancers (p. 239).

Troubleshooting Service Load Balancers

Amazon ECS services can register tasks with an Elastic Load Balancing load balancer. Load balancer configuration errors are common causes for stopped tasks. If your stopped tasks were started by services that use a load balancer, consider the following possible causes.

Improper IAM permissions for the `ecsServiceRole` IAM role

The `ecsServiceRole` allows Amazon ECS services to register container instances with Elastic Load Balancing load balancers. You must have the proper permissions set for this role. For more information, see Amazon ECS Service Scheduler IAM Role (p. 182).

Container instance security group

If your container is mapped to port 80 on your container instance, your container instance security group must allow inbound traffic on port 80 for the load balancer health checks to pass.

Elastic Load Balancing load balancer not configured for all Availability Zones

Your load balancer should be configured to use all of the Availability Zones in a region, or at least all of the Availability Zones in which your container instances reside. If a service uses a load balancer and starts a task on a container instance that resides in an Availability Zone that the load balancer is not configured to use, the task never passes the health check and it is killed.

Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that do not exist. If a container instance is determined to be unhealthy, it is removed from the load balancer. Be sure to verify that the following parameters are configured correctly for your service load balancer.

Ping Port

The **Ping Port** value for a load balancer health check is the port on the container instances that the load balancer checks to determine if it is healthy. If this port is misconfigured, the load balancer will likely deregister your container instance from itself. This port should be configured to use the `hostPort` value for the container in your service's task definition that you are using with the health check.

Ping Path

This value is often set to `index.html`, but if your service does not respond to that request, then the health check fails. If your container does not have an `index.html` file, you can set this to `/` to target the base URL for the container instance.

Response Timeout

This is the amount of time that your container has to return a response to the health check ping. If this value is lower than the amount of time required for a response, the health check fails.

Health Check Interval

This is the amount of time between health check pings. The shorter your health check intervals are, the faster your container instance can reach the **Unhealthy Threshold**.

Unhealthy Threshold

This is the number of times your health check can fail before your container instance is considered unhealthy. If you have an unhealthy threshold of 2, and a health check interval of 30 seconds, then your task has 60 seconds to respond to the health check ping before it is assumed unhealthy. You can raise the unhealthy threshold or the health check interval to give your tasks more time to respond.

Unable to update the service *servicename*: Load balancer container name or port changed in task definition

If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service.

Enabling Docker Debug Output

If you are having trouble with Docker containers or images, you can enable debug mode on your Docker daemon. Enabling debugging provides more verbose output from the daemon and you can use this information to find out more about why your containers or images are having issues.

Enabling Docker debug mode can be especially useful in retrieving error messages that are sent from container registries, such as Amazon ECR, and, in many circumstances, enabling debug mode is the only way to see these error messages.

To enable debug mode on your Docker daemon

1. Connect to your container instance. For more information, see .
2. Open the Docker options file with a text editor, such as **vi**.
   - For Amazon Linux, Red Hat Enterprise Server, and CentOS, the Docker options file is at `/etc/sysconfig/docker`.
   - For Ubuntu and Debian, the Docker options file is at `/etc/default/docker`.
3. Find the Docker options statement and add the `-D` option to the string, inside the quotes.

   Note
   If the Docker options statement begins with a `#`, you need to remove that character to uncomment the statement and enable the options.
   - For Amazon Linux, Red Hat Enterprise Server, and CentOS, the Docker options statement is called `OPTIONS`. For example:

   ```
   # Additional startup options for the Docker daemon, for example:
   # OPTIONS="--ip-forward=true --iptables=true"
   # By default we limit the number of open files per container
   OPTIONS="-D --default-ulimit nofile=1024:4096"
   ```

   - For Ubuntu and Debian, the Docker options statement is called `DOCKER_OPTS`. For example:

   ```
   # Use DOCKER_OPTS to modify the daemon startup options.
   DOCKER_OPTS="-D --dns 8.8.8.8 --dns 8.8.4.4"
   ```

4. Save the file and exit your text editor.
5. Restart the Docker daemon.

   ```
   $ sudo service docker restart
   Stopping docker:                                    [  OK  ]
   Starting docker: .                                  [  OK  ]
   ```

Your Docker logs should now show more verbose output. For example:

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response
 from server: \"{\\\"errors\\\":[{\\\"code\\\":\\\"DENIED\\\",\\\"message\
\\":\\\"User: arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not
 authorized to perform: ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-
east-1:1111:repository/nginx_test\\\"}]}\\n\" http.Header{\"Connection\":
[]string{\"keep-alive\"}, \"Content-Type\":[]string{\"application/json;
 charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015 21:48:21 GMT\"},
 \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"}, \"Content-
Length\":[]string{\"235\"}}"
```

Amazon ECS Log File Locations

Amazon ECS stores logs in the `/var/log/ecs` folder of your container instances. There are logs available from the Amazon ECS container agent and the `ecs-init` service that controls the state of the agent (start/stop) on the container instance. You can view these log files by connecting to a container instance using SSH. For more information, see .

> **Note**
> If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector. For more information, see .

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs at `/var/log/ecs/ecs-agent.log.`*`timestamp`*.

> **Note**
> You can increase the verbosity of the container agent logs by setting `ECS_LOGLEVEL=debug` and restarting the container agent. For more information, see .

```
[ec2-user ~]$ cat /var/log/ecs/ecs-agent.log.2016-08-15-15
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0
 (895f3c1)
2016-08-15T15:54:41Z [INFO] Loading configuration
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be
 overridden to 3h0m0s, parsed value 0, minimum threshold 1m0s
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load
 state
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20
 1.21 1.22]
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

Amazon ECS `ecs-init` Log

The `ecs-init` process stores logs at `/var/log/ecs/ecs-init.log.`*`timestamp`*.

```
[ec2-user ~]$ cat /var/log/ecs/ecs-init.log.2015-04-22-20
2015-04-22T20:51:45Z [INFO] pre-start
2015-04-22T20:51:45Z [INFO] Loading Amazon EC2 Container Service Agent into
 Docker
2015-04-22T20:51:46Z [INFO] start
2015-04-22T20:51:46Z [INFO] No existing agent container to remove.
2015-04-22T20:51:46Z [INFO] Starting Amazon EC2 Container Service Agent
```

IAM Roles for Tasks Credential Audit Log

When the IAM roles for tasks credential provider is used to provide credentials to tasks, these requests are logged in `/var/log/ecs/audit.log.`*`YYYY-MM-DD-HH`*. The log entry format is as follows:

• Timestamp
• HTTP response code

- IP address and port number of request origin
- Relative URI of the credential provider
- The user agent that made the request
- The task ARN that the requesting container belongs to
- The `GetCredentials` API name and version number
- The Amazon ECS cluster name that the container instance is registered to
- The container instance ARN

An example log entry is shown below:

```
[ec2-user ~]$ cat /var/log/ecs/audit.log.2016-07-13-16
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-
requests/2.7.0 CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN
 GetCredentials 1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Amazon ECS Logs Collector

If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector, which is available on GitHub. The script collects general operating system logs as well as Docker and Amazon ECS container agent logs, which can be helpful for troubleshooting AWS Support cases, and then it compresses and archives the collected information into a single file that can easily be shared for diagnostic purposes. It also supports enabling debug mode for the Docker daemon and the Amazon ECS container agent on Amazon Linux variants, such as the Amazon ECS-optimized AMI. Currently, the Amazon ECS logs collector supports the following operating systems:

- Amazon Linux
- Red Hat Enterprise Linux 7
- Debian 8

Note
The source code for the Amazon ECS logs collector is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently provide support for running modified copies of this software.

To download and run the Amazon ECS logs collector

1. Connect to your container instance. For more information, see Connect to your Container Instance (p. 42).

2. Download the Amazon ECS logs collector script.

   ```
   [ec2-user ~]$ curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-
   collector/master/ecs-logs-collector.sh
   ```

3. Run the script to collect the logs and create the archive.

   Note
   To enable debug mode for the Docker daemon and the Amazon ECS container agent, add the `--mode=debug` option to the command below.

   ```
   [ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
   ```

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

Agent Introspection Diagnostics

The Amazon ECS agent introspection API can provide helpful diagnostic information. For example, you can use the agent introspection API to get the Docker ID for a container in your task. You can use the agent introspection API by connecting to a container instance using SSH. For more information, see .

The below example shows two tasks, one that is currently running and one that was stopped.

> **Note**
> The command below is piped through the **python -mjson.tool** for greater readability.

```
[ec2-user ~]$ curl http://localhost:51678/v1/tasks | python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1095  100  1095    0     0   117k      0 --:--:-- --:--:-- --:--:--
 133k
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-
west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-a8d14064fd24",
            "Containers": [
                {
                    "DockerId":
 "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
                    "DockerName": "ecs-console-sample-app-static-6-busybox-
ce83ce978a87a890ab01",
                    "Name": "busybox"
                }
            ],
            "Family": "console-sample-app-static",
            "KnownStatus": "STOPPED",
            "Version": "6"
        },
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-
eaea-4da9-a638-097bea534740",
            "Containers": [
                {
                    "DockerId":
 "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-f0e5859699a7aecfb101",
                    "Name": "simple-app"
                },
```

```
            {
                "DockerId":
 "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                "Name": "busybox"
            }
        ],
        "DesiredStatus": "RUNNING",
        "Family": "console-sample-app-static",
        "KnownStatus": "RUNNING",
        "Version": "6"
    }
    ]
}
```

In the above example, the stopped task (*090eff9b-1ce3-4db6-848a-a8d14064fd24*) has two containers. You can use **docker inspect** *container-ID* to view detailed information on each container. For more information, see Amazon ECS Container Agent Introspection (p. 78).

Docker Diagnostics

Docker provides several diagnostic tools that can help you troubleshoot problems with your containers and tasks. For more information about all of the available Docker command line utilities, go to the Docker Command Line topic in the Docker documentation. You can access the Docker command line utilities by connecting to a container instance using SSH. For more information, see Connect to your Container Instance (p. 42).

The exit codes that Docker containers report can also provide some diagnostic information (for example, exit code 137 means that the container received a SIGKILL signal). For more information, see Exit Status in the Docker documentation.

List Docker Containers

You can use the **docker ps** command on your container instance to list the running containers. In the below example, only the Amazon ECS container agent is running. For more information, go to docker ps in the Docker documentation.

```
[ec2-user ~]$ docker ps
CONTAINER ID        IMAGE                           COMMAND
 CREATED              STATUS            PORTS                      NAMES
cee0d6986de0        amazon/amazon-ecs-agent:latest    "/agent"            22
 hours ago         Up 22 hours         127.0.0.1:51678->51678/tcp   ecs-agent
```

You can use the **docker ps -a** command to see all containers (even stopped or killed containers). This is helpful for listing containers that are unexpectedly stopping. In the following example, container f7f1f8a7a245 exited 9 seconds ago, so it would not show up in a **docker ps** output without the -a flag.

```
[ec2-user ~]$ docker ps -a
CONTAINER ID        IMAGE                                    COMMAND
          CREATED              STATUS                    PORTS
            NAMES
db4d48e411b1        amazon/ecs-emptyvolume-base:autogenerated    "not-
applicable"        19 seconds ago
```

```
                              ecs-console-sample-app-static-6-internalecs-emptyvolume-
source-c09288a6b0cba8a53700
f7f1f8a7a245         busybox:buildroot-2014.02                      "\"sh -c '/
bin/sh -c    22 hours ago        Exited (137) 9 seconds ago
              ecs-console-sample-app-static-6-busybox-ce83ce978a87a890ab01
189a8ff4b5f0         httpd:2                                        "httpd-
foreground"     22 hours ago        Exited (137) 40 seconds ago
                          ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600
0c7dca9321e3         amazon/ecs-emptyvolume-base:autogenerated    "not-
applicable"         22 hours ago
                      ecs-console-sample-app-static-6-internalecs-emptyvolume-
source-90fefaa68498a8a80700
cee0d6986de0         amazon/amazon-ecs-agent:latest                "/agent"
        22 hours ago        Up 22 hours                       127.0.0.1:51678-
>51678/tcp    ecs-agent
```

View Docker Logs

You can view the STDOUT and STDERR streams for a container with the **docker logs** command. In this example, the logs are displayed for the *dc7240fe892a* container and piped through the **head** command for brevity. For more information, go to docker logs in the Docker documentation.

```
[ec2-user ~]$ docker logs dc7240fe892a | head
AH00558: httpd: Could not reliably determine the server's fully qualified
 domain name, using 172.17.0.11. Set the 'ServerName' directive globally to
 suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified
 domain name, using 172.17.0.11. Set the 'ServerName' directive globally to
 suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid
 140327115417472] AH00489: Apache/2.4.12 (Unix) configured -- resuming normal
 operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472]
 AH00094: Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken
 pipe"
```

Inspect Docker Containers

If you have the Docker ID of a container, you can inspect it with the **docker inspect** command. Inspecting containers provides the most detailed view of the environment in which a container was launched. For more information, go to docker inspect in the Docker documentation.

```
[ec2-user ~]$ docker inspect dc7240fe892a
[{
```

```
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/local/apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/
httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
        },
        "Hostname": "dc7240fe892a",
...
```

API `failures` Error Messages

In some cases, an API call that you have triggered through the Amazon ECS console or the AWS CLI exits with a `failures` error message. The following possible API `failures` error messages are explained below for each API call. The failures occur on a particular resource, and the resource in parentheses is the resource associated with the failure.

Many resources are region-specific, so make sure the console is set to the correct region for your resources, or that your AWS CLI commands are being sent to the correct region with the `--region` *region* option.

- `DescribeClusters`

  `MISSING` (cluster ID)
  > Your cluster was not found. The cluster name may not have been spelled correctly or the wrong region may be specified.

- `DescribeInstances`

  `MISSING` (container instance ID)
  > The container instance you are attempting to describe does not exist. Perhaps the wrong cluster or region has been specified, or the container instance ARN or ID is misspelled.

- `DescribeServices`

  `MISSING` (service ID)
  > The service you are attempting to describe does not exist. Perhaps the wrong cluster or region has been specified, or the container instance ARN or ID is misspelled.

- `DescribeTasks`

  `MISSING` (task ID)
  > The task you are trying to describe does not exist. Perhaps the wrong cluster or region has been specified, or the task ARN or ID is misspelled.

- RunTask or StartTask

  RESOURCE:* (container instance ID)

  The resource or resources requested by the task are unavailable on the given container
  instance. If the resource is CPU or memory, you may need to add container instances to your
  cluster.

  AGENT (container instance ID)

  The container instance that you attempted to launch a task onto has an agent which is currently
  disconnected. In order to prevent extended wait times for task placement, the request was
  rejected.

  ATTRIBUTE (container instance ID)

  Your task definition contains a parameter that requires a specific container instance attribute
  that is not available on your container instances. For more information on which attributes are
  required for specific task definition parameters and agent configuration variables, see Task
  Definition Parameters (p. 88) and Amazon ECS Container Agent Configuration (p. 69).

- StartTask

  MISSING (container instance ID)

  The container instance you attempted to launch the task onto does not exist. Perhaps the wrong
  cluster or region has been specified, or the container instance ARN or ID is misspelled.

  INACTIVE (container instance ID)

  The container instance that you attempted to launch a task onto was previously deregistered
  with Amazon ECS and cannot be used.

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.