Amazon EC2 Container Service

Developer Guide

API Version 2014-11-13

Amazon EC2 Container Service: Developer Guide

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 EC2 instances. Amazon ECS lets you launch and stop container-enabled 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.

Components of Amazon ECS

Amazon ECS contains the following components:

Cluster
A logical grouping of container instances that you can place tasks on.

Container instance
An Amazon EC2 instance that is running the Amazon ECS agent and has been registered into a cluster. For more information, see Amazon ECS Container Instances (p. 23).

Task definition
A description of an application that contains one or more container definitions. For more information, see Amazon ECS Task Definitions (p. 48).

Scheduler
The method used for placing tasks on container instances. For more information about the different scheduling options available in Amazon ECS, see Scheduling Amazon ECS Tasks (p. 65).

Service
An Amazon ECS service allows you to run and maintain a specified number of instances of a task definition simultaneously. For more information, see Services (p. 66).

Task
An instantiation of a task definition that is running on a container instance.

Container
A Linux container that was created as part of a task.

How to Get Started with Amazon ECS

To use Amazon ECS, you need to be set up to launch Amazon EC2 instances into your clusters. You can also optionally install the AWS Command Line Interface to use Amazon ECS. For more information, see Setting Up with Amazon ECS (p. 3).

After you are set up, you are ready to complete the Getting Started with Amazon ECS (p. 17) tutorial.

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. To use the AWS CLI with Amazon ECS , you must use a version of the AWS CLI that supports the latest Amazon ECS features (version `1.7.21` or greater).

> **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. 3)
2. Create an IAM User (p. 4)
3. Create an IAM Role for your Container Instances and Services (p. 5)
4. Create a Key Pair (p. 5)
5. Create a Virtual Private Cloud (p. 7)
6. Create a Security Group (p. 8)
7. Install the AWS CLI (p. 9)

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 click **Sign Up**.
2. Follow the on-screen 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 the Administrators group

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, click **Groups**, and then click **Create New Group**.
3. In the **Group Name** box, type **`Administrators`**, and then click **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** box to filter the list of policies.
5. Click **Next Step**, and then click **Create Group**.

Your new group is listed under **Group Name**.

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

1. In the navigation pane, click **Users**, and then click **Create New Users**.
2. In box **1**, type a user name. Clear the check box next to **Generate an access key for each user**. Then click **Create**.
3. In the list of users, click the name (not the check box) of the user you just created. You can use the **Search** box to search for the user name.
4. In the **Groups** section, click **Add User to Groups**.
5. Select the check box next to the **Administrators** group. Then click **Add to Groups**.
6. Scroll down to the **Security Credentials** section. Under **Sign-In Credentials**, click **Manage Password**.
7. Select **Assign a custom password**. Then type a password in the **Password** and **Confirm Password** boxes. When you are finished, click **Apply**.

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 the next section. 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. 94) and Amazon ECS Service Scheduler IAM Role (p. 96) 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 West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
Asia Pacific (Tokyo)	ap-northeast-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*.

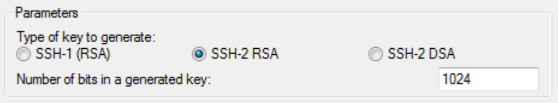
To connect to your instance using your key pair

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**.



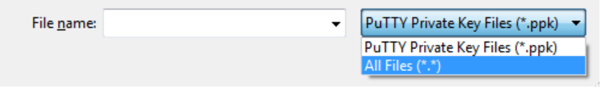
Parameters

Type of key to generate:

○ SSH-1 (RSA) ● SSH-2 RSA ○ SSH-2 DSA

Number of bits in a generated key: 1024

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.


File name:
PuTTY Private Key Files (*.ppk)
PuTTY Private Key Files (*.ppk)
All Files (*.*)

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.

Create a Virtual Private Cloud

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. We strongly suggest that you launch your container instances in a VPC.

> **Note**
> The Amazon ECS console first run experience creates a VPC for your cluster, so if you intend to use the Amazon ECS console, you can move ahead to the next section.

If you have a default VPC, you also can skip this section and move to the next task, Create a Security Group (p. 8). To determine whether you have a default VPC, see Supported Platforms in the Amazon EC2 Console in the *Amazon EC2 User Guide for Linux Instances*. Otherwise, you can create a nondefault VPC in your account using the steps below.

> **Important**
> If your account supports EC2-Classic in a region, then you do not have a default VPC in that region.

To create a nondefault VPC

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. From the navigation bar, select a region for the VPC. VPCs are specific to a region, so you should select the same region in which you created your key pair.
3. On the VPC dashboard, choose **Start VPC Wizard**.
4. On the **Step 1: Select a VPC Configuration** page, ensure that **VPC with a Single Public Subnet** is selected, and choose **Select**.
5. On the **Step 2: VPC with a Single Public Subnet** page, enter a friendly name for your VPC in the **VPC name** field. Leave the other default configuration settings, and choose **Create VPC**. On the confirmation page, choose **OK**.

For more information about Amazon VPC, see What is Amazon VPC? in the *Amazon VPC User Guide*.

Create a Security Group

Security groups act as a firewall for associated container instances, controlling both inbound and outbound traffic at the container instance level. You can add rules to a security group that enable you to connect to your container instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP and HTTPS access from anywhere. Add any rules to open ports that are required by your tasks.

> **Note**
> The Amazon ECS console first run experience creates a security group for your instances and load balancer based on the task definition you use, so if you intend to use the Amazon ECS console, you can move ahead to the next section.

Note that if you plan to launch container instances in multiple regions, you need to create a security group in each region. For more information about regions, see Regions and Availability Zones in the *Amazon EC2 User Guide for Linux Instances*.

> **Tip**
> You need the public IP address of your local computer, which you can get using a service. For example, we provide the following service: http://checkip.amazonaws.com/. To locate another service that provides your IP address, use the search phrase "what is my IP address." If you are connecting through an Internet service provider (ISP) or from behind a firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

To create a security group with least privilege

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for the security group. Security groups are specific to a region, so you should select the same region in which you created your key pair.

   Note
   Amazon ECS is available in the following regions:

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

3. Choose **Security Groups** in the navigation pane.
4. Choose **Create Security Group**.
5. Enter a name for the new security group and a description. Choose a name that is easy for you to remember, such as your IAM user name, followed by _SG_, plus the region name. For example, *me*_SG_*useast1*.
6. In the **VPC** list, ensure that your default VPC is selected; it's marked with an asterisk (*).

   Note
   If your account supports EC2-Classic, select the VPC that you created in the previous task.

7. Amazon ECS container instances do not require any inbound ports to be open. However, you might want to add an SSH rule so you can log into the container instance and examine the tasks with Docker commands. You can also add rules for HTTP and HTTPS if you want your container instance to host a task that runs a web server. Complete the following steps to add these optional security group rules.

   On the **Inbound** tab, create the following rules (choose **Add Rule** for each new rule), and then choose **Create**:

   - Choose **HTTP** from the **Type** list, and make sure that **Source** is set to **Anywhere** (`0.0.0.0/0`).
   - Choose **HTTPS** from the **Type** list, and make sure that **Source** is set to **Anywhere** (`0.0.0.0/0`).
   - Choose **SSH** from the **Type** list. In the **Source** field, ensure that **Custom IP** is selected, and specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix `/32`. For example, if your IP address is `203.0.113.25`, specify `203.0.113.25/32`. If your company allocates addresses from a range, specify the entire range, such as `203.0.113.0/24`.

     Caution

     For security reasons, we don't recommend that you allow SSH access from all IP addresses (`0.0.0.0/0`) to your instance, except for testing purposes and only for a short time.

Install the AWS CLI

To use the AWS CLI with Amazon ECS, install the AWS CLI, version `1.7.21` or greater. If the AWS CLI is installed on your system, you can check the version with the following command:

```
$ aws --version
aws-cli/1.7.21 Python/2.7.8 Darwin/14.0.0
```

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. For more information, see Docker Containers on AWS. 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.

If you'd like to try out Docker before you install it, go to the interactive tutorial on the Docker website.

Topics

Installing Docker

Docker is available on many different operating systems, including most modern Linux distributions, like Ubuntu, and even Mac OSX and Windows (by using **boot2docker**). 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. Amazon ECS requires a minimum Docker version of 1.5.0 (version 1.6.2 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/installation/.

```
[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. 44) on your ECS container instances to use your private images in task definitions.

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, 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 our 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 what image you want for your 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 our 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 our Dockerfile. Substitute *`my-dockerhub-username`* with your Docker Hub user name.

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

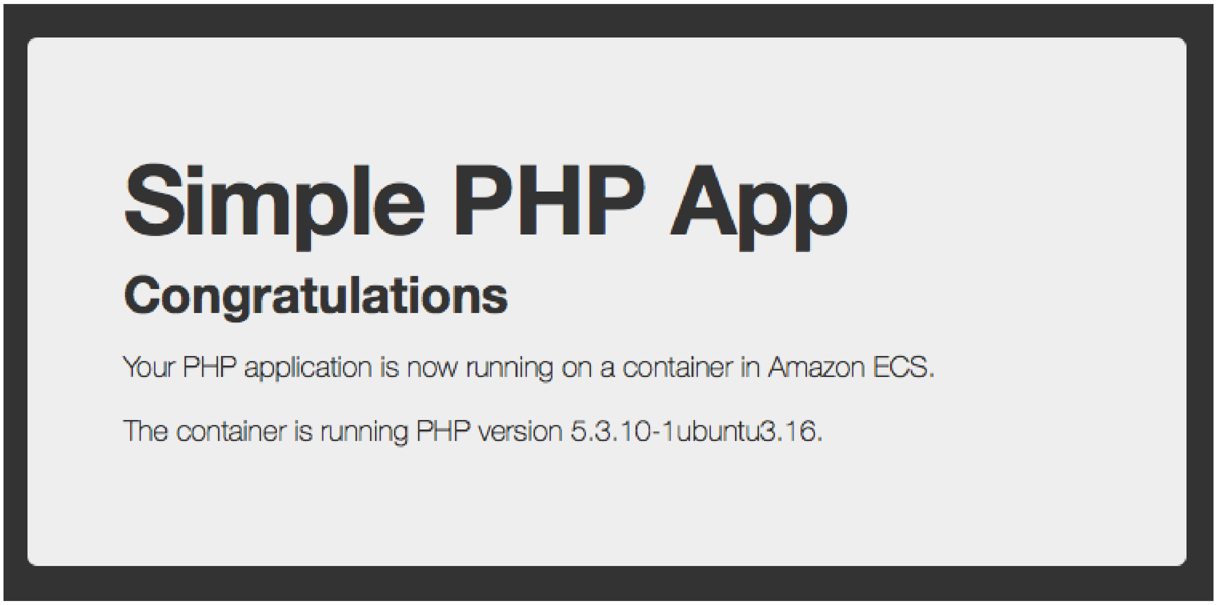
```
[ec2-user ecs-demo-php-simple-app]$ docker run -p 80:80 my-dockerhub-user
name/amazon-ecs-sample
```

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 on a Linux computer, point your browser to http://localhost/.
   - If you are using **boot2docker** on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the **boot2docker ip** command.

   ```
   $ boot2docker ip
   192.168.59.103
   ```

   You should see a web page running the simple PHP app.



Simple PHP App

Congratulations

Your PHP application is now running on a container in Amazon ECS.

The container is running PHP version 5.3.10-1ubuntu3.16.

8. Stop the Docker container by typing **Ctrl**+**c**.
9. (Optional) Upload the Docker image to your Docker Hub account.

   a. Log in to your Docker Hub account.

   ```
   [ec2-user ecs-demo-php-simple-app]$ docker login
   ```

   b. Verify that you have logged in correctly.

   ```
   [ec2-user ecs-demo-php-simple-app]$ docker info
   ```

   You should see "`Username: my-dockerhub-username`" in the output. If not, verify your Docker Hub login information and try to log in again.

   c. Push the image.

   ```
   [ec2-user ecs-demo-php-simple-app]$ docker push my-dockerhub-user
   name/amazon-ecs-sample
   ```

   Note
   If you receive an error stating "`FATA[0012] Error pushing to registry: Authentication is required`", verify that you are using a repository that you have permission to push to. The **docker images** command lists which images are available locally; you should see an image that begins with your Docker Hub user name. If not, return to Step 4 (p. 13) to rebuild the image with the proper repository name and then retry the **docker push** command.

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. 3) and Getting Started with Amazon ECS (p. 17).

- 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 con
sole-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, configuring a cluster in the Amazon ECS console.

> **Important**
> Before you begin be sure that you've completed the steps in Setting Up with Amazon ECS (p. 3).

Step 1: Welcome to Amazon ECS

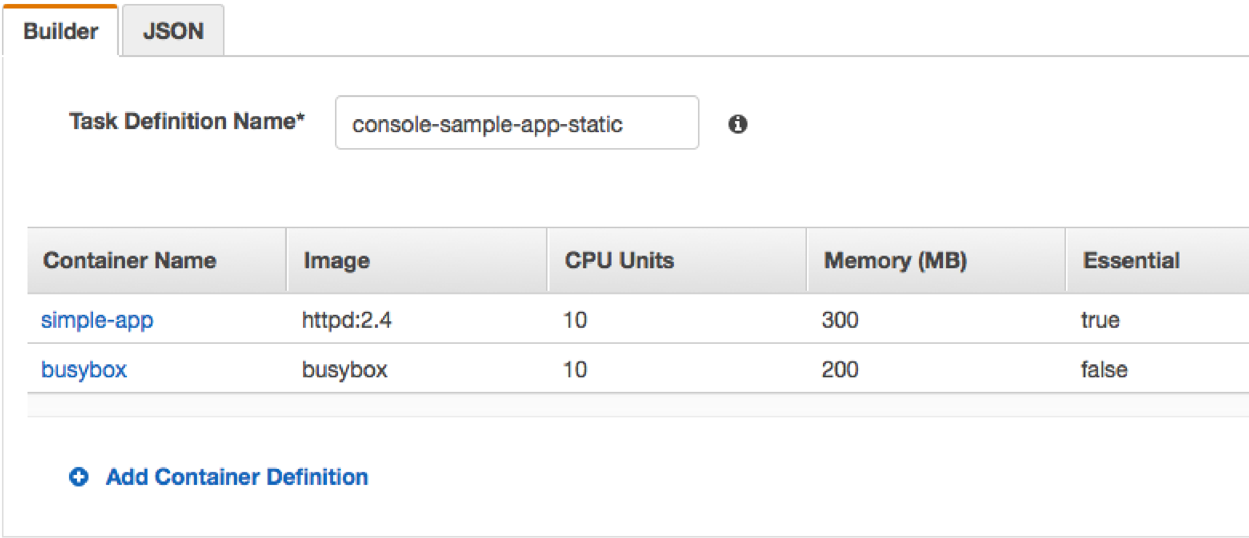
The Amazon ECS first run wizard will guide you through the process to get started with Amazon ECS. The wizard gives you the option of creating a cluster and launching our sample web application, or if you already have a Docker image you would like to launch in Amazon ECS, you can create a task definition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

1. Open the Amazon ECS console first run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.
2. Choose whether you would like to use the **Amazon ECS sample** task definition or a **Custom** task definition that you create yourself and choose **Next Step**. If you don't have a specific Docker image you want to launch into a cluster, you should pick the sample task definition to see what one looks like.

Step 2: 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.



Task Definition Name*: console-sample-app-static

Container Name	Image	CPU Units	Memory (MB)	Essential
simple-app	httpd:2.4	10	300	true
busybox	busybox	10	200	false

If you chose to use the **Amazon ECS sample** task definition, you can see the containers defined, `simple-app`, and `busybox`. You can optionally rename the task definition or review and edit the resources used by each container (such as CPU units and memory) by clicking the container name and editing the values shown. For more information on what each of these task definition parameters does, see Task Definition Parameters (p. 52).

2. (Optional) You can also add containers to your task definition. Click **Add Container Definition**, fill out the required parameters, and click **Add**.
3. When you are finished examining and editing the task definition, click **Next Step**.

Step 3: Schedule tasks

In this section of the wizard, you select how you would like to schedule the tasks from your task definition. You can choose to **Run Tasks Once**, which is ideal for batch jobs that perform work and then stop, or you can choose to **Create a service** to launch and maintain 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 we should run this as a service so it will restart if the task becomes unhealthy or unexpectedly stops.

1. Choose **Create a Service** to launch and maintain your task.
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. In the **Service Name** field, select a name for your service.
4. (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. Click the **Container: Port** menu and select **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. 67).

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

Step 4: Configure Cluster

In this section of the wizard, you 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 **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 task in it that exposes container ports on to container instance ports, such as the **Amazon ECS sample** application, you need to select at least that many instances here.
2. Select 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. 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 have a key pair, you can create one in the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
4. (Optional) In the **Security Group** section, you can choose a CIDR block that restricts access to your instances. The default value allows access from the entire Internet.
5. In the **IAM Role Information** section, choose an existing Amazon ECS container instance (`ecsInstanceRole`) and service (`ecsServiceRole`) role that you have already created, or click **Create IAM Roles** to create the required IAM roles for your container instances and services.
6. Click **Allow** in the authorization window to allow Amazon ECS to make calls on your behalf.
7. Click **Review and Launch** to proceed.

Step 5: Review

- Review your task definition, task configuration, and cluster configurations and click **Launch Instance & Run Service** to finish.

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. 73) 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 ser
vice_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. 74) 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 ser
vice_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. 30) 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.

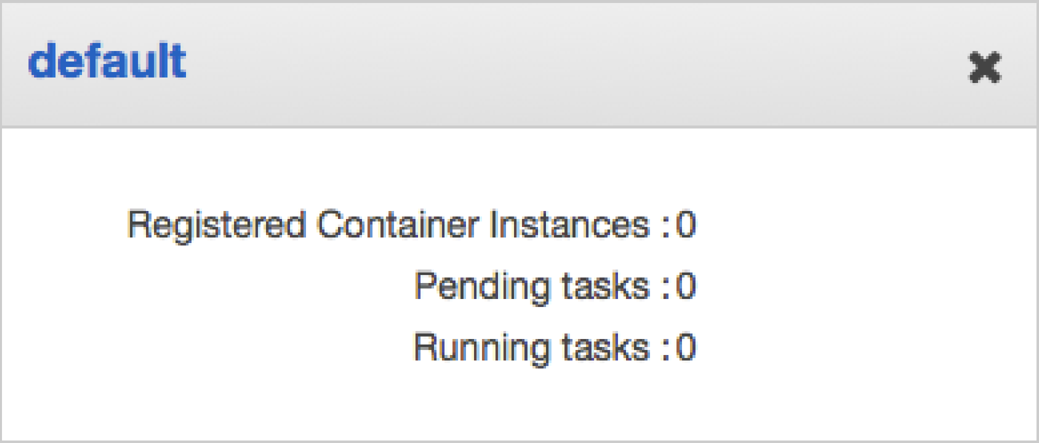
```
$ aws --region us-west-2 ecs deregister-container-instance --cluster default -
-container-instance container_instance_id --force
```

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.



default ✖

Registered Container Instances : 0

Pending tasks : 0

Running tasks : 0

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 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. 32).
- 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. 94).
- Containers associated with your tasks can map their network ports to ports on the host Amazon ECS container instance so they are reachable from the Internet. If your container has external connectivity, then your container instance security group must allow inbound access to the ports you want to expose. For more information, see Create a Security Group (p. 8).
- 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 are running in a private VPC, they need a network address translation (NAT) instance to provide this access. For more information, see NAT Instances in the *Amazon VPC User 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.

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. 35).

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 AWS Management Console and open the 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. 94) to create the role.

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. 3). 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 West (N. California)	us-west-1
US West (Oregon)	us-west-2
EU (Ireland)	eu-west-1
Asia Pacific (Tokyo)	ap-northeast-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. 32).

   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-2015.03.f-amazon-ecs-optimized** AMI. The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI Name	AMI ID
`us-east-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-b540eade`
`us-west-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-5721df13`
`us-west-2`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-cb584dfb`

Region	AMI Name	AMI ID
eu-west-1	**amzn-ami-2015.03.f-amazon-ecs-optimized**	ami-2aaef35d
ap-northeast-1	**amzn-ami-2015.03.f-amazon-ecs-optimized**	ami-8aa61c8a
ap-southeast-2	**amzn-ami-2015.03.f-amazon-ecs-optimized**	ami-5ddc9f67

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, verify that your **Auto-assign Public IP** field is set to **Enable** so that your instance is accessible from the Internet.
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. 3).

   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. 94).

10. (Optional) Configure your Amazon ECS container instance with user data, such as the agent environment variables from Amazon ECS Container Agent Configuration (p. 40).

    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 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. 42).

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

11. Choose **Review and Launch**.
12. 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. 3) 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**.

13. On the **Review Instance Launch** page, choose **Launch**.
14. 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**.
15. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
16. 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**.)

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. 50).
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. 94).

   a. Open the 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. 94) 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. 25), but in Step 10 (p. 26), 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.

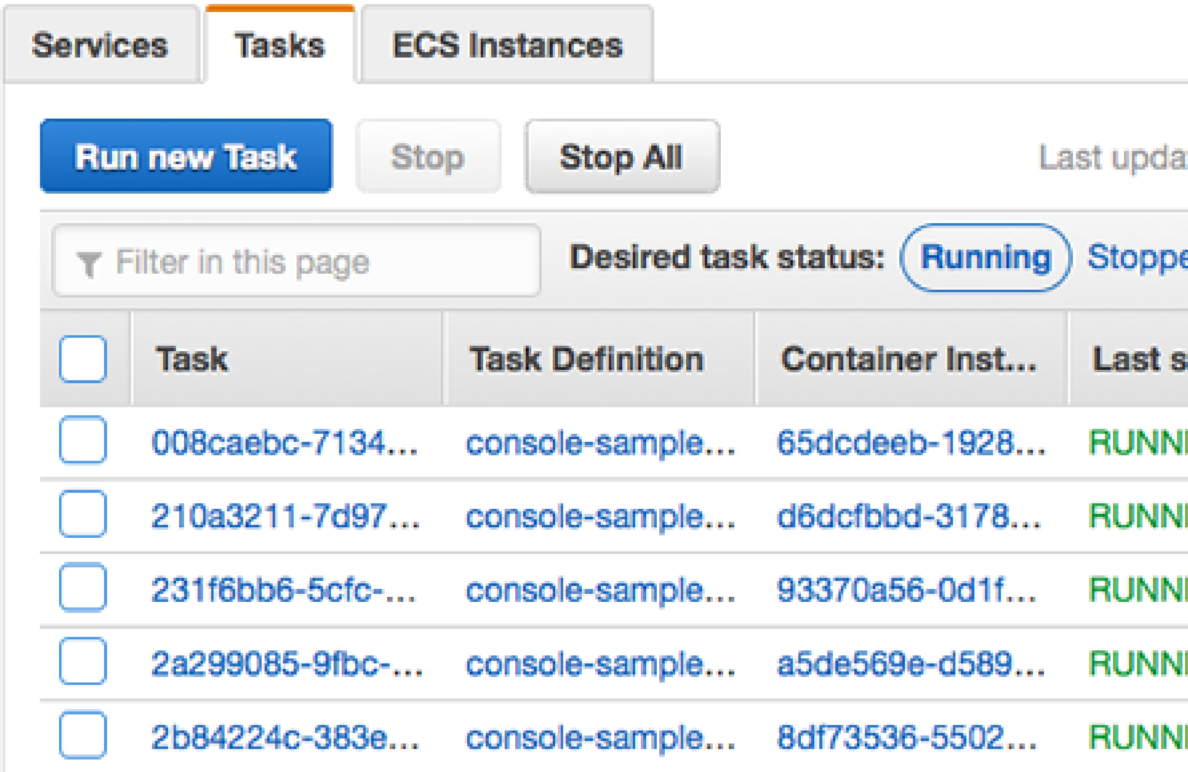
Note

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 '. | .Contain
erInstanceArn' | 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.



□	Task	Task Definition	Container Inst...	Last s
□	008caebc-7134...	console-sample...	65dcdeeb-1928...	RUNNI
□	210a3211-7d97...	console-sample...	d6dcfbbd-3178...	RUNNI
□	231f6bb6-5cfc-...	console-sample...	93370a56-0d1f...	RUNNI
□	2a299085-9fbc-...	console-sample...	a5de569e-d589...	RUNNI
□	2b84224c-383e...	console-sample...	8df73536-5502...	RUNNI

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 and they will continue to pass Elastic Load Balancing load balancer health checks 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 will start another copy of that task on a different container instance if possible.

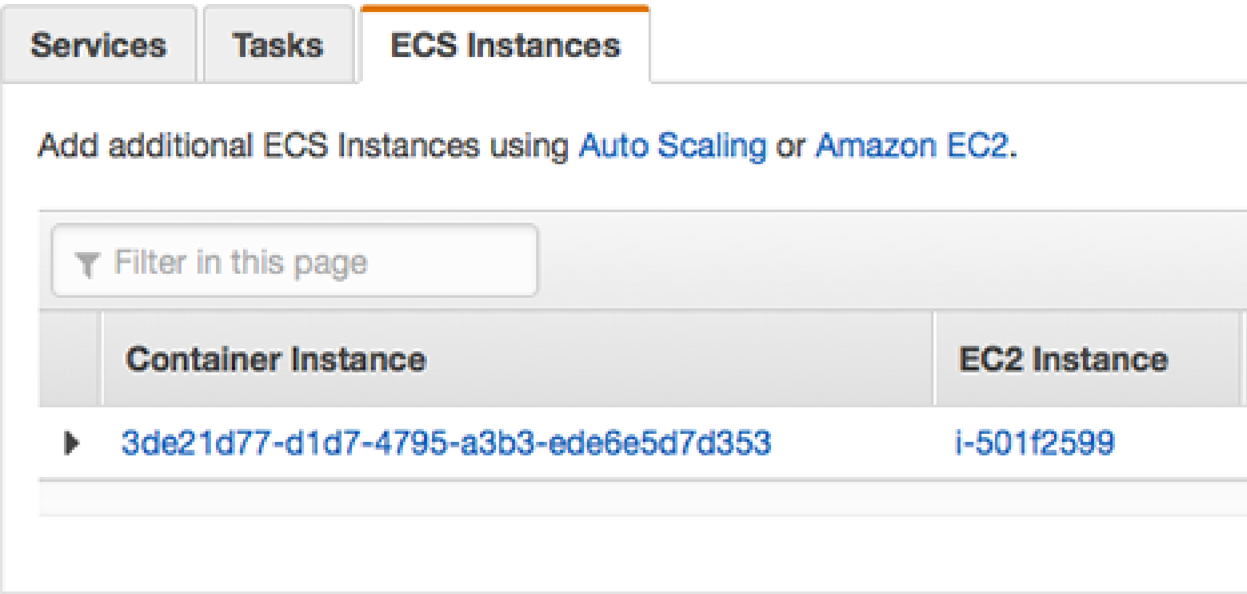
If you intend to use the container instance for some other purpose after deregistration, you should stop all of the tasks running on the container instance before deregistration to avoid any orphaned tasks from consuming resources.

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
When you terminate a container instance, it is automatically deregistered from your cluster.

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.



Add additional ECS Instances using Auto Scaling or Amazon EC2.

	Container Instance	EC2 Instance
▶	3de21d77-d1d7-4795-a3b3-ede6e5d7d353	i-501f2599

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. 94).
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. 46).

```
[ec2-user ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.3.0 (097e4af)"
}
```

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. 94).
2. Connect to your instance.
3. Install Docker on your instance. Amazon ECS requires a minimum Docker version of 1.5.0 (version 1.6.2 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/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/installation/.

5. Pull and run the latest Amazon ECS container agent on your container instance. 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.
   - The `cgroup` volume mount should use the path to the `cgroup` virtual file system for the host and container paths. For Amazon Linux, this path is `/cgroup`; for many other operating systems, this path is `/sys/fs/cgroup`, but you should verify the path in your specific OS documentation.
   - The `execdriver` volume mount host path should use your container instance's OS-specific `execdriver` path. For most operating systems, this path is `/var/run/docker/execdriver/native`, but you should verify the path in your specific OS documentation.

   For more information on these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 40).

```
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 \
--volume=/sys/fs/cgroup:/sys/fs/cgroup:ro \
--volume=/var/run/docker/execdriver/native:/var/lib/docker/execdriver/nat
ive:ro \
--publish=127.0.0.1:51678:51678 \
--env=ECS_LOGFILE=/log/ecs-agent.log \
--env=ECS_LOGLEVEL=info \
--env=ECS_DATADIR=/data \
--env=ECS_CLUSTER=cluster_name \
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 has a minimum Docker version requirement, and each 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. 35).

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. 25).

Amazon ECS container agent version	Minimum Docker version	Supported
`1.4.0`	`1.5.0`	Yes
`1.3.1`	`1.5.0`	Yes
`1.3.0`	`1.5.0`	Yes
`1.2.1`	`1.5.0`	Yes
`1.2.0`	`1.5.0`	Yes
`1.1.0`	`1.5.0`	Yes
`1.0.0`	`1.3.3`	Yes
`0.0.3`	`1.3.3`	No

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 packaged with each Amazon ECS-optimized AMI.

Amazon ECS-optimized AMI	Amazon ECS container agent version	Docker version	`ecs-init` version	Supported
2015.03.f	`1.4.0`	`1.6.2`	`1.4.0`	Yes
2015.03.e	`1.3.1`	`1.6.2`	`1.3.1-1`	Yes
2015.03.d	`1.2.1`	`1.6.2`	`1.2.0-2`	Yes
2015.03.c	`1.2.0`	`1.6.2`	`1.2.0-1`	Yes
2015.03.b	`1.1.0`	`1.6.0`	`1.0-3`	Yes
2015.03.a	`1.0.0`	`1.5.0`	`1.0-1`	Yes
preview3	`0.0.3`	`1.3.3`	`0.3-0`	No

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, choose the cluster that hosts the container instance or instances you would like to check.
3. On the **Cluster : *cluster_name*** page, choose the **ECS Instances** tab.
4. Note the **Agent version** column for your container instances.

   If your agent version is 1.4.0, you are running the latest container agent. If your agent version is below 1.4.0, you can update your container agent with the following procedures.

   > **Important**
   > To update the Amazon ECS agent version from versions prior to v1.0.0 on your Amazon ECS-optimized AMI it is recommended 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. 25).

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. 46).

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 - v1.4.0 (4ab1051)"
}
```

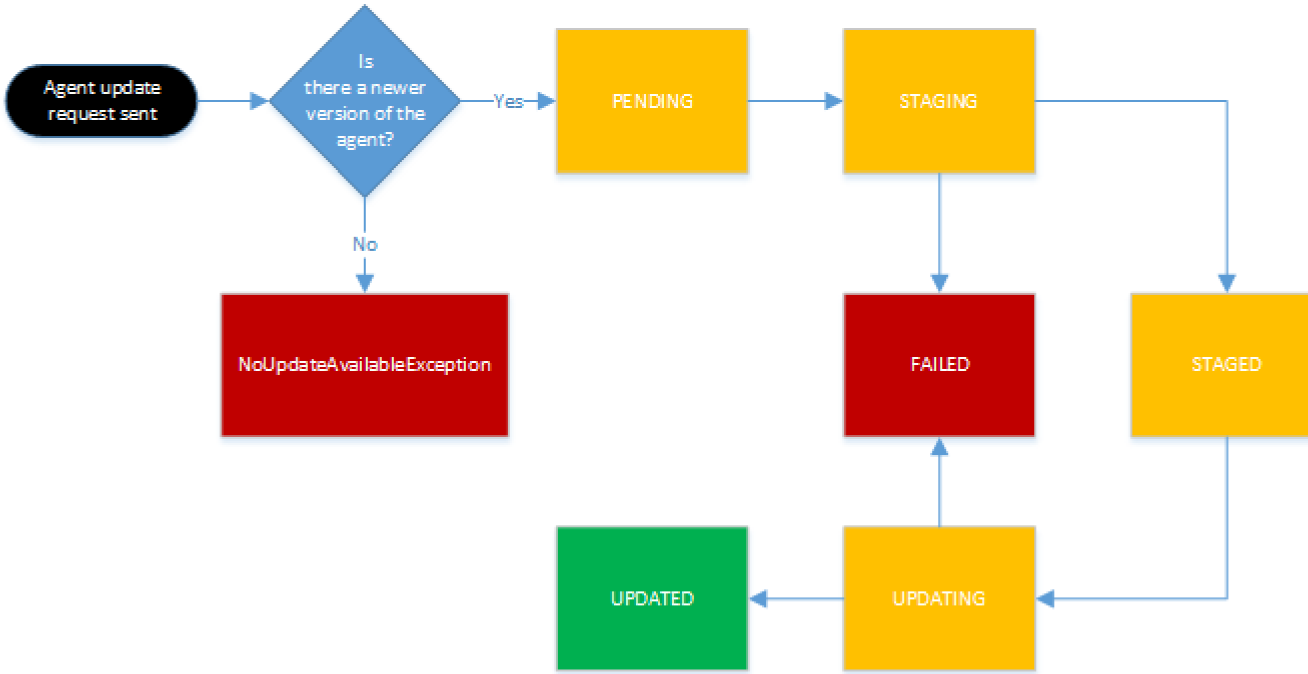
> **Note**
> The introspection API added `Version` information in the version v1.0.0 of the Amazon ECS container agent. If `Version` is not present when querying the introspection API, or the introspection API is not present in your agent at all, then the version you are running is v0.0.3 or earlier, 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 can update the container agent with a one-click operation in the Amazon ECS console or a simple AWS CLI command. You can also update the agent by updating the `ecs-init` package on the container instance. For more information, see To update the `ecs-init` package on the Amazon ECS-optimized AMI (p. 38).

> **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. 38).
> To update the Amazon ECS agent version from versions prior to v1.0.0 on your Amazon ECS-optimized AMI it is recommended 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. 25).

The update process begins when you request an agent update. 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.


Agent update request sent
Is there a newer version of the agent?
Yes
No
PENDING
STAGING
NoUpdateAvailableException
FAILED
STAGED
UPDATED
UPDATING

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, choose the cluster that hosts the container instance or instances you would like to check.
3. On the **Cluster :** ***cluster_name*** page, choose the **ECS Instances** tab.
4. Choose the container instance you want to update by clicking its ID.
5. On the Container Instance page, choose the **Update agent** button to update the container agent on the instance.

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
```

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

1. Log into your container instance via SSH.
2. Update the `ecs-init` package with the following command.

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

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. 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
```

6. Run the latest Amazon ECS container agent on your container instance. 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.
   - The `cgroup` volume mount should use the path to the `cgroup` virtual file system for the host and container paths. For Amazon Linux, this path is `/cgroup`; for many other operating systems, this path is `/sys/fs/cgroup`, but you should verify the path in your specific OS documentation.
   - The `execdriver` volume mount host path should use your container instance's OS-specific `execdriver` path. For most operating systems, this path is `/var/run/docker/execdriver/native`, but you should verify the path in your specific OS documentation.

   For more information on these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 40).

```
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 \
--volume=/sys/fs/cgroup:/sys/fs/cgroup:ro \
--volume=/var/run/docker/execdriver/native:/var/lib/docker/execdriver/nat
ive:ro \
--publish=127.0.0.1:51678:51678 \
--env=ECS_LOGFILE=/log/ecs-agent.log \
--env=ECS_LOGLEVEL=info \
--env=ECS_DATADIR=/data \
--env=ECS_CLUSTER=cluster_name \
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 the restart the agent.

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`*.

Topics

- Available Parameters (p. 40)
- Storing Container Instance Configuration in Amazon S3 (p. 42)

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. 44).	Null

Environment Key	Example Values	Description	Default Value

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 .dockercfg file 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 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.	warn
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

Environment Key	Example Values	Description	Default Value
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
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

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 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. 94).
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. 40) using the following format. This example configures private registry authentication. For more information, see Private Registry Authentication (p. 44).

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":{"au
th":"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. 25). In Step 10 (p. 26), 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
```

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 must receive these environment variables as options to the **docker run** command that starts the container agent, with the syntax `-e` *`VARIABLE_NAME`*`=`*`VARIABLE_VALUE`*.

> **Important**
> Do not inject these authentication environment variables at instance launch time with Amazon EC2 user data. This method is 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. 42).

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. After you create the file, you can get the authentication information with the following command.

```
$ cat ~/.dockercfg
{"https://index.docker.io/v1/":{"au
th":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

In this example, the following environment variables should be set for the Amazon ECS container agent.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":{"au
th":"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 set for the Amazon ECS container agent.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":{"username":"my_name","pass
word":"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 ~]$ 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/":{"user
   name":"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. 35).

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. 46).

```
[ec2-user ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.3.0 (097e4af)"
}
```

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": "arn:aws:ecs:us-east-1:<aws_account_id>:container-
instance/example5-58ff-46c9-ae05-543f8example","Version":"Amazon ECS Agent -
v1.0.0 (4023248)"
}
```

To view information on 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": "9581a69a761a557fb
fce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "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 -notrunc** command on the container instance.

```
[ec2-user ~]$ curl http://localhost:51678/v1/tasks?dockerid=9581a69a761a557fb
fce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1
{
  "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": "9581a69a761a557fb
fce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
      "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
      "Name": "mysql"
    },
    {
      "DockerId":
"bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
      "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
      "Name": "wordpress"
    }
  ]
}
```

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
- 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

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. 52).

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. 48).

Topics

Application Architecture

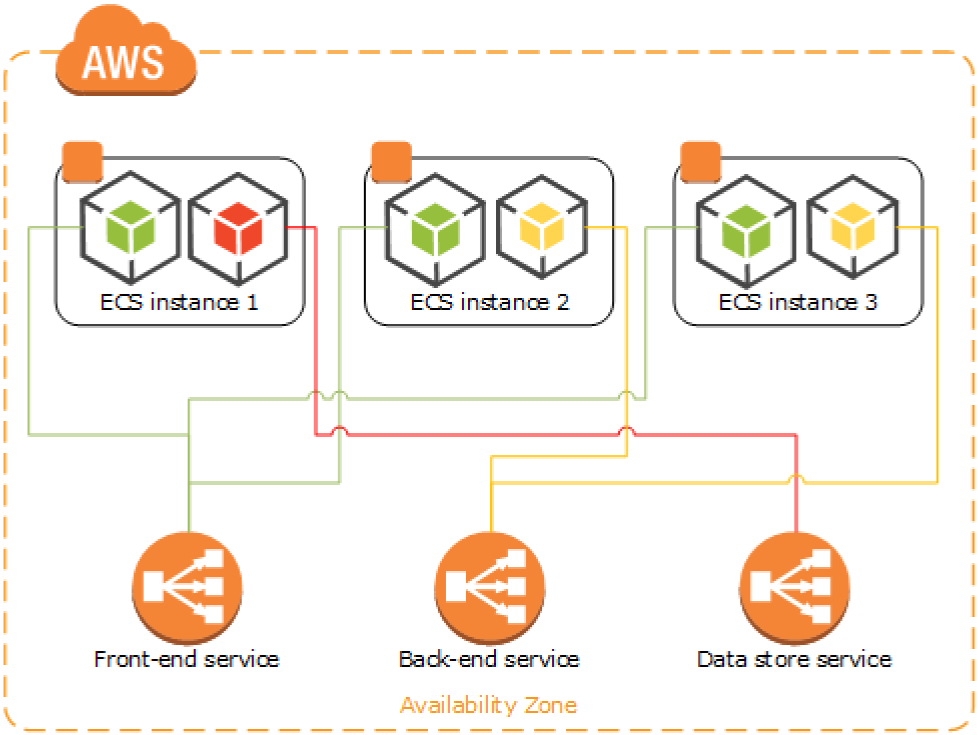
When you’re considering how to model task definitions and services, it helps to think about what processes need to run together on the same instance and how you will scale each component. As an example, imagine an application that consists of the following components:

- 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. 72). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see Service Load Balancing (p. 67). 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. 73).

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, choose the **JSON** tab and paste your task definition JSON into the text area.
   b. Choose the **Builder** tab.
   c. Verify your information and select **Create**.

6. On the **Create a task definition** page, choose the **Builder** tab.
7. In the **Task definition name** field, enter a name for your task definition.
8. 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. For more information, see Task Definition Parameters (p. 52).
   c. Select **Add** to add your container to the task definition.

9. (Optional) To define data volumes for your task, select the **JSON** tab and paste the volume definitions into the `volumes` section of the task definition JSON object. For more information, see Using Data Volumes in Tasks (p. 58).
10. 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. 52).

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "cpu": 0,
            "memory": 0,
            "links": [
                ""
            ],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": ""
                }
            ],
            "essential": true,
            "entryPoint": [
                ""
            ],
            "command": [
                ""
            ],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ]
        }
    ],
    "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
```

Task Definition Parameters

Task definitions are split into three basic parts: the task family, container definitions, and volumes. The family is the name of the task, and each family can have multiple revisions. 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 volumes are optional.

Family

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.

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:

`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.

`image`

Type: string

Required: yes

The image to use for 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`*.

`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.

> **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.

`memory`

Type: integer

Required: yes

The number of MiB of memory to reserve for the container. If your container attempts to exceed the memory allocated here, the container is killed.

`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. For more information about linking Docker containers, go to https://docs.docker.com/userguide/dockerlinks/.

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", ...]
```

`portMappings`

Type: object array

Required: no

Port mappings allow containers to access ports on the host container instance to send or receive traffic.

`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, or you can omit the `hostPort` (or set it to `0`) while specifying a `containerPort` and your container will automatically receive a port 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 50 reserved ports at a time, including the default reserved ports (automatically assigned ports do not count toward this 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`).

`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. 35).

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
    }
    ...
]
```

`essential`

Type: Boolean

Required: no

If the `essential` parameter of a container is marked as `true`, the failure of that container stops the task. 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.

> **Note**
> All tasks must have at least one essential container.

```
"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 `ENTRYPOINT` that is passed to the container. For more information about the Docker `ENTRYPOINT` parameter, go to https://docs.docker.com/reference/builder/#entrypoint.

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

`command`

Type: string array

Required: no

The `CMD` that is passed to the container. For more information about the Docker `CMD` parameter, go to https://docs.docker.com/reference/builder/#cmd.

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

`environment`

Type: object array

Required: no

The environment variables to pass to a container.

`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" }
]
```

`mountPoints`

Type: object array

Required: no

The mount points for data volumes in your container.

`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.

`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
                }
              ]
```

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. The following parameters are allowed in a container definition:

`name`

Type: string

Required: yes

The name of the volume. This name is referenced in the `sourceVolume` parameter of container definition `mountPoints`.

`host`

Type: string

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.

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 Docker garbage collection service eventually deletes it. 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.

Example Task Definitions

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/.

```
{
  "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"
}
```

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. 66).

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. 74).

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 (p. 74)
- Task Life Cycle (p. 75)

Services

Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously. This is called a service. You can optionally run your service behind a load balancer. If any of your tasks should fail or stop, or if your underlying container instance becomes unhealthy, Amazon ECS launches another instance of your task definition to replace it.

Service Concepts

- If a task in a service becomes unhealthy or unresponsive, 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. 67).

Topics

- Service Definition Parameters (p. 66)
- Service Load Balancing (p. 67)
- Creating a Service (p. 72)
- Updating a Service (p. 73)
- Deleting a Service (p. 74)

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": [
        {
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "role": ""
}
```

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 Amazon Resource Name (ARN) of the task definition that you want to run in your service.

`loadBalancers`
: A list of load balancer objects to use with your service. Currently you are limited to one load balancer per 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.

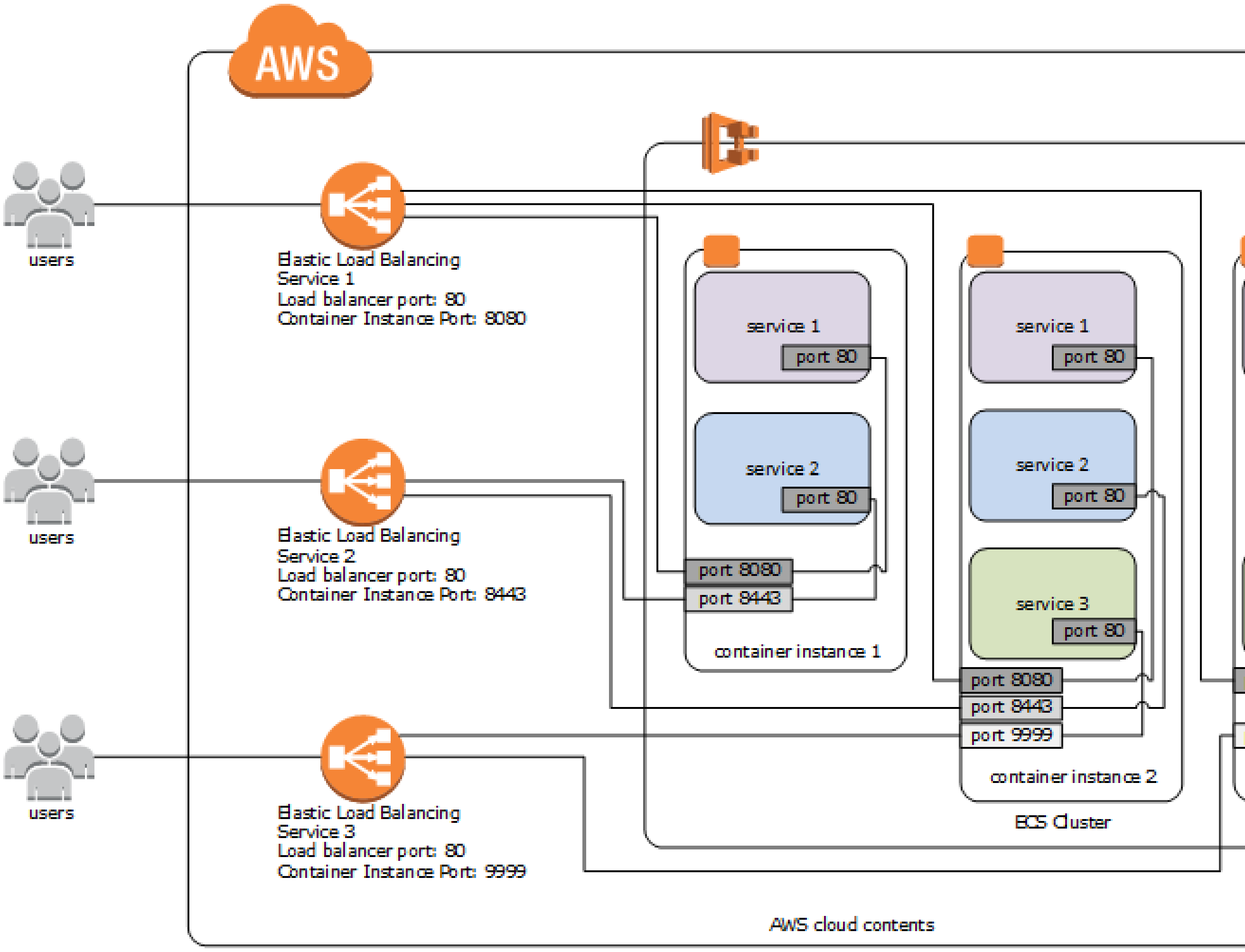
`desiredCount`
: The number of instantiations of the specified task definition to place and keep running on your cluster.

`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 your Amazon ECS container agent 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.

Service Load Balancing

Your Amazon ECS service can optionally be configured to use Elastic Load Balancing to manage traffic.



Load Balancing Concepts

- Elastic Load Balancing currently supports 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.
- All of the containers that are launched in a single task definition are always placed on the same container instance. You may choose to put two different containers 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 Listener Configurations in the *Elastic Load Balancing Developer Guide*.
- There is a limit of one load balancer per service.
- 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.

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 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 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. 96) to create the role. If the role does exist, select the role to view the attached policies.
4. 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**.

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.

Note that you can create your 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.

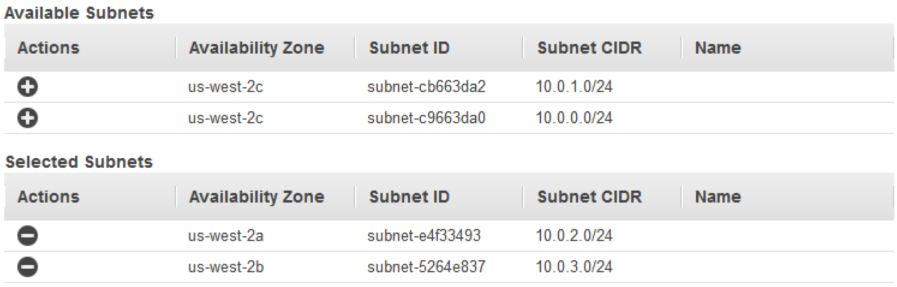
To define your load balancer

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 **NETWORK & SECURITY**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. In **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.

6. From **Create LB inside**, select the same network that your container instances are located in: EC2-Classic or a specific VPC.
7. 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 Listener Configurations in the *Elastic Load Balancing Developer Guide*.
8. [EC2-VPC] To improve the availability of your load balancer, select at least two subnets in different Availability Zones. Your selected subnets must at least include any subnet 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 select **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	

9. 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

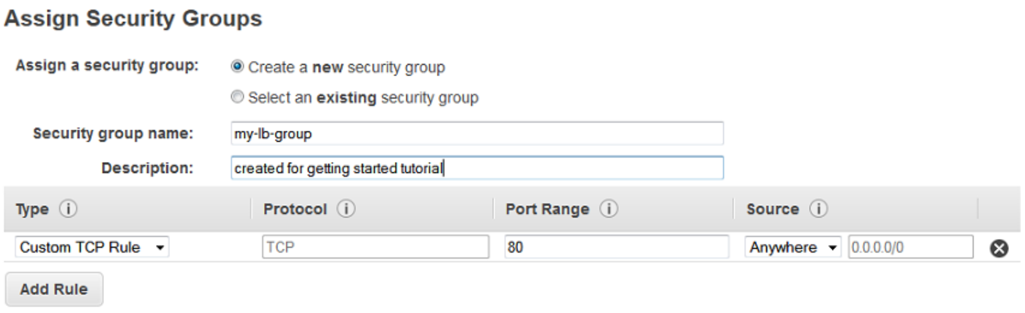
If you created your load balancer in a VPC, you must assign it a security group that allows inbound traffic to the ports that you specified for your load balancer and the health checks for your load balancer.

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.



Assign Security Groups

Assign a security group: ● Create a **new** security group ○ Select an **existing** security group

Security group name: my-lb-group

Description: created for getting started tutorial

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	80	Anywhere 0.0.0.0/0

3. Choose **Next: Configure Security Settings** to go to the next page in the wizard.

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

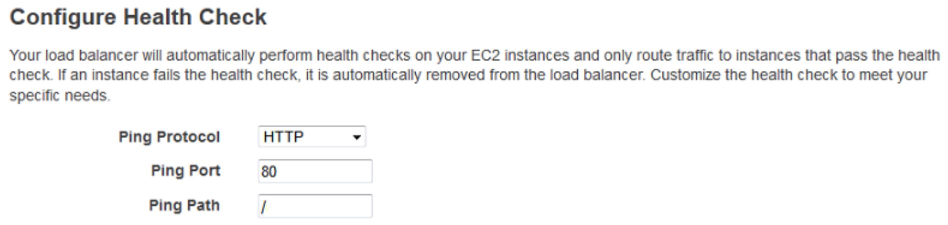
1. On the **Configure Health Check** page, do the following:

   a. Leave **Ping Protocol** set to its default value of `HTTP`.

   b. Leave **Ping Port** set to its default value of `80`.

   c. In the **Ping Path** field, 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, under **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 Load Balancer in the *Elastic Load Balancing Developer 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, click **Create Tag** and specify a key and a value for the tag.
3. After you are finished adding tags, click **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**. 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. 72).

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.

To create a service

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 name of the task definition you would like to create your service from.
5. On the **Task Definition name** page, choose the revision of the task definition you would like to create your service from.
6. Review the task definition, and choose **Create Service**.
7. On the **Create Service** page, enter a unique name for your service in the **Service name** field.
8. In the **Number of tasks** field, enter the number of tasks you would like to launch and maintain on your cluster.

> **Note**
> If your tasks expose specified ports on container instances, then you need at least one container instance with the specified port available in your cluster for each task in your service.

9. (Optional) If you have an available Elastic Load Balancing load balancer configured, you can attach it to your service with the following steps. If you want to configure a new load balancer, see Creating a Load Balancer (p. 69).

   a. choose **Add ELB**.
   b. In the **Add a load balancer** window, configure the following settings as necessary and choose **Add**.

      - **Load Balancer**: Select the load balancer to use with your service.
      - **Container Name**: Select the name of the container to use with the load balancer.
      - **Container Port**: Select the port on the container to direct load balancer traffic to. 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.

10. On the **Create Service** page, in the **Service Role** section, choose **Manage IAM Role** to allow Amazon ECS to register and deregister container instances from the load balancer as tasks are placed on them.
11. Choose **Allow** to authorize the service role for your load balancer.
12. Review your information and choose **Create Service**.

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 you 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, one task at a time. The service scheduler creates a task with the new task definition (provided there is an available container instance to place it on), and after it reaches the `RUNNING` state, a task that is using the old task definition is drained and stopped. This process continues until all of the desired tasks in your service are using the new task definition.

When the service scheduler replaces a task during an update, the equivalent of **docker stop** is issued to the containers running in the task. This results in a `SIGTERM` and a 30-second timeout, after which `SIGKILL` is sent and the containers are forcibly stopped. If the container handles the `SIGTERM` gracefully and exits within 30 seconds from receiving it, no `SIGKILL` is sent.

To update a running service

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, choose the name of the cluster that your service resides in.
5. On the **Cluster :** ***`name`*** page, choose the **Services** tab.
6. Check the box to the left of the service you want to update and choose **Update**.

7. On the **Update Service** page, your service information is pre-populated. Change the task definition or number of desired tasks (or both) and choose **Update 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. 73).

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. From the navigation bar, select the region that your cluster is in.
3. In the navigation pane, select **Clusters**.
4. On the **Clusters** page, choose the name of the cluster that your service resides in.
5. On the **Cluster : *name*** page, choose the **Services** tab.
6. Check the box to the left of the service you want to update and choose **Delete**.

   Note
   Your service must have zero desired or running tasks to delete it.
7. Choose **Yes, Delete** to confirm your service deletion.

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. 66).

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. Enter the number of tasks to launch with this task definition in the **Number of tasks** field.
8. (Optional) To send command or environment variable overrides to one or more containers in your task definition, complete the following steps:

a. Choose the **Advanced Options** menu.

b. On the **Container Overrides** menu, select a container to which to send a command or environment variable override.

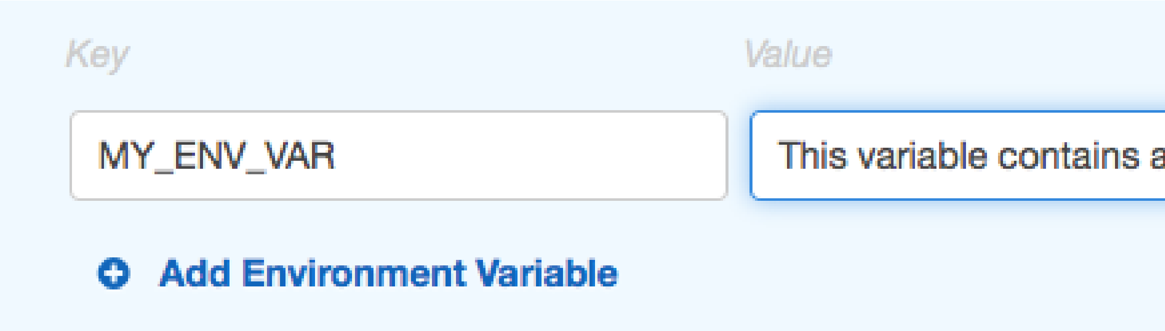
- **For a command override:** In the **Command override** field, type the command override to send. If your container definition does not specify an ENTRYPOINT, the format should be a comma-separated list of non-quoted strings. For example:

  ```
  /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**. In the **Key** field, enter the name of your environment variable. In the **Value** field, enter the string your environment value should be set to (without surrounding quotes).



MY_ENV_VAR

This variable contains a

  The above environment variable override is sent to the container as:

  ```
  MY_ENV_VAR="This variable contains a string."
  ```

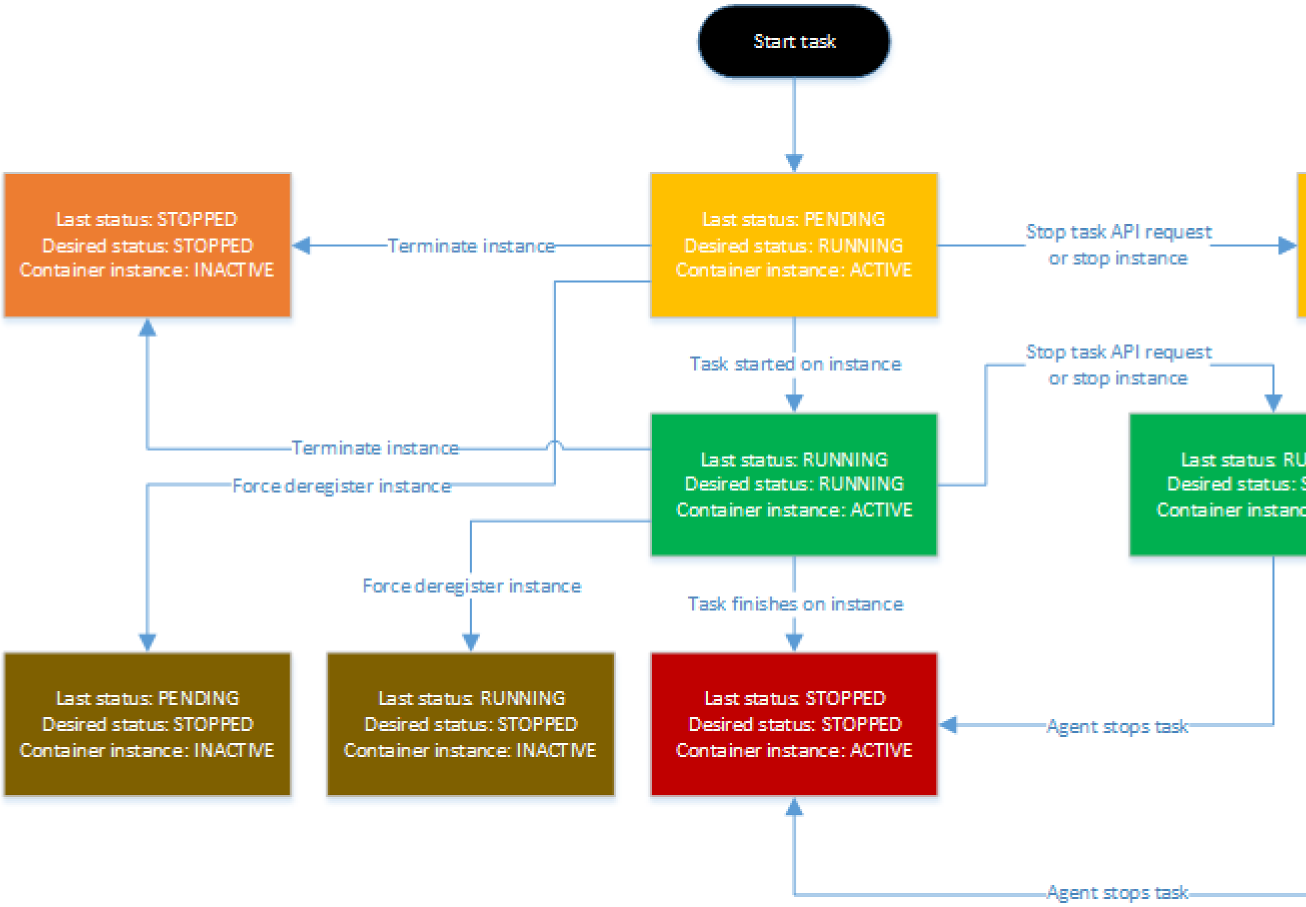
9. Review your task information and choose **Run Task**.

   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. 115) 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.

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 Developer 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. For information on checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 35).

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 an example run command with these volume mounts, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-optimized AMIs) (p. 38).

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 on 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. 95).

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. 40).

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 your CPU and memory utilization across your cluster as a whole, and across the services in your clusters.

Metric	Description
`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. Units: 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. Units: Percent

Dimensions for Amazon ECS Metrics

You can use the dimensions in the following table to refine the metrics returned for your Amazon ECS resources.

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`.

Dimension	Description
ServiceName	This dimension filters the data you request for all resources in a specified service within a specified cluster.

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 in
stances 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 can not 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. In the **ECS > ClusterName,ServiceName** section, choose the service you would like to view running tasks in.
3. Change the period to **1 Minute**.
4. Change the statistic to **Data Samples**. The value displayed indicates the number of `RUNNING` tasks in the service.

Viewing Amazon ECS Metrics

Once 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 CPU and memory utilization, 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.

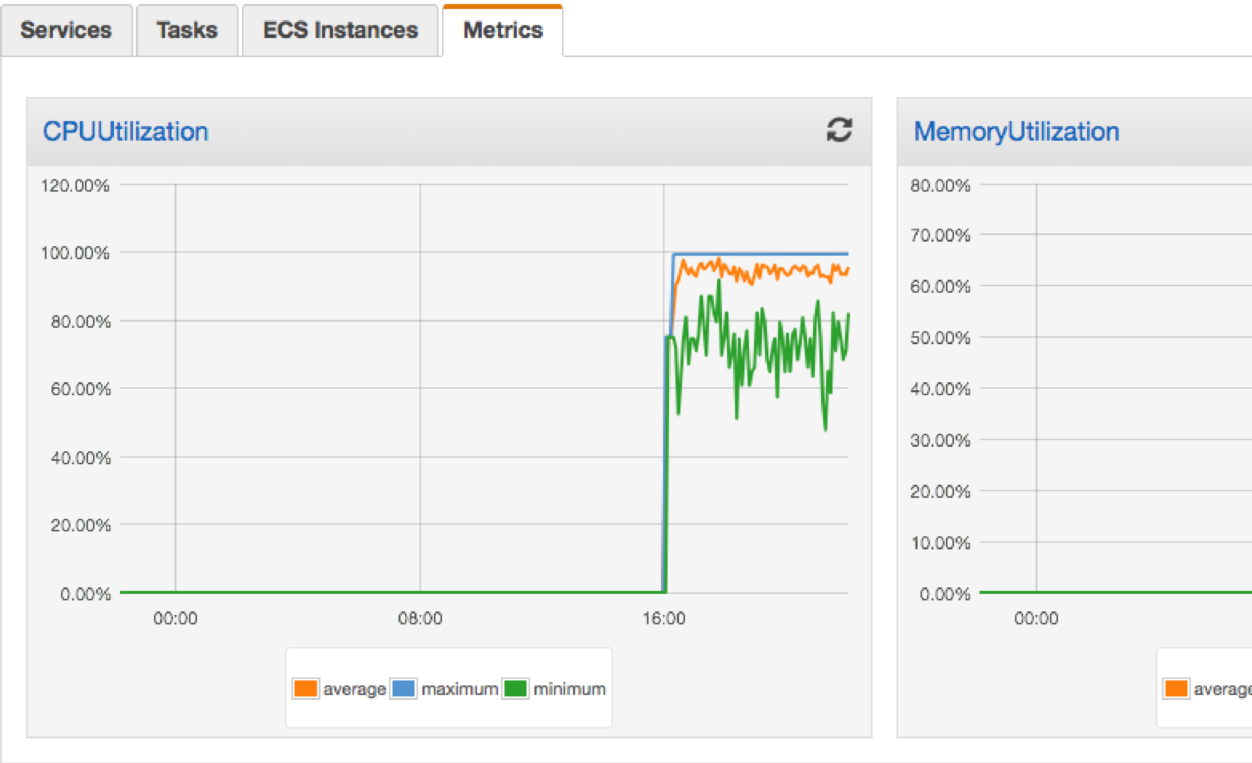
Topics

Viewing Cluster Metrics in the Amazon ECS Console

Cluster CPU and memory utilization 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 on cluster utilization metrics, see Cluster Utilization (p. 79).

To view cluster metrics in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster that you would like to view metrics with.
3. On the **Cluster:** ***`cluster-name`*** page, choose the **Metrics** tab to view cluster metrics.

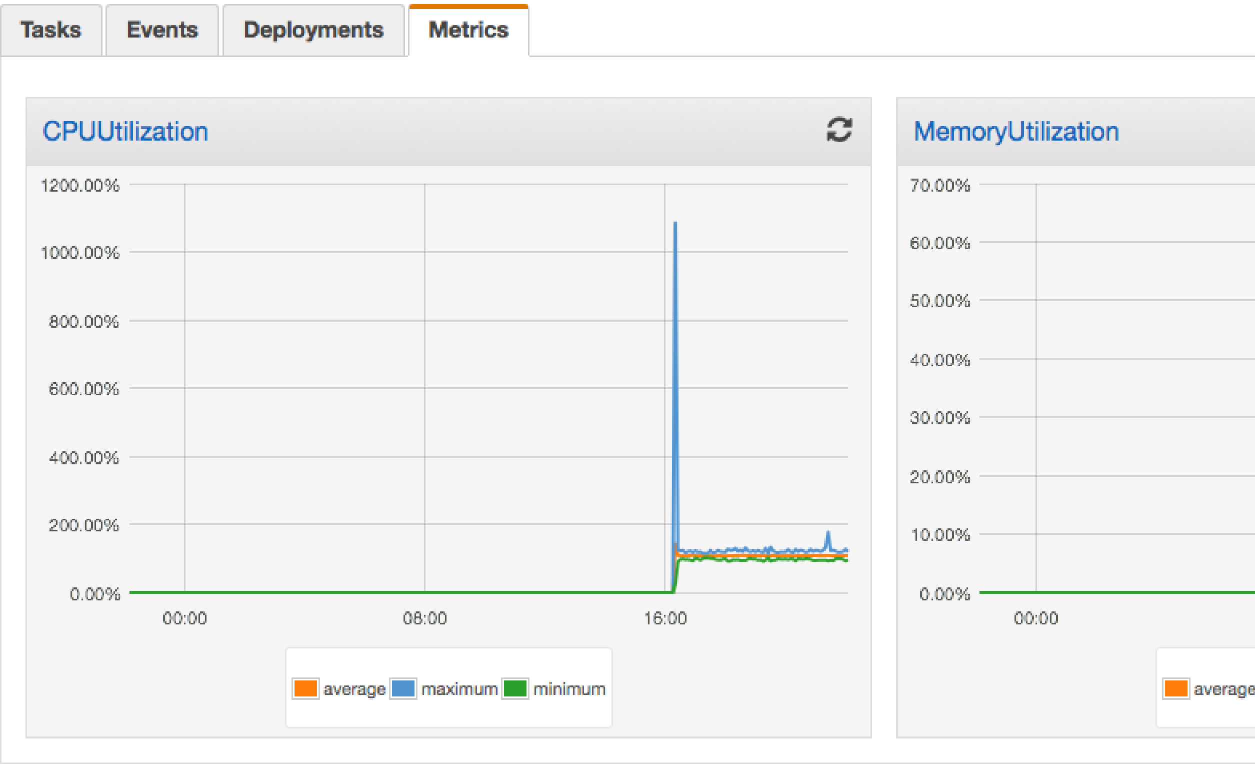


Viewing Service Metrics in the Amazon ECS Console

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 on service utilization metrics, see Service Utilization (p. 79).

To view service metrics in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster that contains the service that you would like 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 that you would like to view metrics with.
5. On the **Service:** ***service-name*** page, choose the **Metrics** tab to view service metrics.

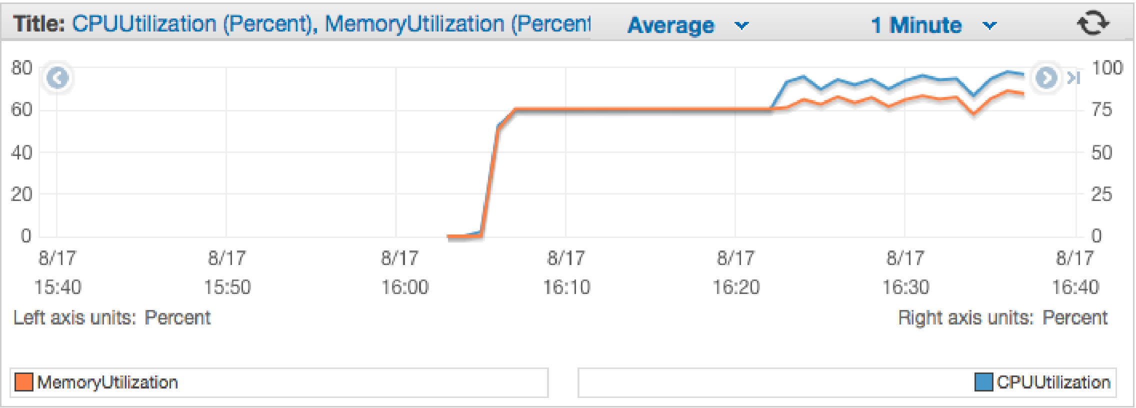


Viewing Amazon ECS Metrics in the CloudWatch Console

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 Utilization (p. 79), Service Utilization (p. 79), and the Service `RUNNING` Task Count (p. 80). For more information on CloudWatch, see the Amazon CloudWatch Developer Guide.

To view cluster metrics in the Amazon ECS 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 that you would like to view. Cluster utilization 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 `MemoryUtilization` metric for your cluster. When the memory utilization of your cluster rises above 75%, 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. 35).

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. 95).

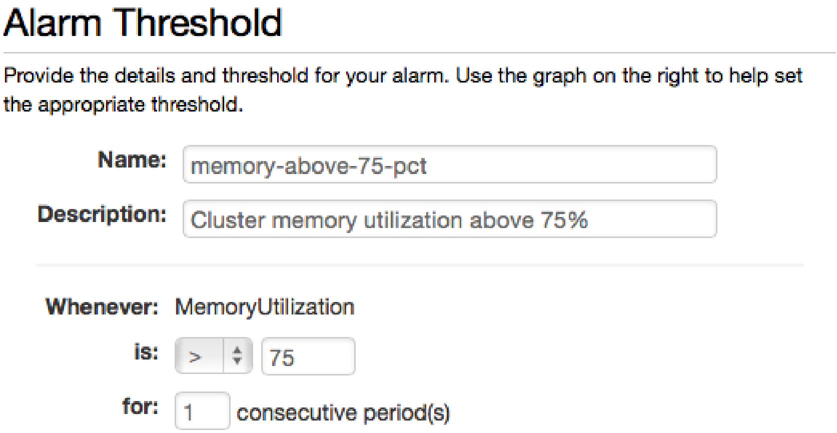
Step 1: Create a CloudWatch Alarm for a Metric

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 Developer Guide*.

For this tutorial, you create an alarm on the cluster `MemoryUtilization` metric to alert when the cluster's memory utilization 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 `MemoryUtilization` 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 utilization above 75%`
7. Set the threshold and time period requirement to `MemoryUtilization` 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 utilization above 75%

Whenever: MemoryUtilization

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 utilization is above 75%.
10. (Optional) You can also create another alarm that triggers when the memory 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 Developer 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-2015.03.f-amazon-ecs-optimized** AMI. The current Amazon ECS-optimized AMI IDs by region are listed below for reference.

Region	AMI Name	AMI ID
`us-east-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-b540eade`
`us-west-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-5721df13`
`us-west-2`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-cb584dfb`
`eu-west-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-2aaef35d`
`ap-northeast-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-8aa61c8a`
`ap-southeast-2`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-5ddc9f67`

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 Developer 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. 94).

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. 40).

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.

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 utilization 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 and Roles

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* 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. 94). 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. 96). For more general information about IAM roles, see IAM Roles in the *IAM User Guide* 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

- Creating Amazon ECS IAM Policies (p. 97)
- Amazon ECS IAM Policy Examples (p. 98)

Policy Structure

The following topics explain the structure of an IAM policy.

Topics

- Policy Syntax (p. 89)
- Actions for Amazon ECS (p. 90)
- Amazon Resource Names for Amazon ECS (p. 90)
- Condition Keys for Amazon ECS (p. 91)
- Checking that Users Have the Required Permissions (p. 92)

Policy Syntax

An IAM policy is a JSON document that consists of one of 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. 90).
- **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. 90). For more information about which API actions support which ARNs, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 93). 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. 91).

For more information about example IAM policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 97).

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. 93).

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* 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. 93). For example policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 97).

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* 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
DeregisterContainer-Instance	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
DescribeContainer-Instances	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 arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A

API action	Resource	Condition keys
ListTasks	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
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

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.

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*"
      ],
      "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 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. 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**.

To create the `ecsInstanceRole` IAM role for your container instances

1. Open the 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. 42).

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

1. Open the 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. 94).
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**.

   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:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer"
      ],
      "Resource": "*"
    }
  ]
}
```

To check for the `ecsServiceRole` in the IAM console

1. Open the 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. 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**.

To create an IAM role for your service scheduler load balancers

1. Open the 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.

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* 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 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. Examples are provided in the sections below.
6. Choose **Create Policy** to finish.

To attach an IAM policy to a user

1. Open the 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** section, choose **Attach User Policy**.
4. In the **Attach Policy** section, select the custom policy you created in the previous procedure and then choose **Attach Policy**.

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

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"

        }
      }
    }
  ]
}
```

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_ac
count_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_ac
count_id>:cluster/<cluster_name>",
          "ecs:container-instances" : [
           "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<contain
er_instance_UUID>"
          ]
        }
      },
      "Resource": [
       "arn:aws:ecs::<region>:<aws_account_id>:task-definition/<task_family>:*"

      ]
    }
  ]
}
```

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_ac
count_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
       "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<container_in
stance_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": [
        "*"
      ]
    }
  ]
}
```

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_ac
count_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following IAM policy allows a user to stop a specified task in a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:StopTask"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_ac
count_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task/<task_UUID>"
      ]
    }
  ]
}
```

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_ac
count_id>:cluster/<cluster_name>"
```

```
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task/<task_UUID>"
      ]
    }
  ]
}
```

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

Region	AMI Name	AMI ID
`us-east-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-b540eade`
`us-west-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-5721df13`
`us-west-2`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-cb584dfb`
`eu-west-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-2aaef35d`
`ap-northeast-1`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-8aa61c8a`
`ap-southeast-2`	**amzn-ami-2015.03.f-amazon-ecs-optimized**	`ami-5ddc9f67`

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_in
stance_UUID"
    ]
}
```

Step 4: Describe your Container Instance

After you have the ARN or UUID 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_UUID
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 2048,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 3955,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "2376",
                        "22",
                        "51678",
                        "2375"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": false,
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:con
tainer-instance/container_instance_UUID",
            "remainingResources": [
                {
                    "integerValue": 2048,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 3955,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "2376",
                        "22",
                        "51678",
                        "2375"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ]
        }
    ]
}
```

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. 48).

```
{
  "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,\"com
mand\":[\"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-defini
tion/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_UUID",
            "overrides": {
                "containerOverrides": [
                    {
                        "name": "sleep"
                    }
                ]
            },
            "lastStatus": "PENDING",
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:con
tainer-instance/container_instance_UUID",
          "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:con
tainer/container_UUID",
                    "taskArn": "arn:aws:ecs:us-east-1:aws_ac
count_id:task/task_UUID",
                    "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 UUID 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_UUID"
    ]
}
```

Step 9: Describe the Running Task

Describe the task using the task UUID retrieved earlier to get more information about the task.

```
$ aws ecs describe-tasks --cluster default --task task_UUID
{
    "failures": [],
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_UUID",
            "overrides": {
                "containerOverrides": [
                    {
                        "name": "sleep"
                    }
                ]
            },
            "lastStatus": "RUNNING",
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:con
tainer-instance/container_instance_UUID",
          "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:con
tainer/container_UUID",
                    "taskArn": "arn:aws:ecs:us-east-1:aws_ac
count_id:task/task_UUID",
                    "lastStatus": "RUNNING",
                    "name": "sleep",
                    "networkBindings": []
                }
            ]
        }
    ]
}
```

Amazon ECS Default Service Limits

The following table provides the default limits for Amazon ECS for an AWS account.

Resource	Default Limit
Number of clusters per region, per account	1000
Number of container instances per cluster	1000
Number of load balancers per service	1
Number of tasks per service	1000
Number of tasks launched (`count`) per **run-task**	10
Number of container instances per **start-task**	10
Throttle on number of container instances per second per **run-task**	5 per cluster
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 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.

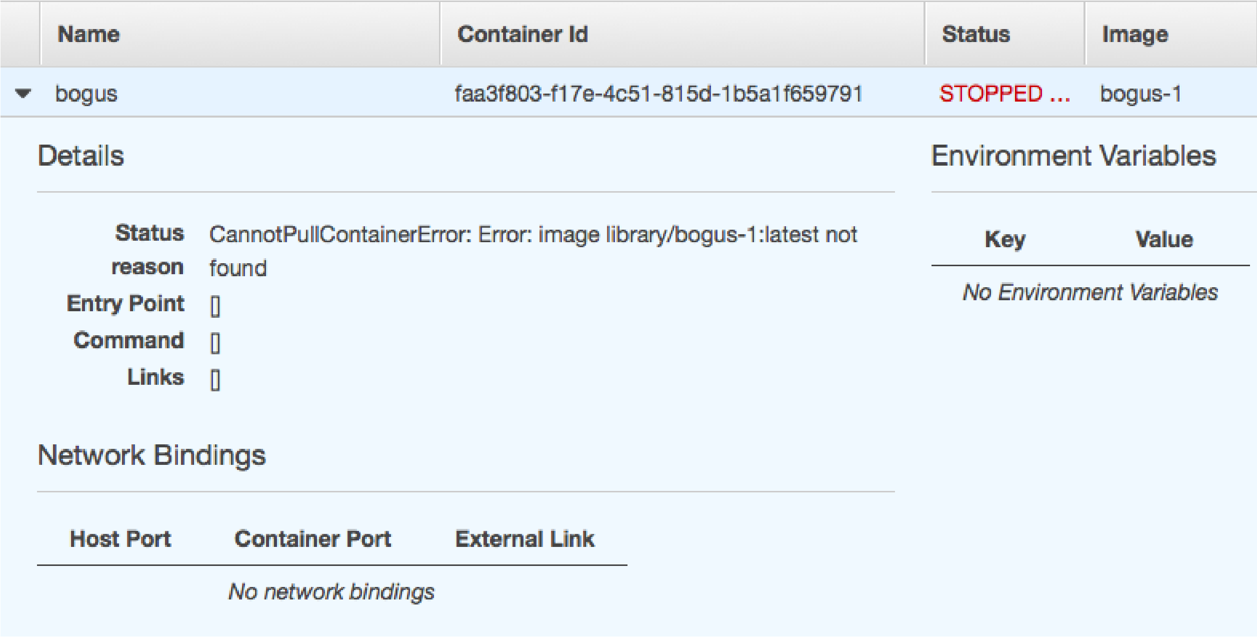
Topics

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. Expand the container and inspect the **Status reason** row to see what caused the task state to change.



	Name	Container Id	Status	Image
▾	bogus	faa3f803-f17e-4c51-815d-1b5a1f659791	STOPPED ...	bogus-1

Details

Status reason CannotPullContainerError: Error: image library/bogus-1:latest not found
Entry Point []
Command []
Links []

Network Bindings

Host Port	Container Port	External Link
	No network bindings	

Environment Variables

Key	Value
No Environment Variables	

In the previous example, the container image name cannot be found. This can happen if you misspell the image name.

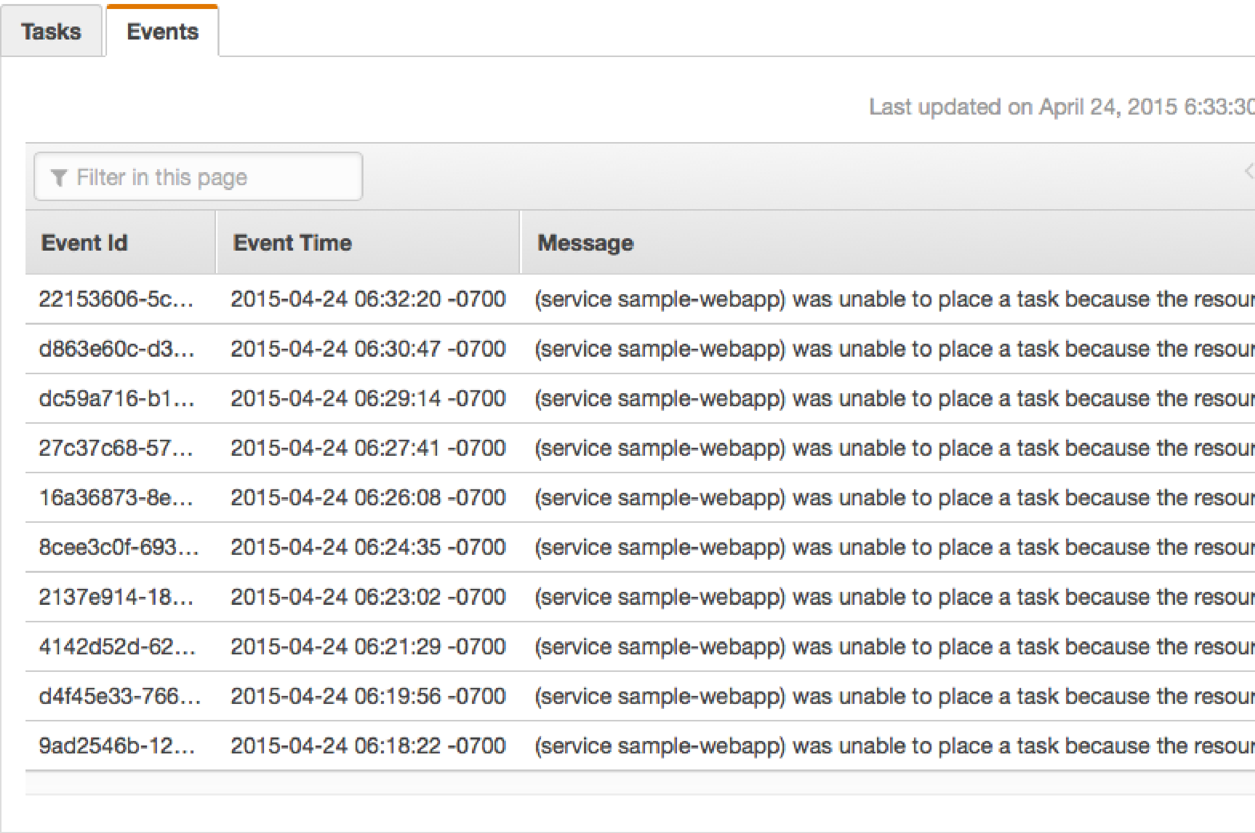
If this inspection does not provide enough information, you can connect to the container instance with SSH and inspect the Docker container locally. For more information, see Inspect Docker Containers (p. 122).

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.



Last updated on April 24, 2015 6:33:30

Event Id	Event Time	Message
22153606-5c…	2015-04-24 06:32:20 -0700	(service sample-webapp) was unable to place a task because the resour
d863e60c-d3…	2015-04-24 06:30:47 -0700	(service sample-webapp) was unable to place a task because the resour
dc59a716-b1…	2015-04-24 06:29:14 -0700	(service sample-webapp) was unable to place a task because the resour
27c37c68-57…	2015-04-24 06:27:41 -0700	(service sample-webapp) was unable to place a task because the resour
16a36873-8e…	2015-04-24 06:26:08 -0700	(service sample-webapp) was unable to place a task because the resour
8cee3c0f-693…	2015-04-24 06:24:35 -0700	(service sample-webapp) was unable to place a task because the resour
2137e914-18…	2015-04-24 06:23:02 -0700	(service sample-webapp) was unable to place a task because the resour
4142d52d-62…	2015-04-24 06:21:29 -0700	(service sample-webapp) was unable to place a task because the resour
d4f45e33-766…	2015-04-24 06:19:56 -0700	(service sample-webapp) was unable to place a task because the resour
9ad2546b-12…	2015-04-24 06:18:22 -0700	(service sample-webapp) was unable to place a task because the resour

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.

Connect to your Container Instance

Much of the diagnostic information for Amazon ECS is available on the container instances themselves. To access this information, you need to connect to the container 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 Securely connect to Linux instances running in a private Amazon VPC.
- 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. Choose the cluster that hosts your container instance.
   c. On the **Cluster** page, choose the **ECS Instances** tab.
   d. On the **Container Instance** column, choose the container instance you wish 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*.

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 Connect to your Container Instance (p. 118).

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs at `/var/log/ecs/ecs-agent.log`.

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 Amazon ECS Container Agent Configuration (p. 40).

```
[ec2-user ~]$ cat /var/log/ecs/ecs-agent.log
t=2015-04-22T20:51:46+0000 lvl=info msg="Starting Agent" module=main
stack=[agent/agent.go:51]
t=2015-04-22T20:51:46+0000 lvl=info msg="Loading configuration" module=main
stack=[agent/agent.go:53]
t=2015-04-22T20:51:46+0000 lvl=info msg="Loading state!" module=statemanager
stack="[github.com/aws/amazon-ecs-agent/agent/statemanager/state_manager.go:215
 agent/agent.go:80]"
t=2015-04-22T20:51:46+0000 lvl=info msg="Registering Instance with ECS" mod
ule=main stack=[agent/agent.go:131]
t=2015-04-22T20:51:46+0000 lvl=info msg=Registered! module="api client"
stack="[github.com/aws/amazon-ecs-agent/agent/api/api_client.go:254 git
hub.com/aws/amazon-ecs-agent/agent/api/api_client.go:193 agent/agent.go:132]"
t=2015-04-22T20:51:46+0000 lvl=info msg="Registration completed successfully"
module=main containerInstance=arn:aws:ecs:us-west-2:aws_account_id:container-
instance/14e8cce9-0b16-4af4-bfac-a85f7587aa98 cluster=default
stack=[agent/agent.go:140]
t=2015-04-22T20:51:46+0000 lvl=info msg="Saving state!" module=statemanager
stack="[github.com/aws/amazon-ecs-agent/agent/statemanager/state_manager.go:180
 github.com/aws/amazon-ecs-agent/agent/statemanager/state_manager.go:154
agent/agent.go:142]"
t=2015-04-22T20:51:46+0000 lvl=info msg="Beginning Polling for updates" mod
ule=main stack=[agent/agent.go:159]
t=2015-04-22T20:51:46+0000 lvl=dbug msg="Added update handlers" module=updater
 stack="[github.com/aws/amazon-ecs-agent/agent/acs/update_handler/updater.go:85
 github.com/aws/amazon-ecs-agent/agent/acs/handler/acs_handler.go:61 git
hub.com/aws/amazon-ecs-agent/agent/utils/utils.go:106 github.com/aws/amazon-
ecs-agent/agent/acs/handler/acs_handler.go:69 agent/agent.go:160]"
t=2015-04-22T20:51:46+0000 lvl=info msg="Creating poll dialer" module="acs
client" host=ecs-a-1.us-west-2.amazonaws.com stack="[github.com/aws/amazon-ecs-
agent/agent/acs/client/acs_client.go:130 github.com/aws/amazon-ecs-
agent/agent/acs/handler/acs_handler.go:63 github.com/aws/amazon-ecs-
agent/agent/utils/utils.go:106 github.com/aws/amazon-ecs-agent/agent/acs/hand
ler/acs_handler.go:69 agent/agent.go:160]"
```

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
```

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 Connect to your Container Instance (p. 118).

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": "189a8ff4b5f04affe40e5160a5ffad
ca395136eb5faf4950c57963c06f82c76d",
                    "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": "dc7240fe892ab233db
bcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "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. 46).

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. 118).

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-fore
ground"     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-90fe
faa68498a8a80700
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/loc
al/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.

- `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*.