Amazon ECS

User Guide for AWS Fargate

API Version 2014-11-13

Amazon ECS: User Guide for AWS Fargate

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon ECS? ... 1
- Features of Amazon ECS ... 1
  - Containers and Images ... 2
  - Task Definitions ... 3
  - Tasks and Scheduling ... 4
  - Clusters ... 4
- How to Get Started with Amazon ECS ... 4
- Related Services ... 4
- Accessing Amazon ECS ... 5

Setting Up ... 7
- Sign Up for AWS ... 7
- Create an IAM User ... 7
- Create an IAM Role ... 9
- Create a Virtual Private Cloud ... 9
- Install the AWS CLI ... 9

Docker Basics for Amazon ECS ... 11
- Installing Docker ... 11
- Create a Docker Image ... 12
- (Optional) Push your image to Amazon Elastic Container Registry ... 13
- Next Steps ... 14

Getting Started with Amazon ECS using Fargate ... 16
- Prerequisites ... 16
- Step 1: Create a Task Definition ... 16
- Step 2: Configure the Service ... 17
- Step 3: Configure the Cluster ... 18
- Step 4: Review ... 18
- Step 5: (Optional) View your Service ... 18

AWS Fargate Platform Versions ... 19
- Platform Version Considerations ... 19
- Available Platform Versions ... 19

Clusters ... 20
- Creating a Cluster ... 20
- Deleting a Cluster ... 21

Task Definitions ... 22
- Task Definition Considerations ... 22
  - Network Mode ... 23
  - Task CPU and Memory ... 23
  - Logging ... 24
  - Amazon ECS Task Execution IAM Role ... 24
  - Example Task Definition ... 24
  - Task Storage ... 25
- Application Architecture ... 26
  - Using the Fargate Launch Type ... 26
- Creating a Task Definition ... 26
  - Task Definition Template ... 27
- Task Definition Parameters ... 31
  - Family ... 31
  - Task Execution Role ... 31
  - Network Mode ... 32
  - Container Definitions ... 32
  - Volumes ... 46
  - Launch Types ... 47
  - Task Size ... 47
  - Other Task Definition Parameters ... 49

Launch Types ... 50
Fargate Launch Type ... 50
EC2 Launch Type ... 51
Using Data Volumes in Tasks ... 52
Task Networking ... 54
Enabling Task Networking ... 54
Task Networking Considerations ... 54
Using the awslogs Log Driver ... 55
Enabling the awslogs Log Driver for Your Containers ... 55
Creating Your Log Groups ... 55
Available awslogs Log Driver Options ... 56
Specifying a Log Configuration in your Task Definition ... 58
Viewing awslogs Container Logs in CloudWatch Logs ... 59
Private Registry Authentication for Tasks ... 61
Private Registry Authentication Required IAM Permissions ... 62
Enabling Private Registry Authentication ... 62
Example Task Definitions ... 64
Updating a Task Definition ... 65
Deregistering Task Definitions ... 66
Scheduling Tasks ... 67
Running Tasks ... 68
Scheduled Tasks (`cron`) ... 70
Task Retirement ... 72
Identifying Tasks Scheduled for Retirement ... 72
Working with Tasks Scheduled for Retirement ... 72
Services ... 74
Service Scheduler Concepts ... 74
Daemon ... 75
Replica ... 75
Additional Service Concepts ... 75
Service Definition Parameters ... 75
Service Load Balancing ... 81
Load Balancing Concepts ... 81
Load Balancer Types ... 82
Creating a Load Balancer ... 84
Service Auto Scaling ... 90
Target Tracking Scaling Policies ... 91
Step Scaling Policies ... 95
Service Discovery ... 97
Service Discovery Concepts ... 98
Service Discovery Considerations ... 98
Service Discovery Pricing ... 99
Tutorial: Creating a Service Using Service Discovery ... 99
Creating a Service ... 108
Configuring Basic Service Parameters ... 108
Configure a Network ... 109
(Optional) Configuring Your Service to Use Service Discovery ... 112
(Optional) Configuring Your Service to Use Service Auto Scaling ... 113
Review and Create Your Service ... 115
Updating a Service ... 115
Service Throttle Logic ... 117
Resources and Tags ... 119
Amazon Resource Names (ARNs) and IDs ... 119
Working with Account Settings ... 120
Tagging Your Resources ... 121
Tag Basics ... 122
Tagging Your Resources ... 122

Tag Restrictions ... 123
Tagging Your Resources for Billing ... 123
Working with Tags Using the Console ... 123
Working with Tags Using the CLI or API ... 124
Usage Reports ... 126
Monitoring ... 127
Monitoring Tools ... 127
Automated Tools ... 127
Manual Tools ... 128
CloudWatch Metrics ... 128
Enabling CloudWatch Metrics ... 129
Available Metrics and Dimensions ... 129
Service Utilization ... 131
Service `RUNNING` Task Count ... 132
Viewing Amazon ECS Metrics ... 133
CloudWatch Events ... 134
Amazon ECS Events ... 134
Handling Events ... 136
Tutorial: Listening for Amazon ECS CloudWatch Events ... 137
Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events ... 139
IAM Policies, Roles, and Permissions ... 142
Policy Structure ... 142
Policy Syntax ... 143
Actions for Amazon ECS ... 143
Amazon Resource Names for Amazon ECS ... 144
Condition Keys for Amazon ECS ... 145
Testing Permissions ... 146
Supported Resource-Level Permissions ... 146
Creating IAM Policies ... 149
Managed Policies and Trust Relationships ... 149
Amazon ECS Managed Policies and Trust Relationships ... 150
Amazon ECR Managed Policies ... 154
Amazon ECS Task Execution IAM Role ... 156
Using Service-Linked Roles ... 157
Service-Linked Role Permissions for Amazon ECS ... 158
Creating a Service-Linked Role for Amazon ECS ... 159
Editing a Service-Linked Role for Amazon ECS ... 160
Deleting a Service-Linked Role for Amazon ECS ... 160
Amazon ECS Service Scheduler IAM Role ... 162
Amazon ECS Service Auto Scaling IAM Role ... 164
Amazon ECS Task Role ... 165
CloudWatch Events IAM Role ... 166
IAM Roles for Tasks ... 168
Benefits of Using IAM Roles for Tasks ... 169
Creating an IAM Role and Policy for your Tasks ... 169
Using a Supported AWS SDK ... 170
Specifying an IAM Role for your Tasks ... 171
Amazon ECS IAM Policy Examples ... 171
Amazon ECS First Run Wizard ... 171
Clusters ... 174
List and Describe Tasks ... 176
Create Services ... 176
Update Services ... 177
Using the Amazon ECS CLI ... 179
Installing the Amazon ECS CLI ... 179
Step 1: Download the Amazon ECS CLI ... 179
Step 2: (Optional) Verify the Amazon ECS CLI ... 179

Step 3: Apply Execute Permissions to the Binary ... 184
Step 4: Complete the Installation ... 184
Configuring the Amazon ECS CLI ... 184
Profiles ... 185
Cluster Configurations ... 185
Order of Precedence ... 185
Migrating Configuration Files ... 186
Migrating Older Configuration Files to the v1.0.0+ Format ... 187
Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI ... 187
Prerequisites ... 187
Step 1: Create the Task Execution IAM Role ... 187
Step 2: Configure the Amazon ECS CLI ... 188
Step 3: Create a Cluster and Security Group ... 188
Step 4: Create a Compose File ... 189
Step 5: Deploy the Compose File to a Cluster ... 190
Step 6: View the Running Containers on a Cluster ... 190
Step 7: View the Container Logs ... 190
Step 8: Scale the Tasks on the Cluster ... 190
Step 9: Clean Up ... 191
Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI ... 191
Prerequisites ... 191
Configure the Amazon ECS CLI ... 191
Create an Amazon ECS Service Configured to Use Service Discovery ... 192
Using the AWS CLI ... 195
Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI ... 195
Prerequisites ... 195
Step 1: (Optional) Create a Cluster ... 195
Step 2: Register a Task Definition ... 196
Step 3: List Task Definitions ... 198
Step 4: Create a Service ... 198
Step 5: List Services ... 199
Step 6: Describe the Running Service ... 200
Task Metadata Endpoint ... 202
Enabling Task Metadata ... 202
Task Metadata Endpoint Paths ... 202
Task Metadata JSON Response ... 202
Example Task Metadata Response ... 204
Tutorial: Continuous Deployment with AWS CodePipeline ... 206
Prerequisites ... 206
Step 1: Add a Build Specification File to Your Source Repository ... 206
Step 2: Creating Your Continuous Deployment Pipeline ... 208
Step 3: Add Amazon ECR Permissions to the AWS CodeBuild Role ... 209
Step 4: Test Your Pipeline ... 209
Service Limits ... 211
Logging Amazon ECS API Calls with AWS CloudTrail ... 213
Amazon ECS Information in CloudTrail ... 213
Understanding Amazon ECS Log File Entries ... 214
Troubleshooting ... 216
Troubleshooting First-Run Wizard Launch Issues ... 216
Checking Stopped Tasks for Errors ... 216
Service Event Messages ... 218
Service Event Messages ... 219
Invalid CPU or Memory Value Specified ... 220
Cannot Pull Container Image Error ... 221
Troubleshooting Service Load Balancers ... 222
Troubleshooting IAM Roles for Tasks ... 223
Document History ... 226

AWS Glossary 232

What is Amazon Elastic Container Service?

Amazon Elastic 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. You can host your cluster on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks using the Fargate launch type. For more control you can host your tasks on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances that you manage by using the EC2 launch type. For more information about launch types, see Amazon ECS Launch Types (p. 50).

Amazon ECS lets you launch and stop container-based applications with simple API calls, allows you to get the state of your cluster from a centralized service, and gives you access to many familiar Amazon EC2 features.

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

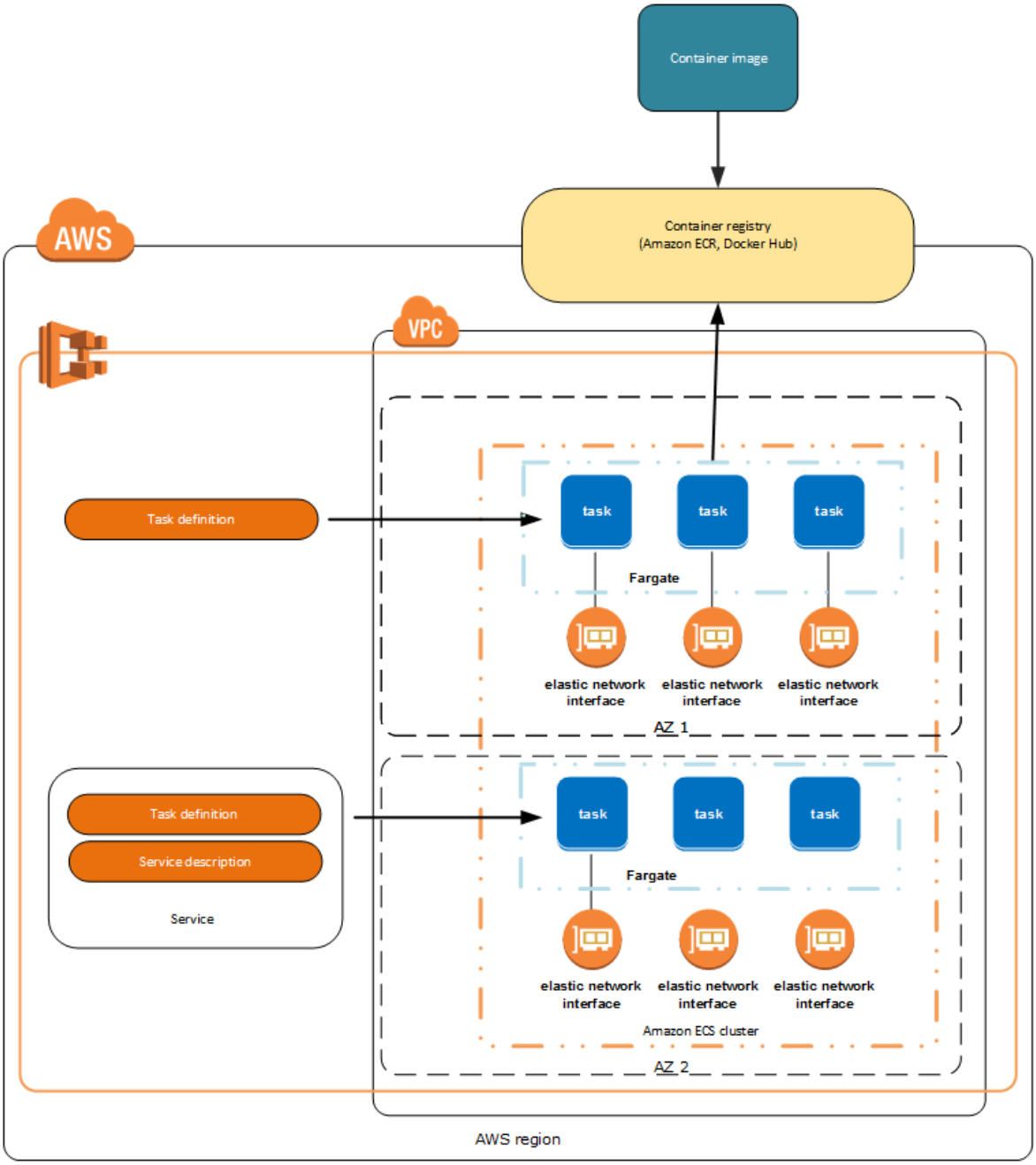
Amazon ECS can be used to create a consistent deployment and build experience, manage, and scale batch and Extract-Transform-Load (ETL) workloads, and build sophisticated application architectures on a microservices model. For more information about Amazon ECS use cases and scenarios, see Container Use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

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

Features of Amazon ECS

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

The following diagram shows the architecture of an Amazon ECS environment using the Fargate launch type:

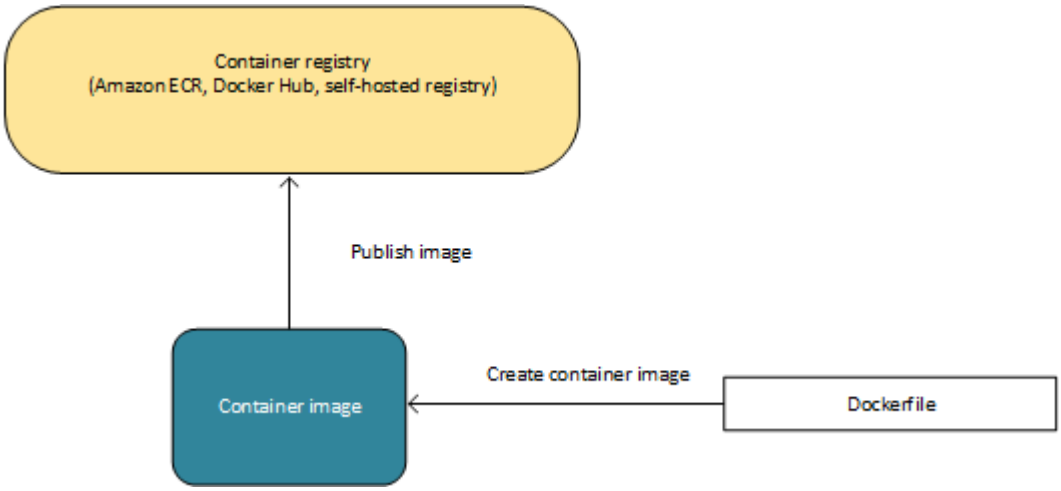


The following sections dive into these individual elements of the Amazon ECS architecture in more detail.

Containers and Images

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

Images are typically built from a Dockerfile, a plain text file that specifies all of the components that are included in the container. These images are then stored in a *registry* from which they can be downloaded and run on your cluster. For more information about container technology, see Docker Basics for Amazon ECS (p. 11).



Task Definitions

To prepare your application to run on Amazon ECS, you create a *task definition*. The task definition is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form your application. It can be thought of as a blueprint for your application. Task definitions specify various parameters for your application. Examples of task definition parameters are which containers to use, which launch type to use, which ports should be opened for your application, and what data volumes should be used with the containers in the task. The specific parameters available for the task definition depend on which launch type you are using. For more information about creating task definitions, see Amazon ECS Task Definitions (p. 22).

The following is an example of a task definition containing a single container that runs an NGINX web server using the Fargate launch type. For a more extended example demonstrating the use of multiple containers in a task definition, see Example Task Definitions (p. 64).

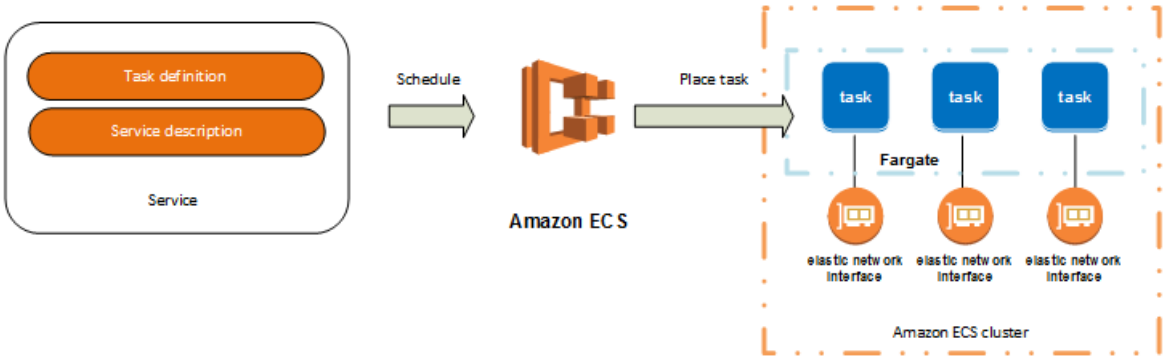
```
{
    "family": "webserver",
    "containerDefinitions": [
        {
            "name": "web",
            "image": "nginx",
            "memory": "100",
            "cpu": "99"
        },
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "memory": "512",
    "cpu": "256",
}
```

Tasks and Scheduling

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

Each task that uses the Fargate launch type has it's own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

The Amazon ECS task scheduler is responsible for placing tasks within your cluster. There are several different scheduling options available. For example, you can define a *service* that runs and maintains a specified number of tasks simultaneously. For more information about the different scheduling options available, see Scheduling Amazon ECS Tasks (p. 67).



Clusters

When you run tasks using Amazon ECS, you place them on a *cluster*, which is a logical grouping of resources. When using the Fargate launch type with tasks within your cluster, Amazon ECS manages your cluster resources.

For more information about creating clusters, see Amazon ECS Clusters (p. 20).

How to Get Started with Amazon ECS

If you are using Amazon ECS for the first time, the AWS Management Console for Amazon ECS provides a first-run wizard that steps you through defining a task definition for a web server, configuring a service, and launching your first Fargate task. The first-run wizard is highly recommended for users who have no prior experience with Amazon ECS. For more information, see the Getting Started with Amazon ECS using Fargate (p. 16) tutorial.

Alternatively, you can install the AWS Command Line Interface (AWS CLI) to use Amazon ECS. For more information, see Setting Up with Amazon ECS (p. 7).

Related Services

Amazon ECS can be used along with the following AWS services:

AWS Identity and Access Management

IAM is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). In Amazon ECS, IAM can be used to control access at the container

instance level using IAM roles, and at the task level using IAM task roles. For more information, see Amazon ECS IAM Policies, Roles, and Permissions (p. 142).

Amazon EC2 Auto Scaling

Auto Scaling is a web service that enables you to automatically scale out or in your tasks based on user-defined policies, health status checks, and schedules. You can use Auto Scaling with a Fargate task within a service to scale in response to a number of metrics. For more information, see Service Auto Scaling (p. 90).

Elastic Load Balancing

Elastic Load Balancing automatically distributes incoming application traffic across the tasks in your Amazon ECS service. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic. You can use Elastic Load Balancing to create an endpoint that balances traffic across services in a cluster. For more information, see Service Load Balancing (p. 81).

Amazon Elastic Container Registry

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

AWS CloudFormation

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

Accessing Amazon ECS

You can work with Amazon ECS in the following ways:

AWS Management Console

The console is a browser-based interface to manage Amazon ECS resources. For a tutorial that guides you through the console, see Getting Started with Amazon ECS using Fargate (p. 16).

AWS command line tools

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

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

Amazon ECS CLI

In addition to using the AWS CLI to access Amazon ECS resources, you can use the Amazon ECS CLI, which provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment using Docker Compose. For more information, see Using the Amazon ECS Command Line Interface (p. 179).

AWS SDKs

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

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

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

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 similar. The following guide prepares you for launching your first cluster using either the Amazon ECS first-run wizard or the Amazon ECS Command Line Interface (CLI).

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. 7)
2. Create an IAM User (p. 7)
3. Create an IAM Role (p. 9)
4. Create a Virtual Private Cloud (p. 9)
5. 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 https://aws.amazon.com/, and then choose **Create an AWS Account**.

   Note
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.
2. Follow the online instructions.

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

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

Create an IAM User

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

user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

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

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

1. Use your AWS account email address and password to sign in as the *AWS account root user* to the IAM console at https://console.aws.amazon.com/iam/.

   Note
   We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type **`Administrator`**.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type **`Administrators`**.
9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

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

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

Before the Amazon ECS container agent can make calls to the Amazon ECS API actions on your behalf, it requires an IAM policy and role for the service to know that the agent belongs to you.

For tasks using the Fargate launch type, you can create an IAM role that allows the agent to pull container images from Amazon ECR or to use the awslogs log driver, which is currently the only supported logging option for this launch type. This role is referred to as the Amazon ECS task execution IAM role. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

> **Note**
> These IAM roles are automatically created for you in the Amazon ECS console first-run experience, so if you intend to use the console, you can move ahead to the next section. If you do not intend to use the console, and instead plan to use the AWS CLI, these IAM roles will need to be manually created.

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.

> **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 skip to the next section.

If you have a default VPC, you also can skip this section and move to the next task, Install the AWS CLI (p. 9). 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 Amazon 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 **Launch 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*.

Install the AWS CLI

The AWS Management Console can be used to manage all operations manually with Amazon ECS. However, installing the AWS CLI on your local desktop or a developer box enables you to build scripts that can automate common management tasks in Amazon ECS.

To use the AWS CLI with Amazon ECS, install the latest AWS CLI, version. 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 for Amazon ECS

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

Topics

- Installing Docker (p. 11)
- Create a Docker Image (p. 12)
- (Optional) Push your image to Amazon Elastic Container Registry (p. 13)
- Next Steps (p. 14)

Installing Docker

> **Note**
> If you already have Docker installed, skip to Create a Docker Image (p. 12).

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

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

To install Docker on an Amazon Linux instance

1. Launch an instance with the Amazon Linux AMI. For more information, see Launching an Instance in the *Amazon EC2 User Guide for Linux Instances*.
2. Connect to your instance. For more information, see Connect to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.
3. Update the installed packages and package cache on your instance.

   ```
   sudo yum update -y
   ```

4. Install the most recent Docker Community Edition package.

   ```
   sudo yum install -y docker
   ```

5. Start the Docker service.

   ```
   sudo service docker start
   ```

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

   ```
   sudo usermod -a -G docker ec2-user
   ```

7. Log out and log back in again to pick up the new `docker` group permissions. You can accomplish this by closing your current SSH terminal window and reconnecting to your instance in a new one. Your new SSH session will have the appropriate `docker` group permissions.
8. Verify that the `ec2-user` can run Docker commands without `sudo`.

```
docker info
```

Note
In some cases, you may need to reboot your instance to provide permissions for the `ec2-user` to access the Docker daemon. Try rebooting your instance if you see the following error:

```
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

Create a Docker Image

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 web application, and test it on your local system or EC2 instance, and then push the image to a container registry (such as Amazon ECR or Docker Hub) so you can use it in an ECS task definition.

To create a Docker image of a simple web application

1. Create a file called `Dockerfile`. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference.

   ```
   touch Dockerfile
   ```

2. Edit the `Dockerfile` you just created and add the following content.

   ```
   FROM ubuntu:16.04

   # Install dependencies
   RUN apt-get update
   RUN apt-get -y install apache2

   # Install apache and write hello world message
   RUN echo 'Hello World!' > /var/www/html/index.html

   # Configure apache
   RUN echo '. /etc/apache2/envvars' > /root/run_apache.sh
   RUN echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh
   RUN echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh
   RUN echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh
   RUN chmod 755 /root/run_apache.sh

   EXPOSE 80

   CMD /root/run_apache.sh
   ```

   This Dockerfile uses the Ubuntu 16.04 image. The `RUN` instructions update the package caches, install some software packages for the web server, and then write the "Hello World!" content to the web server's document root. The `EXPOSE` instruction exposes port 80 on the container, and the `CMD` instruction starts the web server.

3. Build the Docker image from your Dockerfile.

   Note
   Some versions of Docker may require the full path to your Dockerfile in the following command, instead of the relative path shown below.

```
docker build -t hello-world .
```

4. Run **docker images** to verify that the image was created correctly.

```
docker images --filter reference=hello-world
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago       258MB
```

5. 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.

```
docker run -p 80:80 hello-world
```

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

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

   - If you are using an EC2 instance, this is the **Public DNS** value for the server, which is the same address you use to connect to the instance with SSH. Make sure that the security group for your instance allows inbound traffic on port 80.
   - If you are running Docker locally, point your browser to http://localhost/.
   - If you are using **docker-machine** on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the **docker-machine ip** command, substituting *`machine-name`* with the name of the docker machine you are using.

```
docker-machine ip machine-name
```

   You should see a web page with your "Hello World!" statement.

7. Stop the Docker container by typing **Ctrl + c**.

(Optional) Push your image to Amazon Elastic Container Registry

Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. For Amazon ECR product details, featured customer case studies, and FAQs, see the Amazon Elastic Container Registry product detail pages.

This section requires the following:

- You have the AWS CLI installed and configured. If you do not have the AWS CLI installed on your system, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.
- Your user has the required IAM permissions to access the Amazon ECR service. For more information, see Amazon ECR Managed Policies.

To tag your image and push it to Amazon ECR

1. Create an Amazon ECR repository to store your `hello-world` image. Note the `repositoryUri` in the output.

```
aws ecr create-repository --repository-name hello-world
```

Output:

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "hello-world",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/hello-world",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world"
    }
}
```

2. Tag the `hello-world` image with the `repositoryUri` value from the previous step.

```
docker tag hello-world aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world
```

3. Run the **aws ecr get-login --no-include-email** command to get the **docker login** authentication command string for your registry.

   Note
   The **get-login** command is available in the AWS CLI starting with version 1.9.15; however, we recommend version 1.11.91 or later for recent versions of Docker (17.06 or later). You can check your AWS CLI version with the **aws --version** command. If you are using Docker version 17.06 or later, include the `--no-include-email` option after `get-login`. If you receive an `Unknown options: --no-include-email` error, install the latest version of the AWS CLI. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

```
aws ecr get-login --no-include-email
```

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

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

5. Push the image to Amazon ECR with the `repositoryUri` value from the earlier step.

```
docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world
```

Next Steps

After the image push is finished, you can use your image in your Amazon ECS task definitions, which you can use to run tasks with.

Note
This section requires the AWS CLI. If you do not have the AWS CLI installed on your system, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To register a task definition with the `hello-world` image

1. Create a file called `hello-world-task-def.json` with the following contents, substituting the `repositoryUri` from the previous section for the `image` field.

```
{
    "family": "hello-world",
    "containerDefinitions": [
        {
            "name": "hello-world",
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world",
            "cpu": 10,
            "memory": 500,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "entryPoint": [
                "/usr/sbin/apache2",
                "-D",
                "FOREGROUND"
            ],
            "essential": true
        }
    ]
}
```

2. Register a task definition with the `hello-world-task-def.json` file.

```
aws ecs register-task-definition --cli-input-json file://hello-world-task-def.json
```

The task definition is registered in the `hello-world` family as defined in the JSON file.

To run a task with the `hello-world` task definition

Important
Before you can run tasks in Amazon ECS, you need to launch container instances into a default cluster. For more information about how to set up and launch container instances, see Setting Up with Amazon ECS (p. 7) and Getting Started with Amazon ECS using Fargate (p. 16).

- Use the following AWS CLI command to run a task with the `hello-world` task definition.

```
aws ecs run-task --task-definition hello-world
```

Getting Started with Amazon ECS using Fargate

Get started with Amazon Elastic Container Service (Amazon ECS) by creating a task definition that uses the Fargate launch type, scheduling tasks, and configuring a cluster in the Amazon ECS console.

In the Regions that support AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using Fargate. For more information, see Fargate Launch Type (p. 50). The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

> **Important**
> For more information about the Amazon ECS first-run wizard for EC2 tasks, see Getting Started with Amazon ECS.

Complete the following tasks to get started with Amazon ECS using Fargate:

- Prerequisites (p. 16)
- Step 1: Create a Task Definition (p. 16)
- Step 2: Configure the Service (p. 17)
- Step 3: Configure the Cluster (p. 18)
- Step 4: Review (p. 18)
- Step 5: (Optional) View your Service (p. 18)

Prerequisites

Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 7) and that your AWS user has either the permissions specified in the `AdministratorAccess` or Amazon ECS First Run Wizard (p. 171) IAM policy example.

The first-run wizard attempts to automatically create the task execution IAM role, which is required for Fargate tasks. To ensure that the first-run experience is able to create this IAM role, one of the following must be true:

- Your user has administrator access. For more information, see Setting Up with Amazon ECS (p. 7).
- Your user has the IAM permissions to create a service role. For more information, see Creating a Role to Delegate Permissions to an AWS Service.
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

Step 1: Create a Task Definition

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

1. Open the Amazon ECS console first-run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.
2. From the navigation bar, select the **US East (N. Virginia)** Region.

   Note
   You can complete this first-run wizard using these steps for any Region that supports Amazon ECS using Fargate. For more information, see Fargate Launch Type (p. 50).
3. Configure your container definition parameters.

   For **Container definition**, the first-run wizard comes preloaded with the `sample-app`, `nginx`, and `tomcat-webserver` container definitions in the console. You can optionally rename the container or review and edit the resources used by the container (such as CPU units and memory limits) by choosing **Edit** and editing the values shown. For more information, see Container Definitions (p. 32).

   Note
   If you are using an Amazon ECR image in your container definition, be sure to use the full `registry/repository:tag` naming for your Amazon ECR images. For example, *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`*.
4. For **Task definition**, the first-run wizard defines a task definition to use with the preloaded container definitions. You can optionally rename the task definition and edit the resources used by the task (such as the **Task memory** and **Task CPU** values) by choosing **Edit** and editing the values shown. For more information, see Task Definition Parameters (p. 31).

   Task definitions created in the first-run wizard are limited to a single container for simplicity. You can create multi-container task definitions later in the Amazon ECS console.
5. Choose **Next**.

Step 2: Configure the Service

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

The first-run wizard comes preloaded with a service definition, and you can see the `sample-app-service` service defined in the console. You can optionally rename the service or review and edit the details by choosing **Edit** and doing the following:

1. In the **Service name** field, select a name for your service.
2. In the **Number of desired tasks** field, enter the number of tasks to launch with your specified task definition.
3. In the **Security group** field, specify a range of IPv4 addresses to allow inbound traffic from, in CIDR block notation. For example, `203.0.113.0/24`.
4. (Optional) You can choose to use an Application Load Balancer with your service. When a task is launched from a service that is configured to use a load balancer, the task is registered with the load balancer. Traffic from the load balancer is distributed across the instances in the load balancer. For more information, see Introduction to Application Load Balancers.

   Important
   Application Load Balancers do incur cost while they exist in your AWS resources. For more information, see Application Load Balancer Pricing.

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

- In the **Container to load balance** section, choose the **Load balancer listener port**. The default value here is 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. 81).

5. Review your service settings and click **Save**, **Next**.

Step 3: Configure the Cluster

In this section of the wizard, you name your cluster, and then Amazon ECS take cares of the networking and IAM configuration for you.

1. In the **Cluster name** field, choose a name for your cluster.
2. Click **Next** to proceed.

Step 4: Review

1. Review your task definition, task configuration, and cluster configuration and click **Create** to finish. You are directed to a **Launch Status** page that shows the status of your launch. It describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).
2. After the launch is complete, choose **View service**.

Step 5: (Optional) View your Service

If your service is a web-based application, such as the **Amazon ECS sample** application, you can view its containers with a web browser.

1. On the **Service:** ***service-name*** page, choose the **Tasks** tab.
2. Choose a task from the list of tasks in your service.
3. In the **Network** section, choose the **ENI Id** for your task. This takes you to the Amazon EC2 console where you can view the details of the network interface associated with your task, including the **IPv4 Public IP** address.
4. Enter the **IPv4 Public IP** address in your web browser and you should see a webpage that displays the **Amazon ECS sample** application.

AWS Fargate Platform Versions

AWS Fargate platform versions are used to refer to a specific runtime environment for Fargate task infrastructure. It is a combination of the kernel and container runtime versions.

New platform versions are released as the runtime environment evolves, for example, if there are kernel or operating system updates, new features, bug fixes, or security updates. Security updates and patches are deployed automatically for your Fargate tasks. If a security issue is found that affects a platform version, AWS patches the platform version. In some cases, you may be notified that your Fargate tasks have been scheduled for retirement. For more information, see Task Retirement (p. 72).

Topics

- Platform Version Considerations (p. 19)
- Available AWS Fargate Platform Versions (p. 19)

Platform Version Considerations

The following should be considered when specifying a platform version:

- When specifying a platform version, you can use either the version number (for example, `1.2.0`) or `LATEST`.
- To use a specific platform version, specify the version number when creating or updating your service. If you specify `LATEST`, your tasks use the most current platform version available, which may not be the most recent platform version.
- If you have a service with running tasks and want to update their platform version, you can update your service, specify a new platform version, and choose **Force new deployment**. Your tasks are redeployed with the latest platform version. For more information, see Updating a Service (p. 115).
- If your service is scaled up without updating the platform version, those tasks receive the platform version that was specified on the service's current deployment.

Available AWS Fargate Platform Versions

The following is a list of the platform versions currently available:

Fargate Platform Version-1.2.0

- Added support for private registry authentication using AWS Secrets Manager. For more information, see Private Registry Authentication for Tasks (p. 61).

Fargate Platform Version-1.1.0

- Added support for the Amazon ECS task metadata endpoint. For more information, see Amazon ECS Task Metadata Endpoint (p. 202).
- Added support for Docker health checks in container definitions. For more information, see Health Check (p. 36).
- Added support for Amazon ECS service discovery. For more information, see Service Discovery (p. 97).

Fargate Platform Version-1.0.0

- Based on Amazon Linux 2017.09.
- Initial release.

Amazon ECS Clusters

An Amazon ECS cluster is a logical grouping of tasks or services. When you first use Amazon ECS, a default cluster is created for you, but you can create multiple clusters in an account to keep your resources separate.

The following are general concepts about Amazon ECS clusters.

- Clusters are Region-specific.
- Clusters can contain tasks using both the Fargate and EC2 launch types. For more information about launch types, see Amazon ECS Launch Types (p. 50).
- You can create custom IAM policies for your clusters to allow or restrict user access to specific clusters. For more information, see the Clusters (p. 174) section in Amazon ECS IAM Policy Examples (p. 171).
- Clusters with Fargate tasks can be scaled using Service Auto Scaling. For more information, see Service Auto Scaling (p. 90).

Topics

- Creating a Cluster (p. 20)
- Deleting a Cluster (p. 21)

Creating a Cluster

You can create an Amazon ECS cluster using the AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 7).

> **Note**
> This cluster creation wizard provides a simple way to create the resources that are needed by an Amazon ECS cluster, and it lets you customize several common cluster configuration options. However, this wizard does not allow you to customize every resource option. If your requirements extend beyond what is supported in this wizard, consider using our reference architecture at https://github.com/awslabs/ecs-refarch-cloudformation.
> Do not attempt to modify the underlying resources directly after they are created by the wizard.

To create a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **Select cluster compatibility**, choose **Networking only**, then choose **Next Step**.
6. On the **Configure cluster** page, choose a **Cluster name**. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
7. In the **Networking** section, configure the VPC for your cluster. You can leave the default settings in or you can modify these settings by following the substeps below.

   a. (Optional) If you choose to create a new VPC, for **CIDR Block**, select a CIDR block for your VPC. For more information, see Your VPC and Subnets in the *Amazon VPC User Guide*.

   b. For **Subnets**, select the subnets to use for your VPC. You can keep the default settings or you can modify them to meet your needs.
8. Choose **Create**.

Deleting a Cluster

If you are finished using a cluster, you can delete it. When you delete a cluster in the Amazon ECS console, the associated resources that are deleted with it vary depending on how the cluster was created. Step 5 (p. 21) of the following procedure changes based on that condition.

If your cluster was created with the console first-run experience described in Getting Started with Amazon ECS using Fargate (p. 16) after November 24, 2015, or the cluster creation wizard described in Creating a Cluster (p. 20), then the AWS CloudFormation stack that was created for your cluster is also deleted when you 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 to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the cluster to delete.
5. Choose **Delete Cluster**. You see a confirmation prompt.

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:

- The Docker image to use with each container in your task
- How much CPU and memory to use with each task
- The launch type to use, which determines the infrastructure on which your tasks are hosted
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task should continue to run if the container finishes or fails
- The command the container should run when it is started
- Any data volumes that should be used with the containers in the task
- The IAM role that your tasks should use

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. 26).

Topics

- Task Definition Considerations (p. 22)
- Application Architecture (p. 26)
- Creating a Task Definition (p. 26)
- Task Definition Parameters (p. 31)
- Amazon ECS Launch Types (p. 50)
- Using Data Volumes in Tasks (p. 52)
- Task Networking with the awsvpc Network Mode (p. 54)
- Using the awslogs Log Driver (p. 55)
- Private Registry Authentication for Tasks (p. 61)
- Example Task Definitions (p. 64)
- Updating a Task Definition (p. 65)
- Deregistering Task Definitions (p. 66)

Task Definition Considerations

Tasks that use the Fargate launch type do not support all of the Amazon ECS task definition parameters that are available. Some parameters are not supported at all, and others behave differently for Fargate tasks.

The following task definition parameters are not valid in Fargate tasks:

- `devices`
- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`

- `dockerVolumeConfiguration`
- `extraHosts`
- `host`
- `hostname`
- `links`
- `placementConstraints` — By default, Fargate tasks are spread across Availability Zones.
- `privileged`
- `sharedMemorySize`
- `tmpfs`

> **Important**
> When any task definition parameter is not supported, it is assumed that any subflags for that parameter are not supported either.

The following task definition parameters behave differently for Fargate tasks:

- When using `logConfiguration`, the only supported `logDriver` for Fargate tasks is the `awslogs` log driver.
- When using `linuxParameters`, for `capabilities` the `drop` parameter can be used but the `add` parameter is not supported.
- The `healthCheck` parameter is only supported for Fargate tasks using platform version 1.1.0 or later.
- If you use the `'` parameter, you should only specify the `containerPort`. The `hostPort` can either be left blank or be set to the same value as the `containerPort`.

To ensure that your task definition validates for use with the Fargate launch type, you can specify the following when you register the task definition:

- In the AWS Management Console, for the **Requires Compatibilities** field, specify **FARGATE**.
- In the AWS CLI, for the `--requires-compatibilities` option, specify `FARGATE`.
- In the API, specify the `requiresCompatibilities` flag.

Network Mode

Fargate task definitions require that the network mode is set to `awsvpc`. The `awsvpc` network mode provides each task with its own elastic network interface. A network configuration is also required when creating a service or manually running tasks. For more information, see Task Networking with the `awsvpc` Network Mode (p. 54).

Task CPU and Memory

Fargate task definitions require that you specify CPU and memory at the task level. Although you can also specify CPU and memory at the container level for Fargate tasks, this is optional. Most use cases are satisfied by only specifying these resources at the task level. The table below shows the valid combinations of task-level CPU and memory.

CPU value	Memory value
256 (.25 vCPU)	0.5 GB, 1 GB, 2 GB
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB

CPU value	Memory value
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2048 (2 vCPU)	Between 4 GB and 16 GB in 1-GB increments
4096 (4 vCPU)	Between 8 GB and 30 GB in 1-GB increments

Logging

Fargate task definitions only support the `awslogs` log driver for the log configuration. This configures your Fargate tasks to send log information to Amazon CloudWatch Logs. The following shows a snippet of a task definition where the awslogs log driver is configured:

```
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group" : "/ecs/fargate-task-definition",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
}
```

For more information about using the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see Using the awslogs Log Driver (p. 55).

Amazon ECS Task Execution IAM Role

There is an optional task execution IAM role that you can specify with Fargate to allow your Fargate tasks to make API calls to Amazon ECR. The API calls pull container images as well as call CloudWatch to store container application logs. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

Example Task Definition

The following is an example task definition using the Fargate launch type that sets up a web server:

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' >  /usr/local/apache2/htdocs/index.html && httpd-foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
```

```
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

Task Storage

When provisioned, each Fargate task receives the following storage. Task storage is ephemeral. After a Fargate task stops, the storage is deleted.

- 10 GB of Docker layer storage
- An additional 4 GB for volume mounts. This can be mounted and shared among containers using the `volumes`, `mountPoints` and `volumesFrom` parameters in the task definition.

  Note
  The `host` and `sourcePath` parameters are not supported for Fargate tasks.

For more information about Amazon ECS default service limits, see Amazon ECS Service Limits (p. 211).

To provide nonpersistent empty storage for containers in a Fargate tasks

In this 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 volume with the name `database_scratch`.

```
"volumes": [
  {
    "name": "database_scratch",
    "host": {}
  }
]
```

2. In the `containerDefinitions` section, create the database container definitions so they mount the nonpersistent storage.

```
"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"
        }
      ]
    }
  ]
```

Application Architecture

How you architect your application on Amazon ECS depends on several factors, with the launch type you are using being a key differentiator. We give the following guidance which should assist in the process.

Using the Fargate Launch Type

When architecting your application using the Fargate launch type for your tasks, the main question is when should you put multiple containers into the same task definition versus deploying containers separately in multiple task definitions.

You should put multiple containers in the same task definition if:

- Containers share a common lifecycle (that is, they should be launched and terminated together).
- Containers are required to be run on the same underlying host (that is, one container references the other on a localhost port).
- You want your containers to share resources.
- Your containers share data volumes.

Otherwise, you should define your containers in separate tasks definitions so that you can scale, provision, and deprovision them separately.

Creating a Task Definition

Before you can run Docker containers on Amazon ECS, you must create a task definition. You can define multiple containers and data volumes in a task definition. For more information about the parameters available in a task definition, see Task Definition Parameters (p. 31).

To create a new task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Task Definitions**, **Create new Task Definition**.

3. On the **Select launch type compatibilities** page, choose **FARGATE**, **Next step**.

   Note
   The Fargate launch type is not compatible with Windows containers.
4. (Optional) If you have a JSON representation of your task definition, complete the following steps:

   a. On the **Configure task and container definitions** page, scroll to the bottom of the page and choose **Configure via JSON**.
   b. Paste your task definition JSON into the text area and choose **Save**.
   c. Verify your information and choose **Create**.

   Scroll to the bottom of the page and choose **Configure via JSON**.
5. For **Task Definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. For **Task execution IAM role**, either select your task execution role or choose **Create new role** so that the console can create one for you. For more information, seeAmazon ECS Task Execution IAM Role (p. 156).
7. For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. The table below shows the valid combinations.

CPU value	Memory value
256 (.25 vCPU)	512 MB, 1 GB, 2 GB
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments

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

   a. Choose **Add container**.
   b. Fill out each required field and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see Task Definition Parameters (p. 31).
   c. Choose **Add** to add your container to the task definition.
9. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see Using Data Volumes in Tasks (p. 52).

   - For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
10. In the **Tags** section, specify the key and value for each tag to associate with the task definition. For more information, see Tagging Your Amazon ECS Resources.
11. Choose **Create**.

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, see Task Definition Parameters (p. 31).

```
{
    "family": "",
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "host",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "repositoryCredentials": {
                "credentialsParameter": ""
            },
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [
                ""
            ],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                ""
            ],
            "command": [
                ""
            ],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [
                        ""
                    ],
                    "drop": [
                        ""
                    ]
                },
                "devices": [
                    {
                        "hostPath": "",
                        "containerPath": "",
                        "permissions": [
```

```
                "mknod"
            ]
        }
    ],
    "initProcessEnabled": true,
    "sharedMemorySize": 0,
    "tmpfs": [
        {
            "containerPath": "",
            "size": 0,
            "mountOptions": [
                ""
            ]
        }
    ]
},
"inferenceDevices": [
    ""
],
"secrets": [
    {
        "name": "",
        "valueFrom": ""
    }
],
"hostname": "",
"user": "",
"workingDirectory": "",
"disableNetworking": true,
"privileged": true,
"readonlyRootFilesystem": true,
"dnsServers": [
    ""
],
"dnsSearchDomains": [
    ""
],
"extraHosts": [
    {
        "hostname": "",
        "ipAddress": ""
    }
],
"dockerSecurityOptions": [
    ""
],
"interactive": true,
"pseudoTerminal": true,
"dockerLabels": {
    "KeyName": ""
},
"ulimits": [
    {
        "name": "sigpending",
        "softLimit": 0,
        "hardLimit": 0
    }
],
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "KeyName": ""
    }
},
"healthCheck": {
    "command": [
```

```
                    ""
                ],
                "interval": 0,
                "timeout": 0,
                "retries": 0,
                "startPeriod": 0
            },
            "systemControls": [
                {
                    "namespace": "",
                    "value": ""
                }
            ],
            "resourceRequirements": [
                {
                    "value": "",
                    "type": "GPU"
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {
                "sourcePath": ""
            },
            "dockerVolumeConfiguration": {
                "scope": "shared",
                "autoprovision": true,
                "driver": "",
                "driverOpts": {
                    "KeyName": ""
                },
                "labels": {
                    "KeyName": ""
                }
            }
        }
    ],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": ""
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "",
    "memory": "",
    "inferenceAccelerators": [
        {
            "deviceName": "",
            "deviceType": "",
            "devicePolicy": ""
        }
    ],
    "pidMode": "host",
    "ipcMode": "none",
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ]
```

```
}
```

You can generate this task definition template using the following AWS CLI command:

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

Task Definition Parameters

Task definitions are split into separate parts: the task family, the IAM task role, the network mode, container definitions, volumes, task placement constraints, and launch types. The family is the name of the task, and each family can have multiple revisions. The IAM task role specifies the permissions that containers in the task should have. The network mode determines how the networking is configured for your containers. 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 task placement constraints customize how your tasks are placed within the infrastructure. The launch type determines which infrastructure your tasks use.

The family and container definitions are required in a task definition, while task role, network mode, volumes, task placement constraints, and launch type are optional.

Parts

- Family (p. 31)
- Task Execution Role (p. 31)
- Network Mode (p. 32)
- Container Definitions (p. 32)
- Volumes (p. 46)
- Launch Types (p. 47)
- Task Size (p. 47)
- Other Task Definition Parameters (p. 49)

Family

`family`

Type: string

Required: yes

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

Task Execution Role

`executionRoleArn`

Type: string

Required: no

When you register a task definition, you can provide a task execution role that allows the containers in the task to pull container images and publish container logs to CloudWatch on your behalf. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

Network Mode

`networkMode`

Type: string

Required: no

The Docker networking mode to use for the containers in the task. When using the Fargate launch type, the `awsvpc` network mode is required.

When the network mode is `awsvpc`, the task is allocated an elastic network interface, and you must specify a `NetworkConfiguration` when you create a service or run a task with the task definition. For more information, see Task Networking with the `awsvpc` Network Mode (p. 54).

The `awsvpc` network mode offers the highest networking performance for containers because they use the Amazon EC2 network stack. Exposed container ports are mapped directly to the attached elastic network interface port, so you cannot take advantage of dynamic host port mappings.

Container Definitions

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

Topics

- Standard Container Definition Parameters (p. 32)
- Advanced Container Definition Parameters (p. 35)
- Other Container Definition Parameters (p. 43)

Standard Container Definition Parameters

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

`name`

Type: string

Required: yes

The name of a container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. 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. This parameter maps to `name` in the Create a container section of the Docker Remote API and the `--name` option to **docker run**.

`image`

Type: string

Required: yes

The image used to start a container. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. You can also specify other repositories with either

repository-url/*image*:*tag* or *repository-url*/*image*@*digest*. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to `Image` in the Create a container section of the Docker Remote API and the `IMAGE` parameter of **docker run**.

- When a new task starts, the Amazon ECS container agent pulls the latest version of the specified image and tag for the container to use. However, subsequent updates to a repository image are not propagated to already running tasks.
- Images in private registries are supported. For more information, see Private Registry Authentication for Tasks (p. 61).
- Images in Amazon ECR repositories can be specified by using either the full `registry/repository:tag` or `registry/repository@digest` naming convention. For example, *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* or *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*
- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

`memory`

Type: integer

Required: no

The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to `Memory` in the Create a container section of the Docker Remote API and the `--memory` option to **docker run**.

If your containers are part of a task using the Fargate launch type, this field is optional and the only requirement is that the total amount of memory reserved for all containers within a task be lower than the task `memory` value.

The Docker daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

`memoryReservation`

Type: integer

Required: no

The soft limit (in MiB) of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory to this soft limit; however, your container can consume more memory when needed, up to either the hard limit specified with the `memory` parameter (if applicable), or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the Create a container section of the Docker Remote API and the `--memory-reservation` option to **docker run**.

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

For example, if your container normally uses 128 MiB of memory, but occasionally bursts to 256 MiB of memory for short periods of time, you can set a `memoryReservation` of 128 MiB, and a `memory` hard limit of 300 MiB. This configuration would allow the container to only reserve 128 MiB

of memory from the remaining resources on the container instance, but also allow the container to consume more memory resources when needed.

The Docker daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

`portMappings`

Type: object array

Required: no

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

For task definitions that use the `awsvpc` network mode, you should only specify the `containerPort`. The `hostPort` can be left blank or it must be the same value as the `containerPort`.

This parameter maps to `PortBindings` in the Create a container section of the Docker Remote API and the `--publish` option to **docker run**. If the network mode of a task definition is set to `host`, then host ports must either be undefined or they must match the container port in the port mapping.

> **Note**
> After a task reaches the `RUNNING` status, manual and automatic host and container port assignments are visible in the following locations:
> - Console: The **Network Bindings** section of a container description for a selected task.
> - AWS CLI: The `networkBindings` section of the **describe-tasks** command output.
> - API: The `DescribeTasks` response.

`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 using containers in a task with the Fargate launch type, exposed ports should be specified using `containerPort`.

If using containers in a task with the EC2 launch type and you specify a container port and not a host port, your container automatically receives a host port in the ephemeral port range. For more information, see `hostPort`. Port mappings that are automatically assigned in this way do not count toward the 100 reserved ports limit of a container instance.

`hostPort`

Type: integer

Required: no

The port number on the container instance to reserve for your container.

If using containers in a task with the Fargate launch type, the `hostPort` can either be left blank or be the same value as `containerPort`.

If using containers in a task with the EC2 launch type, you can specify a non-reserved host port for your container port mapping (this is referred to as *static* host port mapping), or you can omit the `hostPort` (or set it to `0`) while specifying a `containerPort` and your container automatically receives a port (this is referred to as *dynamic* host port mapping) in the ephemeral port range for your container instance operating system and Docker version.

The default ephemeral port range is `49153-65535`, and this range is used for Docker versions before 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 32768–61000 on the latest Amazon ECS-optimized AMI); if this kernel parameter is unavailable, the default ephemeral port range is used. Do not attempt to specify a host port in the ephemeral port range, as these are reserved for automatic assignment. In general, ports below 32768 are outside of the ephemeral port range.

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

`protocol`

Type: string

Required: no

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

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

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

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

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

Advanced Container Definition Parameters

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

Topics

- Health Check (p. 36)
- Environment (p. 37)
- Network Settings (p. 39)
- Storage and Logging (p. 40)
- Security (p. 42)
- Resource Limits (p. 42)
- Docker Labels (p. 43)

Health Check

`healthCheck`

The health check command and associated configuration parameters for the container. This parameter maps to `HealthCheck` in the Create a container section of the Docker Remote API and the `HEALTHCHECK` parameter of docker run.

> **Note**
> The Amazon ECS container agent only monitors and reports on the health checks specified in the task definition. Amazon ECS does not monitor Docker health checks that are embedded in a container image and not specified in the container definition. Health check parameters that are specified in a container definition override any Docker health checks that exist in the container image.

Task health is reported by the `healthStatus` of the task, which is determined by the health of the essential containers in the task. If all essential containers in the task are reporting as `HEALTHY`, then the task status also reports as `HEALTHY`. If any essential containers in the task are reporting as `UNHEALTHY` or `UNKNOWN`, then the task status also reports as `UNHEALTHY` or `UNKNOWN`, accordingly. If a service's task reports as unhealthy, it is removed from a service and replaced.

The following are notes about container health check support:

- Container health checks are supported for Fargate tasks if you are using platform version 1.1.0 or later. For more information, see AWS Fargate Platform Versions (p. 19).
- Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer.

`command`

A string array representing the command that the container runs to determine if it is healthy. The string array can start with `CMD` to execute the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used by default.

In the console, example input for a health check could be:

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

Similarly, in the console JSON panel, the AWS CLI, or the APIs, example input for a health check could be:

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

An exit code of 0 indicates success, and a non-zero exit code indicates failure. For more information, see `HealthCheck` in the Create a container section of the Docker Remote API.

`interval`

The time period in seconds between each health check execution. You may specify between 5 and 300 seconds. The default value is 30 seconds.

`timeout`

The time period in seconds to wait for a health check to succeed before it is considered a failure. You may specify between 2 and 60 seconds. The default value is 5 seconds.

`retries`

The number of times to retry a failed health check before the container is considered unhealthy. You may specify between 1 and 10 retries. The default value is three retries.

`startPeriod`

The optional grace period within which to provide containers time to bootstrap before failed health checks count towards the maximum number of retries. You may specify between 0 and 300 seconds. The `startPeriod` is disabled by default.

Environment

`cpu`

Type: integer

Required: no

The number of `cpu` units the Amazon ECS container agent will reserve for the container. This parameter maps to `CpuShares` in the Create a container section of the Docker Remote API and the `--cpu-shares` option to **docker run**.

This field is optional for tasks using the Fargate launch type, and the only requirement is that the total amount of CPU reserved for all containers within a task be lower than the task-level `cpu` value.

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

Linux containers share unallocated CPU units with other containers on the container instance with the same ratio as their allocated amount. 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.

On Linux container instances, 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 allows 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 two CPU shares.
- **Agent versions >= 1.2.0:** Null, zero, and CPU values of 1 are passed to Docker as two CPU shares.

On Windows container instances, the CPU limit is enforced as an absolute limit, or a quota. Windows containers only have access to the specified amount of CPU that is described in the task definition.

`essential`

Type: Boolean

Required: no

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

All tasks must have at least one essential container. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see Application Architecture (p. 26).

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

`entryPoint`

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

Type: string array

Required: no

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

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

`command`

Type: string array

Required: no

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

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

`workingDirectory`

Type: string

Required: no

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

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

`environment`

Type: object array

Required: no

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

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

`name`

Type: string

Required: yes, when `environment` is used

The name of the environment variable.

`value`

Type: string

Required: yes, when `environment` is used

The value of the environment variable.

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

`secrets`

Type: object array

Required: no

The secrets to pass to the container as environment variables.

`name`

Type: string

Required: yes, when `environment` is used

The value to set as the environment variable on the container.

`valueFrom`

Type: string

Required: yes, when `environment` is used

The secret to expose to the container. Supported values are either the full ARN or the name of the parameter in the AWS Systems Manager Parameter Store.

> **Note**
> If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching then you can use either the full ARN or name of the secret. If the parameter exists in a different Region then the full ARN must be specified.

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

Network Settings

`dnsServers`

Type: string array

Required: no

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

> **Note**
> This parameter is not supported for Windows containers.

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

Storage and Logging

`readonlyRootFilesystem`

Type: Boolean

Required: no

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

> **Note**
> This parameter is not supported for Windows containers.

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

`mountPoints`

Type: Object

Required: No

The mount points for data volumes in your container.

This parameter maps to `Volumes` in the Create a container section of the Docker Remote API and the `--volume` option to **docker run**.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives.

`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 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`.

`volumesFrom`

Type: object array

Required: no

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

`sourceContainer`

Type: string

Required: yes, when `volumesFrom` is used

The name of the container to mount volumes from.

`readOnly`

Type: Boolean

Required: no

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

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

Type: LogConfiguration object

Required: no

The log configuration specification for the container.

If using the Fargate launch type, the only supported value is `awslogs`. For more information on using the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see Using the awslogs Log Driver (p. 55).

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

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

Note

```
"logConfiguration": {
      "logDriver": "json-file"|"syslog"|"journald"|"gelf"|"fluentd"|"awslogs"|"splunk",
      "options": {"string": "string"
```

```
        ...}
```

`logDriver`

Type: string

Valid values: `"json-file" | "syslog" | "journald" | "gelf" | "fluentd" | "awslogs" | "splunk"`

Required: yes, when `logConfiguration` is used

The log driver to use for the container. The valid values listed earlier are log drivers that the Amazon ECS container agent can communicate with by default.

If using the Fargate launch type, the only supported value is `awslogs`.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

`options`

Type: string to string map

Required: no

The configuration options to send to the log driver.

This parameter requires version 1.19 of the Docker Remote API or greater on your container instance.

Security

`user`

Type: string

Required: no

The user name to use inside the container. This parameter maps to `User` in the Create a container section of the Docker Remote API and the `--user` option to **docker run**.

Note
This parameter is not supported for Windows containers.

```
"user": "string"
```

Resource Limits

`ulimits`

Type: object array

Required: no

A list of `ulimits` to set in the container. This parameter maps to `Ulimits` in the Create a container section of the Docker Remote API and the `--ulimit` option to **docker run**.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

Note
This parameter is not supported for Windows containers.

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtprio"|"r
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

`name`

Type: string

Valid values: `"core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"`

Required: yes, when `ulimits` are used

The `type` of the `ulimit`.

`hardLimit`

Type: integer

Required: yes, when `ulimits` are used

The hard limit for the `ulimit` type.

`softLimit`

Type: integer

Required: yes, when `ulimits` are used

The soft limit for the `ulimit` type.

Docker Labels

`dockerLabels`

Type: string to string map

Required: no

A key/value map of labels to add to the container. This parameter maps to `Labels` in the Create a container section of the Docker Remote API and the `--label` option to **docker run**.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

```
"dockerLabels": {"string": "string"
      ...}
```

Other Container Definition Parameters

The following container definition parameters are able to be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating a Task Definition (p. 26).

Topics

- Linux Parameters (p. 44)
- System Controls (p. 45)
- Interactive (p. 46)
- Pseudo Terminal (p. 46)

Linux Parameters

`linuxParameters`

Type: LinuxParameters object

Required: no

Linux-specific options that are applied to the container, such as KernelCapabilities.

> **Note**
> This parameter is not supported for Windows containers.

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

Type: KernelCapabilities object

Required: no

The Linux capabilities for the container that are dropped from the default configuration provided by Docker. For more information about the default capabilities and the non-default available capabilities, see Runtime privilege and Linux capabilities in the *Docker run reference*. For more detailed information about these Linux capabilities, see the capabilities(7) Linux manual page.

`drop`

Type: string array

Valid values: `"ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"`

Required: no

The Linux capabilities for the container to remove from the default configuration provided by Docker. This parameter maps to `CapDrop` in the Create a container section of the Docker Remote API and the `--cap-drop` option to docker run.

`initProcessEnabled`

Run an `init` process inside the container that forwards signals and reaps processes. This parameter maps to the `--init` option to docker run.

This parameter requires version 1.25 of the Docker Remote API or greater on your container instance.

System Controls

`systemControls`

Type: SystemControl object

Required: no

A list of namespaced kernel parameters to set in the container. This parameter maps to `Sysctls` in the Create a container section of the Docker Remote API and the `--sysctl` option to **docker run**.

It is not recommended that you specify network-related `systemControls` parameters for multiple containers in a single task that also uses either the `awsvpc` or `host` network mode for the following reasons:

- For tasks that use the `awsvpc` network mode, if you set `systemControls` for any container it will apply to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that is started last will determine which `systemControls` take effect.
- For tasks that use the `host` network mode, the network namespace `systemControls` are not supported.

If you are setting an IPC resource namespace to use for the containers in the task, the following will apply to your system controls. For more information, see IPC Mode (p. 49).

- For tasks that use the `host` IPC mode, IPC namespace `systemControls` are not supported.
- For tasks that use the `task` IPC mode, IPC namespace `systemControls` values will apply to all containers within a task.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

`namespace`

Type: String

Required: no

The namespaced kernel parameter to set a `value` for.

Valid IPC namespace values: `"kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"`, as well as Sysctls beginning with `"fs.mqueue.*"`

Valid network namespace values: Sysctls beginning with `"net.*"`

`value`

Type: String

Required: no

The value for the namespaced kernel parameter specifed in `namespace`.

Interactive

`interactive`

Type: Boolean

Required: no

When this parameter is `true`, this allows you to deploy containerized applications that require stdin or a tty to be allocated. This parameter maps to `OpenStdin` in the Create a container section of the Docker Remote API and the `--interactive` option to **docker run**.

Pseudo Terminal

`pseudoTerminal`

Type: Boolean

Required: no

When this parameter is `true`, a TTY is allocated. This parameter maps to `Tty` in the Create a container section of the Docker Remote API and the `--tty` option to **docker run**.

Volumes

When you register a task definition, you can optionally specify a list of volumes to be passed to the Docker daemon on a container instance, which then becomes available for access by other containers on the same container instance.

For more information, see Using Data Volumes in Tasks (p. 52).

The following parameters are allowed in a container definition:

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This name is referenced in the `sourceVolume` parameter of container definition `mountPoints`.

`host`

Required: No

This parameter is specified when using bind mounts. To use Docker volumes, specify a `dockerVolumeConfiguration` instead. The contents of the `host` parameter determine whether your bind mount 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.

Bind mount host volumes are supported when using either the EC2 or Fargate launch types.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives. For example, you can mount `C:\my\path:C:\my\path` and `D:\:D:\`, but not `D:\my\path:C:\my\path` or `D:\:C:\my\path`.

`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon has assigned 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.

Launch Types

When you register a task definition, you specify the launch type to use for your task. For more information, see Amazon ECS Launch Types (p. 50).

The following parameter is allowed in a task definition:

`requiresCompatibilities`

Type: string array

Required: no

Valid Values: `EC2` | `FARGATE`

The launch type the task is using. This enables a check to ensure that all of the parameters used in the task definition meet the requirements of the launch type.

Valid values are `FARGATE` and `EC2`. For more information about launch types, see Amazon ECS Launch Types (p. 50).

Task Size

When you register a task definition, you can specify the total cpu and memory used for the task. This is separate from the `cpu` and `memory` values at the container definition level. If using the EC2 launch type, these fields are optional. If using the Fargate launch type, these fields are required and there are specific values for both `cpu` and `memory` that are supported.

Note
Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

The following parameter is allowed in a task definition:

`cpu`

Type: string

Required: no

Note
This parameter is not supported for Windows containers.

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example `1024`, or as a string using vCPUs, for example `1 vCPU` or `1 vcpu`, in a task definition. When the task definition is registered, a vCPU value is converted to an integer indicating the CPU units.

If using the EC2 launch type, this field is optional. Supported values are between `128` CPU units (`0.125` vCPUs) and `10240` CPU units (`10` vCPUs).

If using the Fargate launch type, this field is required and you must use one of the following values, which determines your range of supported values for the `memory` parameter:

CPU value	Memory value (MiB)
256 (.25 vCPU)	512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)
512 (.5 vCPU)	1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)
1024 (1 vCPU)	2048 (2 GB), 3072 (3 GB), 4096 (4 GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)
2048 (2 vCPU)	Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)
4096 (4 vCPU)	Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)

`memory`

Type: string

Required: no

Note
This parameter is not supported for Windows containers.

The amount of memory (in MiB) used by the task. It can be expressed as an integer using MiB, for example `1024`, or as a string using GB, for example `1GB` or `1 GB`, in a task definition. When the task definition is registered, a GB value is converted to an integer indicating the MiB.

If using the EC2 launch type, this field is optional.

If using the Fargate launch type, this field is required and you must use one of the following values, which determines your range of supported values for the `cpu` parameter:

Memory value (MiB)	CPU value
512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)	256 (.25 vCPU)
1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)	512 (.5 vCPU)
2048 (2 GB), 3072 (3 GB), 4096 (4GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)	1024 (1 vCPU)

Memory value (MiB)	CPU value
Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)	2048 (2 vCPU)
Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)	4096 (4 vCPU)

Other Task Definition Parameters

The following task definition parameters are able to be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating a Task Definition (p. 26).

Topics

- IPC Mode (p. 49)
- PID Mode (p. 49)

IPC Mode

`ipcMode`

Type: String

Required: No

The IPC resource namespace to use for the containers in the task. The valid values are `host`, `task`, or `none`. If `host` is specified, then all containers within the tasks that specified the `host` IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same IPC resources. If `none` is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance. For more information, see IPC settings in the *Docker run reference*.

If the `host` IPC mode is used, be aware that there is a heightened risk of undesired IPC namespace expose. For more information, see Docker security.

If you are setting namespaced kernel parameters using `systemControls` for the containers in the task, the following will apply to your IPC resource namespace. For more information, see System Controls (p. 45).

- For tasks that use the `host` IPC mode, IPC namespace related `systemControls` are not supported.
- For tasks that use the `task` IPC mode, IPC namespace related `systemControls` will apply to all containers within a task.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

PID Mode

`pidMode`

Type: String

Required: No

The process namespace to use for the containers in the task. The valid values are `host` or `task`. If `host` is specified, then all containers within the tasks that specified the `host` PID mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same process namespace. If no value is specified, the default is a private namespace. For more information, see PID settings in the *Docker run reference*.

If the `host` PID mode is used, be aware that there is a heightened risk of undesired process namespace expose. For more information, see Docker security.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

Amazon ECS Launch Types

An Amazon ECS launch type determines the type of infrastructure on which your tasks and services are hosted.

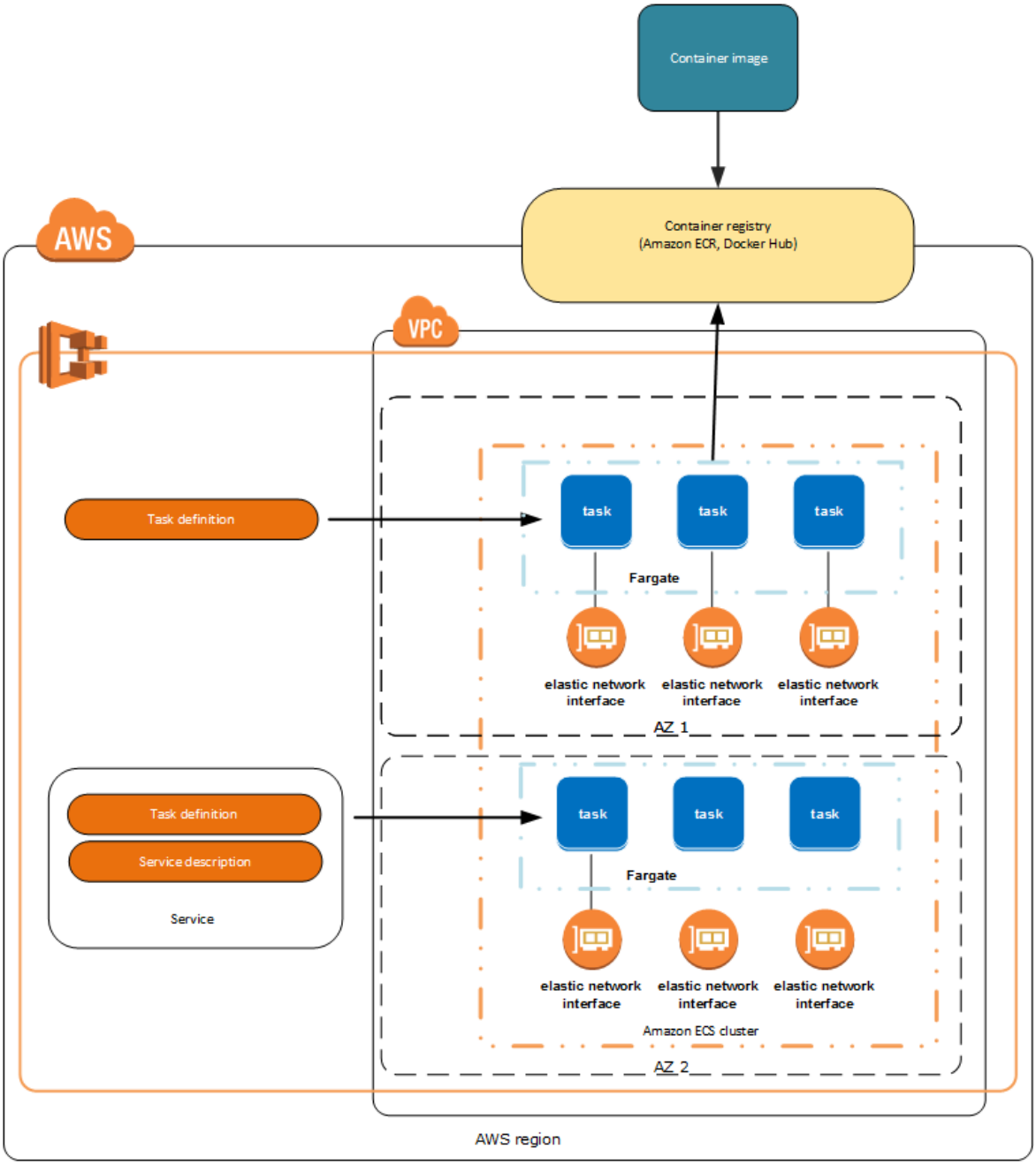
Fargate Launch Type

The Fargate launch type allows you to run your containerized applications without the need to provision and manage the backend infrastructure. Just register your task definition and Fargate launches the container for you.

The AWS Fargate launch type is currently available in the following Regions:

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

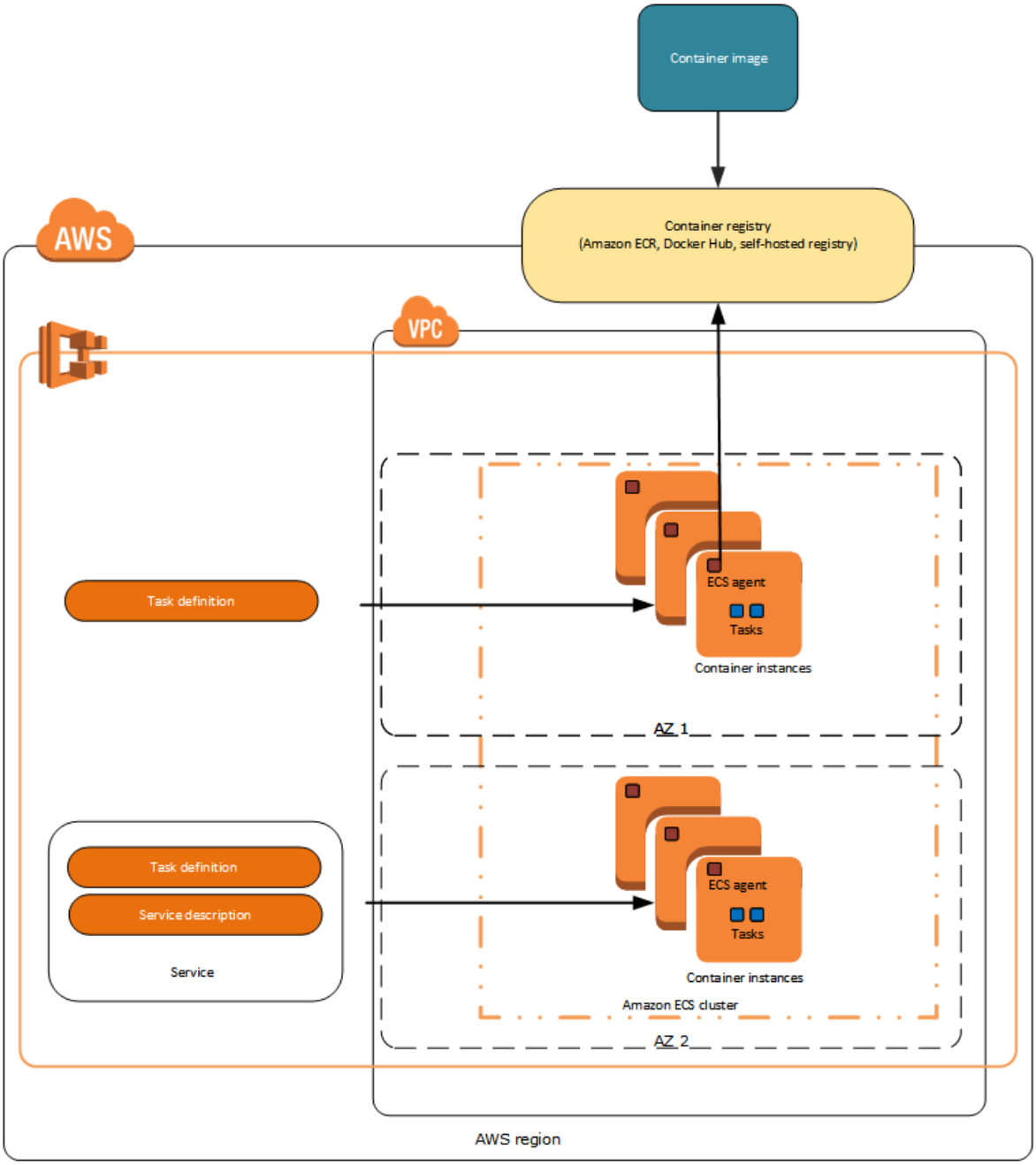
This diagram shows the general architecture:



EC2 Launch Type

The EC2 launch type allows you to run your containerized applications on a cluster of Amazon EC2 instances that you manage.

This diagram shows the general architecture:



Using Data Volumes in Tasks

- To provide persistent data volumes for use with a container
- To define an empty, nonpersistent data volume and mount it on multiple containers
- To share defined data volumes at different locations on different containers on the same container instance
- To provide a data volume to your task that is managed by a third-party volume driver

The lifecycle of the volume can be tied to either a specific task or to the lifecycle of a specific container instance.

When provisioned, each Fargate task receives the following storage. Task storage is ephemeral. After a Fargate task stops, the storage is deleted.

- 10 GB of Docker layer storage
- An additional 4 GB for volume mounts. This can be mounted and shared among containers using the `volumes`, `mountPoints` and `volumesFrom` parameters in the task definition.

  Note
  The `host` and `sourcePath` parameters are not supported for Fargate tasks.

For more information about Amazon ECS default service limits, see Amazon ECS Service Limits (p. 211).

To provide nonpersistent empty storage for containers in a Fargate tasks

In this 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 volume with the name `database_scratch`.

```
"volumes": [
  {
    "name": "database_scratch",
    "host": {}
  }
]
```

2. In the `containerDefinitions` section, create the database container definitions so they mount the nonpersistent storage.

```
"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"
      }
    ]
  }
]
```

Task Networking with the `awsvpc` Network Mode

The task networking features provided by the `awsvpc` network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. When you use the `awsvpc` network mode in your task definitions, every task that is launched from that task definition gets its own elastic network interface, a primary private IP address, and an internal DNS hostname. The task networking feature simplifies container networking and gives you more control over how containerized applications communicate with each other and other services within your VPCs.

Task networking also provides greater security for your containers by allowing you to use security groups and network monitoring tools at a more granular level within ECS tasks. Because each task gets its own elastic network interface, you can also take advantage of other Amazon EC2 networking features like VPC Flow Logs so that you can monitor traffic to and from your tasks. Additionally, containers that belong to the same task can communicate over the `localhost` interface. A task can only have one elastic network interface associated with it at a given time.

To use task networking, specify the `awsvpc` network mode in your task definition. Then, when you run a task or create a service, specify a network configuration that includes the subnets in which to place your tasks and the security groups to attach to its associated elastic network interface. The Fargate tasks are then launched in those subnets and the specified security groups are associated with the elastic network interface that is provisioned for the task.

The elastic network interface that is created for your task is fully managed by Amazon ECS. The task sends and receives network traffic on the elastic network interface in the same way that Amazon EC2 instances do with their primary network interfaces. These elastic network interfaces are visible in the Amazon EC2 console for your account, but they cannot be detached manually or modified by your account. This is to prevent accidental deletion of an elastic network interface that is associated with a running task. You can view the elastic network interface attachment information for tasks in the Amazon ECS console or with the DescribeTasks API operation. When the task stops or if the service is scaled down, the elastic network interface is released.

Enabling Task Networking

Fargate tasks require the use of the `awsvpc` network mode so task networking is enabled by default. For more information, see Network Mode (p. 32). When you run tasks or create services using a task definition that specifies the `awsvpc` network mode, you specify a network configuration that contains the VPC subnets to be considered for placement and the security groups to attach to the task's elastic network interface.

Tasks and services that use the `awsvpc` network mode require the Amazon ECS service-linked role to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service in the AWS Management Console. For more information, see Using Service-Linked Roles for Amazon ECS (p. 157). You can also create the service-linked role with the following AWS CLI command:

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

Task Networking Considerations

There are several things to consider when using task networking.

- Fargate tasks are able to be configured to receive public IP addresses.
- There is a limit of 10 subnets and 5 security groups that are able to be specified in the `awsvpcConfiguration` section of a task definition.
- The elastic network interfaces that are created and attached by Amazon ECS cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an elastic network

interface that is associated with a running task. To release the elastic network interfaces for a task, stop the task.

- When a task is started with the `awsvpc` network mode, the Amazon ECS container agent creates an additional `pause` container for each task before starting the containers in the task definition. It then configures the network namespace of the `pause` container by executing the amazon-ecs-cni-plugins CNI plugins. The agent then starts the rest of the containers in the task so that they share the network stack of the `pause` container. This means that all containers in a task are addressable by the IP addresses of the elastic network interface, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode (for example, those with the Fargate launch type) only support Application Load Balancers and Network Load Balancers; Classic Load Balancers are not supported. Also, when you create any target groups for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance. For more information, see Service Load Balancing (p. 81).

Using the awslogs Log Driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. This allows you to view the logs from the containers in your Fargate tasks.This topic helps you get started using the `awslogs` log driver in your task definitions.

> **Note**
> The type of information that is logged by your task's containers depends mostly on their `ENTRYPOINT` command. By default, the logs that are captured show the command output that you would normally see in an interactive terminal if you ran the container locally, which are the `STDOUT` and `STDERR` I/O streams. The `awslogs` log driver simply passes these logs from Docker to CloudWatch. For more information on how Docker logs are processed, including alternative ways to capture different file data or streams, see View logs for a container or service in the Docker documentation.

Topics

- Enabling the awslogs Log Driver for Your Containers (p. 55)
- Creating Your Log Groups (p. 55)
- Available awslogs Log Driver Options (p. 56)
- Specifying a Log Configuration in your Task Definition (p. 58)
- Viewing awslogs Container Logs in CloudWatch Logs (p. 59)

Enabling the awslogs Log Driver for Your Containers

If you are using the Fargate launch type for your tasks, all you need to do to enable the `awslogs` log driver is add the required `logConfiguration` parameters to your task definition. This configures your Fargate tasks to send log information to Amazon CloudWatch Logs. For more information, see Specifying a Log Configuration in your Task Definition (p. 58).

Creating Your Log Groups

The `awslogs` log driver can send log streams to existing log groups in CloudWatch Logs, but it cannot create log groups. Before you launch any tasks that use the `awslogs` log driver, you should ensure the log groups that you intend your containers to use are created. The console provides an auto-configure option. If you register your task definitions in the console and choose the **Auto-configure CloudWatch Logs** option, your log groups are created for you. Alternatively, you can manually create your log groups using the following steps.

As an example, you could have a task with a WordPress container (which uses the `awslogs-wordpress` log group) that is linked to a MySQL container (which uses the `awslogs-mysql` log group). The sections below show how to create these log groups with the AWS CLI and with the CloudWatch console.

Creating a Log Group with the AWS CLI

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information, see the AWS Command Line Interface User Guide.

If you have a working installation of the AWS CLI, you can use it to create your log groups. The command below creates a log group called `awslogs-wordpress` in the `us-west-2` region. Run this command for each log group to create, replacing the log group name with your value and region name to the desired log destination.

```
aws logs create-log-group --log-group-name awslogs-wordpress --region us-west-2
```

Using the Auto-configuration Feature to Create a Log Group

When registering a task definition in the Amazon ECS console, you have the option to allow Amazon ECS to auto-configure your CloudWatch logs. The option also creates the specified log groups for you. To make it easy, the auto-configuration option sets up the CloudWatch logs and log groups with the specified prefix.

To create a log group in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the left navigation pane, choose **Task Definitions**, **Create new Task Definition**.
3. Select your compatibility option and choose **Next Step**.
4. Choose **Add container**.
5. In the **Storage and Logging** section, for **Log configuration**, choose **Auto-configure CloudWatch Logs**.
6. Enter your awslogs log driver options. For more information, see Specifying a Log Configuration in your Task Definition (p. 58).
7. Complete the rest of the task definition wizard.

Creating a Log Group with the CloudWatch Console

The following procedure creates a log group in the CloudWatch console.

To create a log group in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**.
3. Choose **Actions**, **Create log group**.
4. For **Log Group Name**, enter the name of the log group to create.
5. Choose **Create log group** to finish.

Available awslogs Log Driver Options

The `awslogs` log driver supports the following options in Amazon ECS task definitions. For more information, see CloudWatch Logs logging driver.

`awslogs-create-group`

Required: No

Specify whether you want the log group automatically created. If this option is not specified, it defaults to `false`.

> **Note**
> Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

`awslogs-datetime-format`

Required: No

This option defines a multiline start pattern in Python `strftime` format. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. Thus the matched line is the delimiter between log messages.

One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

This option always takes precedence if both `awslogs-datetime-format` and `awslogs-multiline-pattern` are configured.

> **Note**
> Multiline logging performs regular expression parsing and matching of all log messages, which may have a negative impact on logging performance.

`awslogs-region`

Required: Yes

Specify the region to which the `awslogs` log driver should send your Docker logs. You can choose to send all of your logs from clusters in different regions to a single region in CloudWatch Logs so that they are all visible in one location, or you can separate them by region for more granularity. Be sure that the specified log group exists in the region that you specify with this option.

`awslogs-group`

Required: Yes

You must specify a log group to which the `awslogs` log driver sends its log streams. For more information, see Creating Your Log Groups (p. 55).

`awslogs-multiline-pattern`

Required: No

This option defines a multiline start pattern using a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. Thus the matched line is the delimiter between log messages.

This option is ignored if `awslogs-datetime-format` is also configured.

> **Note**
> Multiline logging performs regular expression parsing and matching of all log messages. This may have a negative impact on logging performance.

`awslogs-stream-prefix`

Required: Yes, when using the Fargate launch type.

The `awslogs-stream-prefix` option allows you to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task to which the container belongs. If you specify a prefix with this option, then the log stream takes the following format:

```
prefix-name/container-name/ecs-task-id
```

For Amazon ECS services, you could use the service name as the prefix, which would allow you to trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task to which the container belongs.

Specifying a Log Configuration in your Task Definition

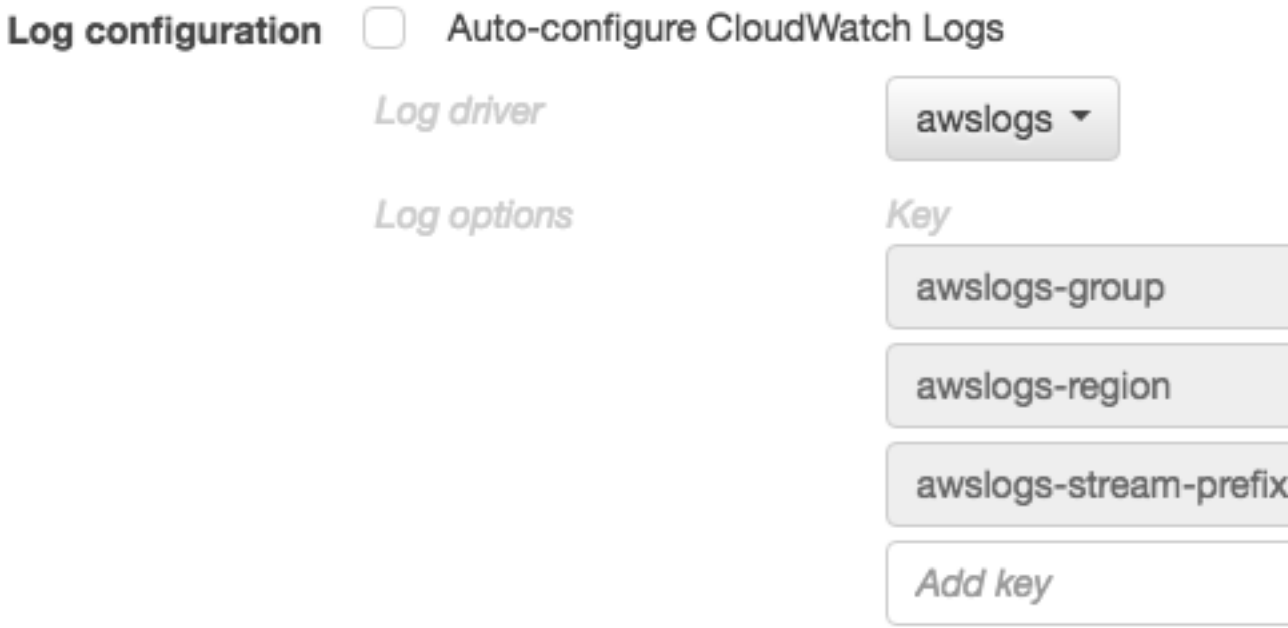
Before your containers can send logs to CloudWatch, you must specify the `awslogs` log driver for containers in your task definition. This section describes the log configuration for a container to use the `awslogs` log driver. For more information, see Creating a Task Definition (p. 26).

The task definition JSON shown below has a `logConfiguration` object specified for each container; one for the WordPress container that sends logs to a log group called `awslogs-wordpress`, and one for a MySQL container that sends logs to a log group called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```
{
    "containerDefinitions": [
        {
            "name": "wordpress",
            "links": [
                "mysql"
            ],
            "image": "wordpress",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "awslogs-wordpress",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
                }
            ],
            "name": "mysql",
            "image": "mysql",
            "cpu": 10,
            "memory": 500,
```

```
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            }
        }
    ],
    "family": "awslogs-example"
}
```

In the Amazon ECS console, the log configuration for the `wordpress` container is specified as shown in the image below.

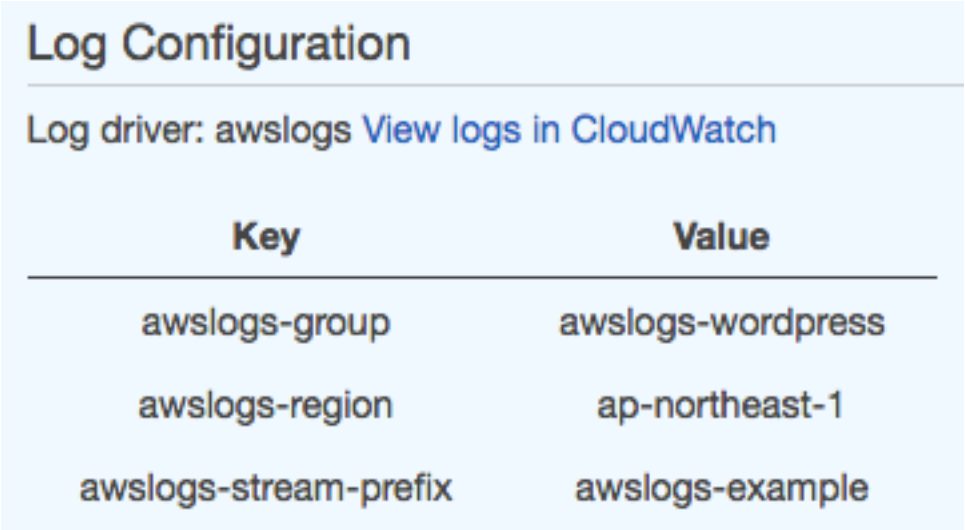


Viewing awslogs Container Logs in CloudWatch Logs

After your Fargate tasks that use the `awslogs` log driver have launched, your configured containers should be sending their log data to CloudWatch Logs. You can view and search these logs in the console.

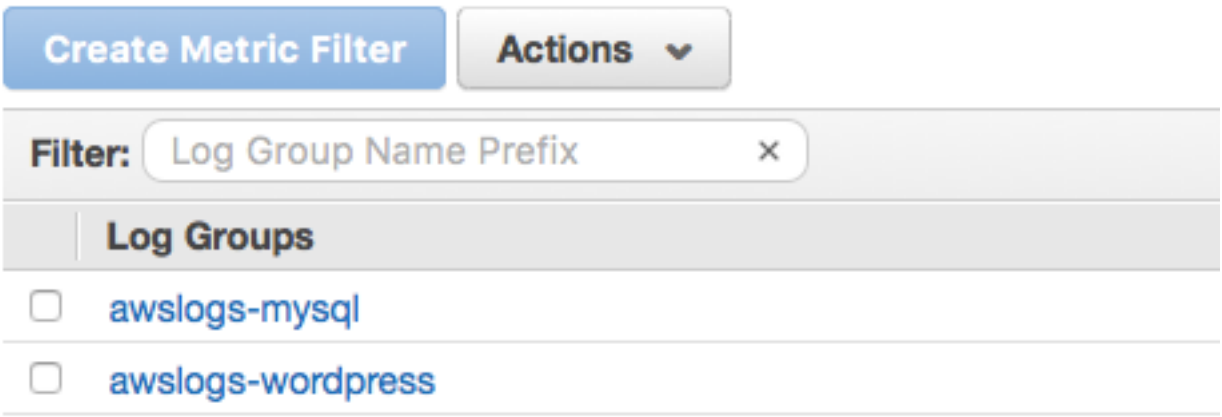
To view your CloudWatch Logs data for a container from the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, select the cluster that contains the task to view.
3. On the **Cluster: *cluster_name*** page, choose **Tasks** and select the task to view.
4. On the **Task: *task_id*** page, expand the container view by choosing the arrow to the left of the container name.
5. In the **Log Configuration** section, choose **View logs in CloudWatch**, which opens the associated log stream in the CloudWatch console.

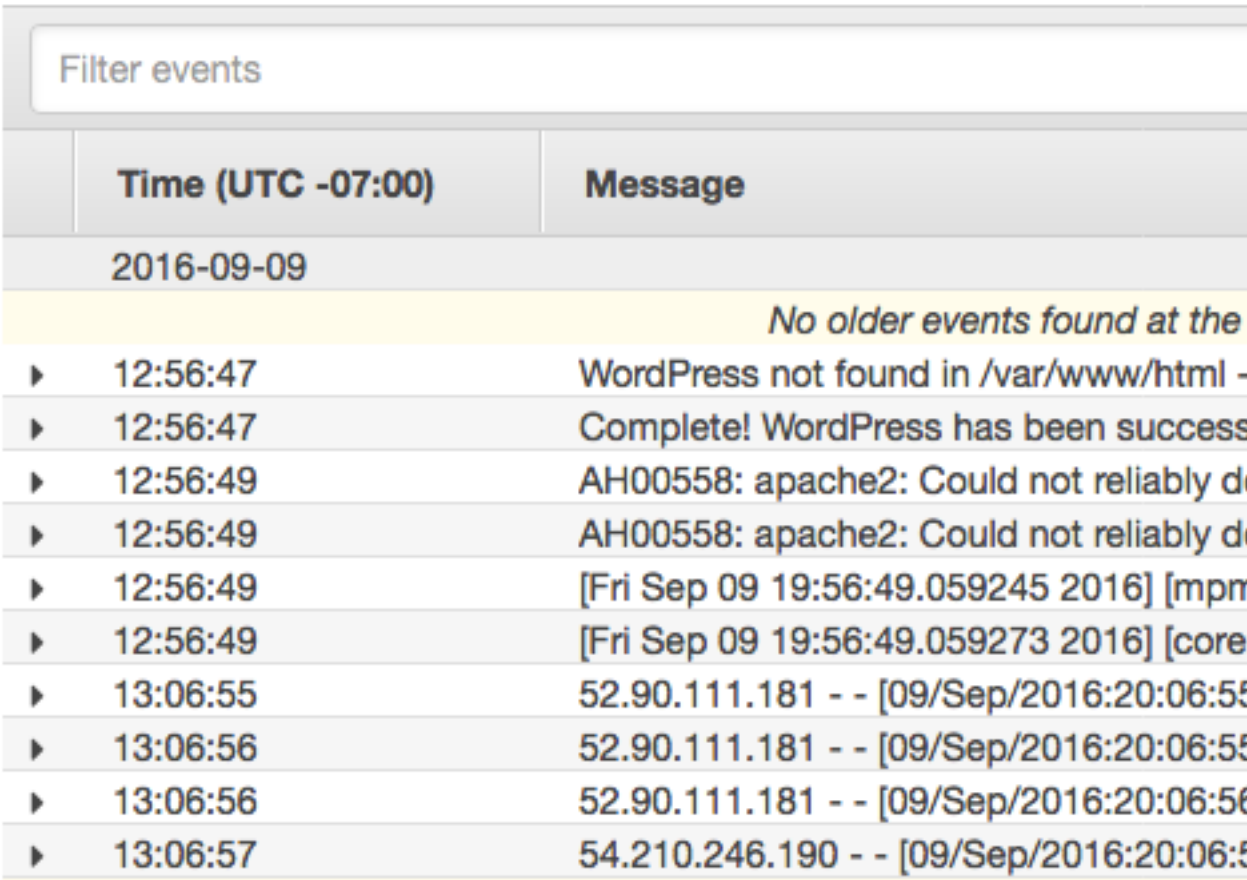


To view your CloudWatch Logs data in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**.
3. Select a log group to view. You should see the log groups that you created in Creating Your Log Groups (p. 55).



4. Choose a log stream to view.



Private Registry Authentication for Tasks

Private registry authentication for tasks using AWS Secrets Manager enables you to store your credentials securely and then reference them in your container definition. This allows your tasks to use images from private repositories. This feature is supported by tasks using both the Fargate or EC2 launch types.

For tasks using the Fargate launch type, this feature requires platform version 1.2.0 or later. For information, see AWS Fargate Platform Versions (p. 19).

Within your container definition, specify `repositoryCredentials` with the full ARN of the secret that you created. The secret you reference can be from a different Region than the task using it, but must be from within the same account.

> **Note**
> When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same Region as the task you are launching then you can use either the full ARN or name of the secret. When using the AWS Management Console, the full ARN of the secret must be specified.

The following is a snippet of a task definition showing the required parameters:

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
```

```
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

Private Registry Authentication Required IAM Permissions

The Amazon ECS task execution role is required to use this feature. This allows the container agent to pull the container image. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

To provide access to the secrets that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`—Required only if your key uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

An example inline policy adding the permissions is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name",
        "arn:aws:kms:region:aws_account_id:key:key_id"
      ]
    }
  ]
}
```

Enabling Private Registry Authentication

To create a basic secret

Use AWS Secrets Manager to create a secret for your private registry credentials.

1. Open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose **Store a new secret**.
3. For **Select secret type**, choose **Other type of secrets**.
4. Select **Plaintext** and enter your private registry credentials using the following format:

   ```
   {
     "username" : "privateRegistryUsername",
     "password" : "privateRegistryPassword"
   }
   ```

5. Choose **Next**.

6. For **Secret name**, type an optional path and name, such as **`production/MyAwesomeAppSecret`** or **`development/TestSecret`**, and choose **Next**. You can optionally add a description to help you remember the purpose of this secret later.

   The secret name must be ASCII letters, digits, or any of the following characters: /_+=.@-

7. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at **Disable automatic rotation** and choose **Next**.

   For information about how to configure rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets.

8. Review your settings, and then choose **Store secret** to save everything you entered as a new secret in Secrets Manager.

To create a task definition that uses private registry authentication

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Task Definitions**.
3. On the **Task Definitions** page, choose **Create new Task Definition**.
4. On the **Select launch type compatibility** page, choose the launch type for your tasks and then **Next step**.
5. For **Task Definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. For **Task execution role**, either select your existing task execution role or choose **Create new role** to have one created for you. This role authorizes Amazon ECS to pull private images for your task. For more information, see Private Registry Authentication Required IAM Permissions (p. 62).

   Important
   If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following shows the syntax:

   ```
   "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
   ```

7. For each container to create in your task definition, complete the following steps:

   a. In the **Container Definitions** section, choose **Add container**.
   b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   d. Select the **Private repository authentication** option.
   e. For **Secrets manager ARN**, enter the full Amazon Resource Name (ARN) of the secret that you created earlier. The value must be between 20 and 2048 characters.
   f. Fill out the remaining required fields and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see Task Definition Parameters (p. 31).
   g. Choose **Add**.

8. When your containers are added, choose **Create**.

Example Task Definitions

Below are some task definition examples that you can use to start creating your own task definitions. For more information, see Task Definition Parameters (p. 31) and Creating a Task Definition (p. 26).

Example Example: Webserver

The following is an example task definition using the Fargate launch type that sets up a web server:

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' >  /usr/local/apache2/htdocs/index.html && httpd-foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

Important
If you use this task definition with a load balancer, you need to complete the WordPress setup installation through the web interface on the container instance immediately after the container starts. The load balancer health check ping expects a `200` response from the server, but WordPress returns a `301` until the installation is completed. If the load balancer health check fails, the load balancer deregisters the instance.

Example Example: `awslogs` Log Driver

The following example demonstrates how to use the `awslogs` log driver in a task definition that uses the Fargate launch type. The `nginx` container sends its logs to the `ecs-log-streaming` log group in the `us-west-2` region. For more information, see Using the awslogs Log Driver (p. 55).

```
{
  "containerDefinitions": [
    {
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "name": "nginx-container",
      "image": "nginx",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "ecs-log-streaming",
          "awslogs-region": "us-west-2",
          "awslogs-stream-prefix": "fargate-task-1"
        }
      },
      "cpu": 0
    }
  ],
  "networkMode": "awsvpc",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "memory": "2048",
  "cpu": "1024",
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "family": "example_task_1"
}
```

Updating a Task Definition

To update a task definition, create a task definition revision. If the task definition is used in a service, you must update that service to use the updated task definition.

To create a task definition revision

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, select the box to the left of the task definition to revise and choose **Create new revision**.
5. On the **Create new revision of Task Definition** page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, make the changes, and then choose **Update**.
6. Verify the information and choose **Create**.

7. If your task definition is used in a service, update your service with the updated task definition. For more information, see Updating a Service (p. 115).

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).

> **Note**
> At this time, `INACTIVE` task definitions remain discoverable in your account indefinitely; however, this behavior is subject to change in the future, so you should not rely on `INACTIVE` task definitions persisting beyond the lifecycle of any associated tasks and services.

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**, **Deregister**.
7. Verify the information in the **Deregister Task Definition** window, and choose **Deregister** to finish.

Scheduling Amazon ECS Tasks

Amazon Elastic 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 the same cluster state information provided by the Amazon ECS API to make appropriate placement decisions.

Each task that uses the Fargate launch type has it's own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

Amazon ECS provides a service scheduler (for long-running tasks and applications), the ability to run tasks manually (for batch jobs or single run tasks), with Amazon ECS placing tasks on your cluster for you. You can specify task placement strategies and constraints that allow you to run tasks in the configuration you choose, such as spread out across Availability Zones. It is also possible to integrate with custom or third-party schedulers.

Service Scheduler

The service scheduler is ideally suited for long running stateless services and applications. The service scheduler ensures that the scheduling strategy you specify is followed and reschedules tasks when a task fails (for example, if the underlying infrastructure fails for some reason).

There are two service scheduler strategies available:

- `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica (p. 75).
- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 75).

  > **Note**
  > Fargate tasks do not support the `DAEMON` scheduling strategy.

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. By default, the service scheduler spreads tasks across Availability Zones, but you can use task placement strategies and constraints to customize task placement decisions. For more information, see Services (p. 74).

Manually Running Tasks

The `RunTask` action is ideally suited for processes such as batch jobs that perform work and then stop. For example, you could have a process call `RunTask` when work comes into a queue. The task pulls work from the queue, performs the work, and then exits. Using `RunTask`, you can allow the default task placement strategy to distribute tasks randomly across your cluster, which minimizes the chances that a single instance gets a disproportionate number of tasks. Alternatively, you can use `RunTask` to customize how the scheduler places tasks using task placement strategies and constraints. For more information, see Running Tasks (p. 68) and RunTask in the *Amazon Elastic Container Service API Reference*.

Running Tasks on a `cron`-like Schedule

If you have tasks to run at set intervals in your cluster, such as a backup operation or a log scan, you can use the Amazon ECS console to create a CloudWatch Events rule that runs one or more tasks in your cluster at specified times. Your scheduled event rule can be set to either a specific interval (run every *N* minutes, hours, or days), or for more complicated scheduling, you can use a `cron` expression. For more information, see Scheduled Tasks (`cron`) (p. 70).

Contents

- Running Tasks (p. 68)
- Scheduled Tasks (cron) (p. 70)
- Task Retirement (p. 72)

Running Tasks

Running tasks manually is ideal in certain situations. For example, suppose that you are developing a task but you are not ready to deploy this task with the service scheduler. Perhaps your task is a one-time or periodic batch job that does not make sense to keep running or restart when it finishes.

To keep a specified number of tasks running or to place your tasks behind a load balancer, use the Amazon ECS service scheduler instead. For more information, see Services (p. 74).

To run a task using the Fargate launch type

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Task Definitions** and select the task definition to run.
   - To run the latest revision of a task definition shown here, select the box to the left of the task definition to run.
   - To run an earlier revision of a task definition shown here, select the task definition to view all active revisions, then select the revision to run.
3. Choose **Actions**, **Run Task**.
4. In the **Run Task** section, complete the following steps:
   a. For **Launch type**, choose **FARGATE**. For more information about launch types, see Amazon ECS Launch Types (p. 50).
   b. For **Platform version**, choose **LATEST**. For more information about platform versions, see AWS Fargate Platform Versions (p. 19).
   c. For **Cluster**, choose the cluster to use.
   d. For **Number of tasks**, type the number of tasks to launch with this task definition.
   e. For **Task Group**, type the name of the task group.
5. In the **VPC and security groups** section, complete the following steps:
   a. For **Cluster VPC**, choose the VPC for your tasks to use. Ensure that the VPC that you choose is not configured to require dedicated hardware tenancy, as that is not supported by Fargate tasks.
   b. For **Subnets**, choose the available subnets for your task.
   c. For **Security groups**, a security group has been created for your task that allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.
   d. For **Auto-assign public IP**, choose **ENABLED** if you want the elastic network interface attached to the Fargate task to be assigned a public IP address. This is required if your task needs

outbound network access, for example to pull an image. If outbound network access is not required, then you can choose **DISABLED**.

6. In the **Advanced Options** section, complete the following steps:

   - (Optional) To send command or environment variable overrides to one or more containers in your task definition, or to specify an IAM role task override, choose **Advanced Options** and complete the following steps:

     i. For **Task Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 168).

        Only roles with the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 169).

     ii. For **Task Execution Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 168).

        Only roles with the **Amazon EC2 Container Service Task Execution Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 169).

     iii. For **Container Overrides**, choose a container to which to send a command or environment variable override.

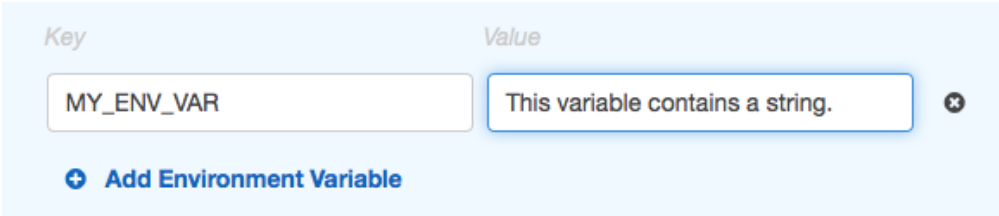
        - **For a command override:** For **Command override**, type the command override to send. If your container definition does not specify an `ENTRYPOINT`, the format should be a comma-separated list of non-quoted strings. For example:

          ```
          /bin/sh,-c,echo,$DATE
          ```

          If your container definition does specify an `ENTRYPOINT` (such as **sh,-c**), the format should be an unquoted string, which is surrounded with double quotes and passed as an argument to the `ENTRYPOINT` command. For example:

          ```
          while true; do echo $DATE > /var/www/html/index.html; sleep 1; done
          ```

        - **For environment variable overrides:** Choose **Add Environment Variable**. For **Key**, type the name of your environment variable. For **Value**, type a string value for your environment value (without surrounding quotes).

          This environment variable override is sent to the container as:

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

7. In the **Task tagging configuration** section, complete the following steps:

   a. Select **Enable ECS managed tags** if you want Amazon ECS to automatically tag each task with the Amazon ECS managed tags. For more information, see Tagging Your Amazon ECS Resources.

b. For **Propagate tags from**, select one of the following:

- **Do not propagate** – This option will not propagate any tags.
- **Task Definitions** – This option will propagate the tags specified in the task definition to the task.

Note
If you specify a tag with the same `key` in the **Tags** section, it will override the tag propagated from the task definition.

8. In the **Tags** section, specify the key and value for each tag to associate with the task. For more information, see Tagging Your Amazon ECS Resources.
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. 216) in the troubleshooting section.

Scheduled Tasks (cron)

You can run Amazon ECS tasks on a `cron`-like schedule using CloudWatch Events rules and targets.

If you have tasks to run at set intervals in your cluster, such as a backup operation or a log scan, you can use the Amazon ECS console to create a CloudWatch Events rule that runs one or more tasks in your cluster at the specified times. Your scheduled event rule can be set to either a specific interval (run every *N* minutes, hours, or days), or for more complicated scheduling, you can use a `cron` expression. For more information, see Schedule Expressions for Rules in the *Amazon CloudWatch Events User Guide*.

Creating a scheduled task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster in which to create your scheduled task.
3. On the **Cluster: *cluster-name*** page, choose **Scheduled Tasks**, **Create**.
4. For **Schedule rule name**, enter a unique name for your schedule rule. Up to 64 letters, numbers, periods, hyphens, and underscores are allowed.
5. (Optional) For **Schedule rule description**, enter a description for your rule. Up to 512 characters are allowed.
6. For **Schedule rule type**, choose whether to use a fixed interval schedule or a `cron` expression for your schedule rule. For more information, see Schedule Expressions for Rules in the *Amazon CloudWatch Events User Guide*.

   - For **Run at fixed interval**, enter the interval and unit for your schedule.
   - For **Cron expression**, enter the `cron` expression for your task schedule. These expressions have six required fields, and fields are separated by white space. For more information, and examples of `cron` expressions, see Cron Expressions in the *Amazon CloudWatch Events User Guide*.
7. Create a target for your schedule rule.

   a. For **Target id**, enter a unique identifier for your target. Up to 64 letters, numbers, periods, hyphens, and underscores are allowed.

   b. For **Launch type**, choose whether your service should run tasks on Fargate infrastructure, or Amazon EC2 container instances that you maintain. For more information, see Amazon ECS Launch Types (p. 50).

Important
This feature is not yet available for Fargate tasks in the eu-west-2 (London), ap-northeast-2 (Seoul), and us-west-1 (N. California) Regions.

c. For **Task definition**, choose the family and revision (family:revision) of the task definition to run for this target.

d. For **Platform version**, choose the platform version to use for this target. For more information, see AWS Fargate Platform Versions (p. 19).

Note
Platform versions are only applicable to tasks that use the Fargate launch type.

e. For **Number of tasks**, enter the number of instantiations of the specified task definition to run on your cluster when the rule executes.

f. (Optional) For **Task role override**, choose the IAM role to use for the task in your target, instead of the task definition default. For more information, see IAM Roles for Tasks (p. 168). Only roles with the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 169). You must add `iam:PassRole` permissions for any task role overrides to the CloudWatch IAM role. For more information, see CloudWatch Events IAM Role (p. 166).

g. If your scheduled task's task definition uses the `awsvpc` network mode, you must configure a VPC, subnet, and security group settings for your scheduled task. For more information, see Task Networking with the `awsvpc` Network Mode (p. 54).

   i. For **Cluster VPC**, if you selected the EC2 launch type, choose the VPC in which your container instances reside. If you selected the Fargate launch type, select the VPC that the Fargate tasks should use. Ensure that the VPC you choose is not configured to require dedicated hardware tenancy as that is not supported by Fargate tasks.

   ii. For **Subnets**, choose the available subnets for your scheduled task placement.

   Important
   Only private subnets are supported for the `awsvpc` network mode. Because tasks do not receive public IP addresses, a NAT gateway is required for outbound internet access, and inbound internet traffic should be routed through a load balancer.

   iii. For **Security groups**, a security group has been created for your scheduled tasks, which allows HTTP traffic from the internet (`0.0.0.0/0`). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.

   iv. For **Auto-assign Public IP**, choose whether to have your tasks receive a public IP address. If you are using Fargate tasks, a public IP address must be assigned to the task's elastic network interface, with a route to the internet, or a NAT gateway that can route requests to the internet. This allows the task to pull container images.

h. For **CloudWatch Events IAM role for this target**, choose an existing CloudWatch Events service role (`ecsEventsRole`) that you may have already created. Or, choose **Create new role** to create the required IAM role that allows CloudWatch Events to make calls to Amazon ECS to run tasks on your behalf. For more information, see CloudWatch Events IAM Role (p. 166).

Important
If your scheduled tasks require the use of the task execution role, or if they use a task role override, then you must add `iam:PassRole` permissions for your task execution role or task role override to the CloudWatch IAM role. For more information, see CloudWatch Events IAM Role (p. 166).

i. (Optional) In the **Container overrides** section, you can expand individual containers and override the command and/or environment variables for that container that are defined in the task definition.

8. (Optional) To add additional targets (other tasks to run when this rule is executed), choose **Add targets** and repeat the previous substeps for each additional target.
9. Choose **Create**.

To edit a scheduled task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster in which to edit your scheduled task.
3. On the **Cluster: *cluster-name*** page, choose **Scheduled Tasks**.
4. Select the box to the left of the schedule rule to edit, and choose **Edit**.
5. Edit the fields to update and choose **Update**.

Task Retirement

A task can be scheduled for retirement in the following scenarios:

- Your Fargate tasks are running on a platform version that has a security vulnerability that requires you to launch new tasks using a patched platform version.
- AWS detects the irreparable failure of the underlying hardware hosting the task.

When a task reaches its scheduled retirement date, it is stopped or terminated by AWS. If the task is part of a service, then the task is automatically stopped and the service scheduler starts a new one to replace it. If you are using standalone tasks, then you receive notification of the task retirement and must launch new tasks to replace them.

Identifying Tasks Scheduled for Retirement

If your task is scheduled for retirement, you receive an email before the event with the task ID and retirement date. This email is sent to the address that's associated with your account, the same email address that you use to log in to the AWS Management Console. If you use an email account that you do not check regularly, then you can use the AWS Personal Health Dashboard to determine if any of your tasks are scheduled for retirement. To update the contact information for your account, go to the Account Settings page.

Working with Tasks Scheduled for Retirement

If the task is part of a service, then the task is automatically stopped, The service scheduler starts a new one to replace it after it reaches its scheduled retirement date. If you would like to update your service tasks before the retirement date, you can use the following steps. For more information, see Updating a Service (p. 115).

To update a running service (AWS Management Console)

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the name of the cluster in which your service resides.
5. On the **Cluster: *name*** page, choose **Services**.
6. Check the box to the left of the service to update and choose **Update**.

7. On the **Configure service** page, your service information is pre-populated. Select **Force new deployment** and choose **Next step**.

   Note
   For tasks using the Fargate launch type, forcing a new deployment launches new tasks using the patched platform version. Your tasks do not require you select a different platform version. For more information, see AWS Fargate Platform Versions (p. 19).
8. On the **Configure network** and **Set Auto Scaling (optional)** pages, choose **Next step**.
9. Choose **Update Service** to finish and update your service.

To update a running service (AWS CLI)

1. Obtain the ARN for the service.

```
aws ecs list-services --cluster cluster_name --region region
```

   Output:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/MyService"
    ]
}
```

2. Update your service, forcing a new deployment that deploys new tasks.

```
aws ecs update-service --service serviceArn --force-new-deployment --
cluster cluster_name --region region
```

If you are using standalone tasks, then you can start a new task to replace it. For more information, see Running Tasks (p. 68).

Services

Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. This is called a service. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service depending on the scheduling strategy used.

In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

Topics

- Service Scheduler Concepts (p. 74)
- Additional Service Concepts (p. 75)
- Service Definition Parameters (p. 75)
- Service Load Balancing (p. 81)
- Service Auto Scaling (p. 90)
- Service Discovery (p. 97)
- Creating a Service (p. 108)
- Updating a Service (p. 115)
- Service Throttle Logic (p. 117)

Service Scheduler Concepts

If a task in a service stops, the task is killed and a new task is launched. This process continues until your service reaches the number of desired running tasks based on the scheduling strategy that you specified.

The service scheduler includes logic that throttles how often tasks are restarted if they repeatedly fail to start. If a task is stopped without having entered a `RUNNING` state, determined by the task having a `startedAt` time stamp, the service scheduler starts to incrementally slow down the launch attempts and emits a service event message. This behavior prevents unnecessary resources from being used for failed tasks, giving you a chance to resolve the issue. After the service is updated, the service scheduler resumes normal behavior. For more information, see Service Throttle Logic (p. 117) and Service Event Messages (p. 218).

There are two service scheduler strategies available:

- `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica (p. 75).
- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 75).

  Note
  Fargate tasks do not support the `DAEMON` scheduling strategy.

Daemon

The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints specified in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies.

The daemon service scheduler does not place any tasks on instances that have the `DRAINING` status. If a container instance transitions to `DRAINING`, the daemon tasks on it are stopped. It also monitors when new container instances are added to your cluster and adds the daemon tasks to them.

If `deploymentConfiguration` is specified, the maximum percent parameter must be `100`. The default value for a daemon service for `maximumPercent` is 100%. The default value for a daemon service for `minimumHealthyPercent` is 0% for the AWS CLI, the AWS SDKs, and the APIs, and 50% for the AWS Management Console.

> **Note**
> The daemon service scheduler does not support the use of Classic Load Balancers.

Replica

The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions.

When the service scheduler, using the `REPLICA` strategy, launches new tasks or stops running tasks that use the Fargate launch type, it attempts to maintain balance across the Availability Zones in your service.

Additional Service Concepts

- You can optionally run your service behind a load balancer. For more information, see Service Load Balancing (p. 81).
- You can optionally specify a deployment configuration for your service. During a deployment (which is triggered by updating the task definition or desired count of a service), the service scheduler uses the minimum healthy percent and maximum percent parameters to determine the deployment strategy. For more information, see Service Definition Parameters (p. 75).
- You can optionally configure your service to use Amazon ECS service discovery. Service discovery uses Amazon Route 53 auto naming APIs to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see Service Discovery (p. 97).

Service Definition Parameters

A service definition defines which task definition to use with your service, how many instantiations of that task to run, and which load balancers (if any) to associate with your tasks.

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
```

```
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "FARGATE",
    "platformVersion": "",
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "random",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "ENABLED",
            "networkInterfaceOwner": "",
            "networkInterfaceCredential": ""
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "ECS"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION"
}
```

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. If you do not specify a cluster, the `default` cluster is assumed.

`serviceName`

The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a region or across multiple regions.

`taskDefinition`

The `family` and `revision` (`family:revision`) or full ARN of the task definition to run in your service. If a `revision` is not specified, the latest `ACTIVE` revision is used.

`loadBalancers`

A load balancer object representing the load balancer to use with your service. Currently, you are limited to one load balancer or target group per service. After you create a service, the load balancer name or target group ARN, container name, and container port specified in the service definition are immutable.

For Classic Load Balancers, this object must contain the load balancer name, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance is registered with the load balancer specified here.

For Application Load Balancers and Network Load Balancers, this object must contain the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified here.

`targetGroupArn`

The full Amazon Resource Name (ARN) of the Elastic Load Balancing target group associated with a service.

`loadBalancerName`

The name of the load balancer.

`containerName`

The name of the container (as it appears in a container definition) to associate with the load balancer.

`containerPort`

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the service's task definition. Your container instances must allow ingress traffic on the `hostPort` of the port mapping.

`serviceRegistries`

The details of the service discovery configuration for your service. For more information, see Service Discovery (p. 97).

`registryArn`

The Amazon Resource Name (ARN) of the service registry. The currently supported service registry is Amazon Route 53 Auto Naming. For more information, see Service.

`port`

The port value used if your service discovery service specified an SRV record. This field is required if both the `awsvpc` network mode and SRV records are used.

`containerName`

The container name value, already specified in the task definition, to be used for your service discovery service. If the task definition that your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition that your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

`containerPort`

The port value, already specified in the task definition, to be used for your service discovery service. If the task definition your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

`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.

`launchType`

The launch type on which to run your service. If one is not specified, `EC2` is used by default. For more information, see Amazon ECS Launch Types (p. 50).

`platformVersion`

The platform version on which to run your service. If one is not specified, the latest version (`LATEST`) is used by default.

AWS Fargate platform versions are used to refer to a specific runtime environment for the Fargate task infrastructure. When specifying the `LATEST` platform version when running a task or creating a service, you get the most current platform version available for your tasks. When you scale up your service, those tasks receive the platform version that was specified on the service's current deployment. For more information, see AWS Fargate Platform Versions (p. 19).

`role`

The name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is required if you are using a load balancer with your service. If you specify the `role` parameter, you must also specify a load balancer object with the `loadBalancers` parameter.

If your specified role has a path other than `/`, then you must either specify the full role ARN (this is recommended) or prefix the role name with the path. For example, if a role with the name `bar` has a path of `/foo/` then you would specify `/foo/bar` as the role name. For more information, see Friendly Names and Paths in the *IAM User Guide*.

`deploymentConfiguration`

Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

`maximumPercent`

The `maximumPercent` parameter represents an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the `desiredCount` (rounded down to the nearest integer). This parameter enables you to define the deployment batch size. For example, if your replica service has a `desiredCount` of four tasks and a `maximumPercent` value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for a replica service for `maximumPercent` is 200%.

If you are using a daemon service type, the `maximumPercent` should remain at 100%, which is the default value.

The maximum number of tasks during a deployment is the `desiredCount` multiplied by the `maximumPercent`/100, rounded down to the nearest integer value.

`minimumHealthyPercent`

The `minimumHealthyPercent` represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the `desiredCount` (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a `desiredCount` of four tasks and a `minimumHealthyPercent` of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that *do not* use a load balancer are considered healthy if they are in the `RUNNING` state. Tasks for services that *do* use a load balancer are considered healthy if they are in the `RUNNING` state and the container instance on which the load balancer is hosted is reported as healthy. The default value for a replica service for `minimumHealthyPercent` is 50% in the AWS Management Console and 100% for the AWS CLI, the AWS SDKs, and the APIs. The default value for a daemon service for `minimumHealthyPercent` is 0% for the AWS CLI, the AWS SDKs, and the APIs and 50% for the AWS Management Console.

The minimum number of healthy tasks during a deployment is the `desiredCount` multiplied by the `minimumHealthyPercent`/100, rounded up to the nearest integer value.

`placementStrategy`

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

`type`

The type of placement strategy. The `random` placement strategy randomly places tasks on available candidates. The `spread` placement strategy spreads placement across available candidates evenly based on the `field` parameter. The `binpack` strategy places tasks on available candidates that have the least available amount of the resource that is specified with the `field` parameter. For example, if you binpack on memory, a task is placed on the instance with the least amount of remaining memory (but still enough to run the task).

`field`

The field to apply the placement strategy against. For the `spread` placement strategy, valid values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that is applied to a container instance, such as `attribute:ecs.availability-zone`. For the `binpack` placement strategy, valid values are `cpu` and `memory`. For the `random` placement strategy, this field is not used.

`networkConfiguration`

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own Elastic Network Interface, and it is not supported for other network modes. If using the Fargate launch type, the `awsvpc` network mode is required. For more information, see Task Networking with the `awsvpc` Network Mode (p. 54).

`awsvpcConfiguration`

An object representing the subnets and security groups for a task or service.

`subnets`

The subnets associated with the task or service.

`securityGroups`

The security groups associated with the task or service. If you do not specify a security group, the default security group for the VPC is used.

`assignPublicIP`

Whether the task's elastic network interface receives a public IP address.

`healthCheckGracePeriodSeconds`

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. This is only valid if your service is configured to use a load balancer. If your service's tasks take a while to start and respond to health checks, you can specify a health check grace period of up to 7,200 seconds during which the ECS service scheduler ignores the health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

`schedulingStrategy`

The scheduling strategy to use. For more information, see Service Scheduler Concepts (p. 74).

There are two service scheduler strategies available:

- `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica (p. 75).
- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 75).

  Note
  Fargate tasks do not support the `DAEMON` scheduling strategy.

`tags`

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. For more information, see Resources and Tags (p. 119).

`enableECSManagedTags`

Specifies whether to enable Amazon ECS managed tags for the tasks in the service. For more information, see Tagging Your Resources for Billing (p. 123).

`key`

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

`propagateTags`

Specifies whether to propagate the tags from the task definition or the service to the task. If no value is specified, the tags are not propagated.

Service Load Balancing

Your Amazon ECS service can optionally be configured to use Elastic Load Balancing to distribute traffic evenly across the tasks in your service.

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers, and Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic. For more information, see Load Balancer Types (p. 82).

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance).
- Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features, unless your service requires a feature that is only available with Network Load Balancers or Classic Load Balancers. For more information about Elastic Load Balancing and the differences between the load balancer types, see the Elastic Load Balancing User Guide.

> **Note**
> Currently, Amazon ECS services can only specify a single load balancer or target group. If your service requires access to multiple load balanced ports (for example, port 80 and port 443 for an HTTP/HTTPS service), you must use a Classic Load Balancer with multiple listeners. To use an Application Load Balancer, separate the single HTTP/HTTPS service into two services, where each handles requests for different ports. Then, each service could use a different target group behind a single Application Load Balancer.

Topics

- Load Balancing Concepts (p. 81)
- Load Balancer Types (p. 82)
- Creating a Load Balancer (p. 84)

Load Balancing Concepts

- All of the containers that are launched in a single task definition are always placed on the same container instance. For Classic Load Balancers, you may choose to put multiple containers (in the same task definition) behind the same load balancer by defining multiple host ports in the service definition and adding those listener ports to the load balancer. For example, if a task definition consists of Elasticsearch using port 3030 on the container instance, with Logstash and Kibana using port 4040 on the container instance, the same load balancer can route traffic to Elasticsearch and Kibana through two listeners. For more information, see Listeners for Your Classic Load Balancer in the *User Guide for Classic Load Balancers*.

Important
We do not recommend connecting multiple services to the same Classic Load Balancer. Because entire container instances are registered and deregistered with Classic Load Balancers (and not host and port combinations), this configuration can cause issues if a task from one service stops, causing the entire container instance to be deregistered from the Classic Load Balancer while another task from a different service on the same container instance is still using it. If you want to connect multiple services to a single load balancer (for example, to save costs), we recommend using an Application Load Balancer.

- There is a limit of one load balancer or target group per service.
- Services with tasks that use the `awsvpc` network mode (for example, those with the Fargate launch type) only support Application Load Balancers and Network Load Balancers. Classic Load Balancers are not supported. Also, when you create any target groups for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.
- If a service's task fails the load balancer health check criteria, the task is killed and restarted. This process continues until your service reaches the number of desired running tasks.
- If you configure your Application Load Balancer to use slow start mode, you must configure your task health check to return an `UNHEALTHY` status until after the slow start period is over. For more information about slow start mode, see Target Groups for Your Application Load Balancers.
- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting Service Load Balancers (p. 222).

Load Balancer Types

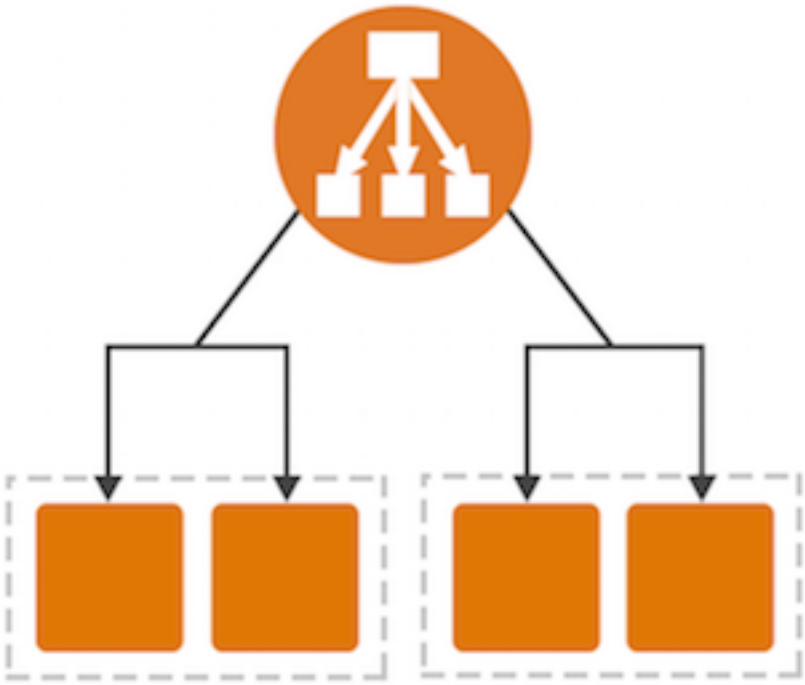
Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic.

Topics

- Application Load Balancer (p. 82)
- Network Load Balancer (p. 83)

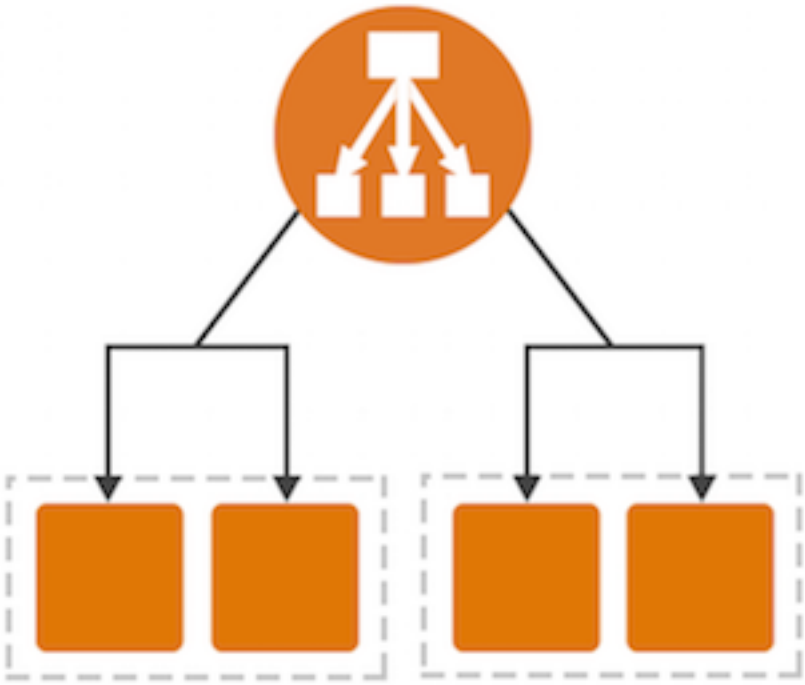
Application Load Balancer

An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster. Application Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Application Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the User Guide for Application Load Balancers.



Network Load Balancer

A Network Load Balancer makes routing decisions at the transport layer (TCP/SSL). It can handle millions of requests per second. After the load balancer receives a connection, it selects a target from the target group for the default rule using a flow hash routing algorithm. It attempts to open a TCP connection to the selected target on the port specified in the listener configuration. It forwards the request without modifying the headers. Network Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Network Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the User Guide for Network Load Balancers.



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 network traffic and routes it to your Amazon ECS container instances.

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers, and Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS traffic. Network Load Balancers and Classic Load Balancers are used to route TCP or Layer 4 traffic.

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance).
- Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features. For more information about Elastic Load Balancing and the differences between the load balancer types, see the Elastic Load Balancing User Guide.

> **Note**
> Currently, Amazon ECS services can only specify a single load balancer or target group. If your service requires access to multiple load balanced ports (for example, port 80 and port 443 for an HTTP/HTTPS service), you must use a Classic Load Balancer with multiple listeners. To use an Application Load Balancer, separate the single HTTP/HTTPS service into two services, where

each handles requests for different ports. Then, each service could use a different target group behind a single Application Load Balancer.

Topics

- Creating an Application Load Balancer (p. 85)
- Creating a Network Load Balancer (p. 89)

Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and a port for the frontend (client to load balancer) connections, and protocol and a port for the backend (load balancer to backend instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the containers in your tasks on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. On the **Select load balancer type** page, choose **Application Load Balancer** and then choose **Continue**.
6. Complete the **Configure Load Balancer** page as follows:

   a. For **Name**, type a name for your load balancer.

   b. For **Scheme**, an internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.

   c. For **IP address type**, choose **ipv4** to support IPv4 addresses only or **dualstack** to support both IPv4 and IPv6 addresses.

   d. For **Listeners**, the default is a listener that accepts HTTP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose **Add** to add another listener.

      Note
      If you plan on routing traffic to more than one target group, see ListenerRules for details on how to add host or path-based rules.

   e. For **VPC**, select the same VPC that you used for the container instances on which you intend to run your service.

   f. For **Availability Zones**, select the check box for the Availability Zones to enable for your load balancer. If there is one subnet for that Availability Zone, it is selected. If there is more than one subnet for that Availability Zone, select one of the subnets. You can select only one subnet per Availability Zone. Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.

   g. Choose **Next: Configure Security Settings**.

(Optional) Configure Security Settings

If you created a secure listener in the previous step, complete the **Configure Security Settings** page as follows; otherwise, choose **Next: Configure Security Groups**.

To configure security settings

1. If you have a certificate from AWS Certificate Manager, choose **Choose an existing certificate from AWS Certificate Manager (ACM)**, and then choose the certificate from **Certificate name**.
2. If you have already uploaded a certificate using IAM, choose **Choose an existing certificate from AWS Identity and Access Management (IAM)**, and then choose your certificate from **Certificate name**.
3. If you have a certificate ready to upload, choose **Upload a new SSL Certificate to AWS Identity and Access Management (IAM)**. For **Certificate name**, type a name for the certificate. For **Private Key**, copy and paste the contents of the private key file (PEM-encoded). In **Public Key Certificate**, copy and paste the contents of the public key certificate file (PEM-encoded). In **Certificate Chain**, copy and paste the contents of the certificate chain file (PEM-encoded), unless you are using a self-signed certificate and it's not important that browsers implicitly accept the certificate.
4. For **Select policy**, choose a predefined security policy. For details on the security policies, see Security Policies.
5. Choose **Next: Configure Security Groups**.

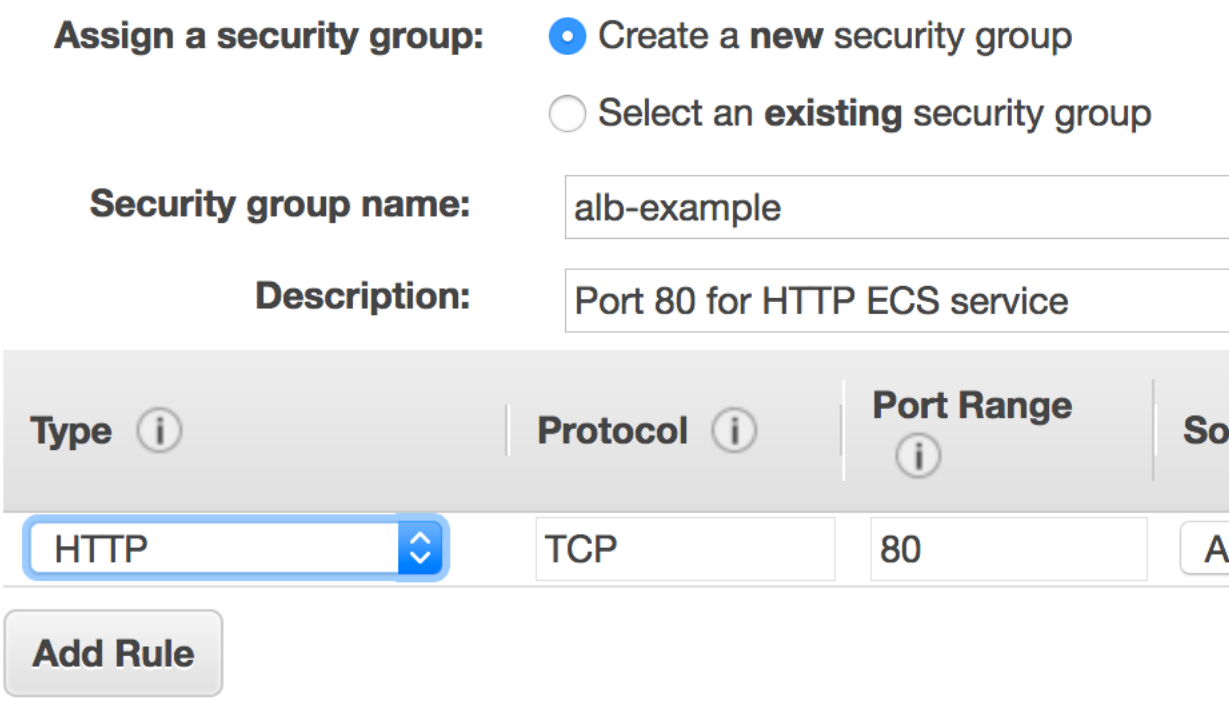
Configure Security Groups

You must assign a security group to your load balancer that allows inbound traffic to the ports that you specified for your listeners. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.
2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your listener to use.

   Note
   Later in this topic, you create a security group rule for your container instances that allows traffic on all ports coming from the security group created here, so that the Application Load Balancer can route traffic to dynamically assigned host ports on your container instances.



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

Configure Routing

In this section, you create a target group for your load balancer and the health check criteria for targets that are registered within that group.

To create a target group and configure health checks

1. For **Target group**, keep the default, **New target group**.
2. For **Name**, type a name for the new target group.
3. Set **Protocol** and **Port** as needed.
4. For **Target type**, choose whether to register your targets with an instance ID or an IP address.

   Important
   If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
5. For **Health checks**, keep the default health check settings.
6. Choose **Next: Register Targets**.

Register Targets

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.
2. Choose **Next: Review** to go to the next page in the wizard.

Review and Create

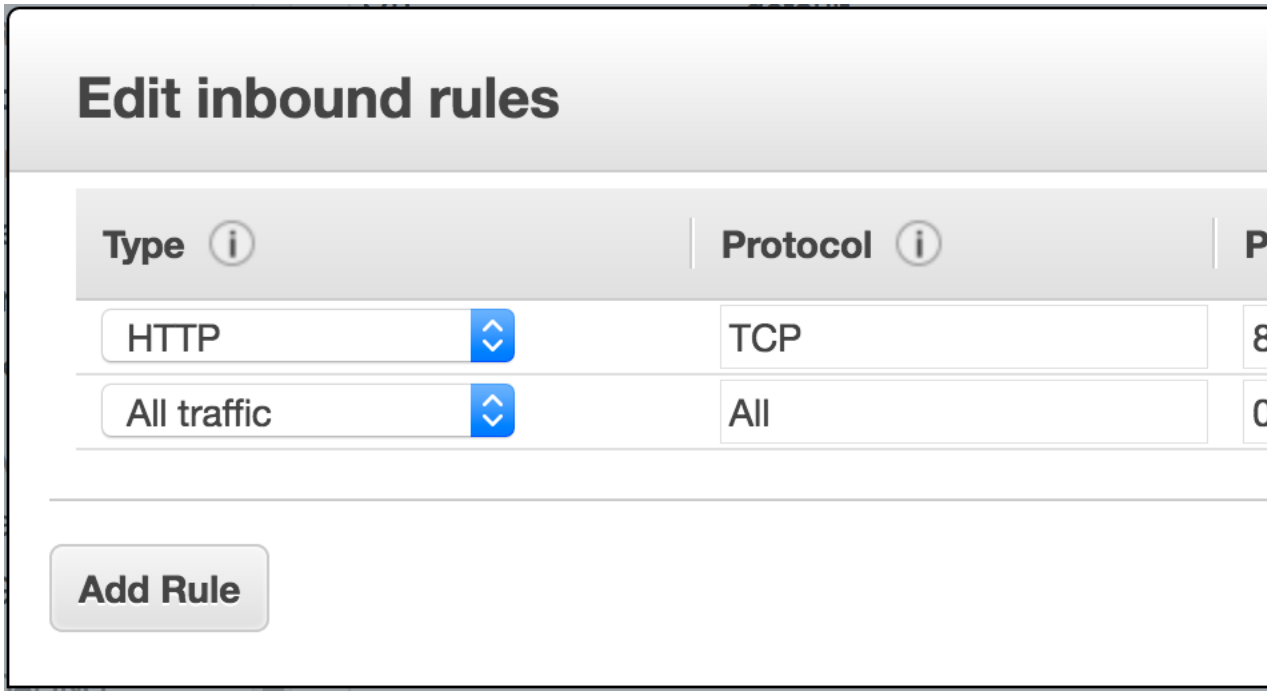
Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create a Security Group Rule for Your Container Instances

After your Application Load Balancer has been created, you must add an inbound rule to your container instance security group that allows traffic from your load balancer to reach the containers.

To allow inbound traffic from your load balancer to your container instances

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the left navigation, choose **Security Groups**.
3. Choose the security group that your container instances use. If you created your container instances by using the Amazon ECS first run wizard, this security group may have the description, **ECS Allowed Ports**.
4. Choose the **Inbound** tab, and then choose **Edit**.
5. For **Type**, choose **All traffic**.
6. For **Source**, choose **Custom**, and then type the name of your Application Load Balancer security group that you created in Configure Security Groups (p. 86). This rule allows all traffic from your Application Load Balancer to reach the containers in your tasks that are registered with your load balancer.



7. Choose **Save** to finish.

Create an Amazon ECS Service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see Creating a Service (p. 108).

Creating a Network Load Balancer

This section walks you through the process of creating a Network Load Balancer in the AWS Management Console.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections, and a protocol and port for the backend (load balancer to backend instance) connections. In this example, you configure an Internet-facing load balancer in the selected network with a listener that receives TCP traffic on port 80.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. On the **Select load balancer type** page, choose **Create** under **Network Load Balancer**.
6. Complete the **Configure Load Balancer** page as follows:
   a. For **Name**, type a name for your load balancer.
   b. For **Scheme**, choose either **internet-facing** or **internal**. An internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.
   c. For **Listeners**, the default is a listener that accepts TCP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose **Add listener** to add another listener.

      Note
      If you plan on routing traffic to more than one target group, see ListenerRules for details on how to add host or path-based rules.

   d. For **Availability Zones**, select the VPC that you used for your Amazon EC2 instances. For each Availability Zone that you used to launch your Amazon EC2 instances, select an Availability Zone and then select the public subnet for that Availability Zone. To associate an Elastic IP address with the subnet, select it from **Elastic IP**.
   e. Choose **Next: Configure Routing**.

Configure Routing

You register targets, such as Amazon EC2 instances, with a target group. The target group that you configure in this step is used as the target group in the listener rule, which forwards requests to the target group. For more information, see Target Groups for Your Network Load Balancers.

To configure your target group

1. For **Target group**, keep the default, **New target group**.
2. For **Name**, type a name for the target group.
3. Set **Protocol** and **Port** as needed.
4. For **Target type**, choose whether to register your targets with an instance ID or an IP address.

   Important
   If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
   You cannot register instances by instance ID if they have the following instance types: C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, and T1. You can register instances of these types by IP address.
5. For **Health checks**, keep the default health check settings.
6. Choose **Next: Register Targets**.

Register Targets with the Target Group

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.
2. Choose **Next: Review** to go to the next page in the wizard.

Review and Create

Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create an Amazon ECS Service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see Creating a Service (p. 108).

Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling leverages the Application Auto Scaling service to provide this functionality. Service Auto Scaling is available in all regions that support Amazon ECS. For more information, see the Application Auto Scaling User Guide.

Amazon ECS Service Auto Scaling supports the following types of scaling policies:

- Target Tracking Scaling Policies (p. 91)—Increase or decrease

the number of tasks that your service runs based on a target value for a specific metric. This is similar to the way that your thermostat maintains the temperature of your home. You select temperature and the thermostat does the rest.

- Step Scaling Policies (p. 95)—Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach.

Target Tracking Scaling Policies

With target tracking scaling policies, you select a metric and set a target value. Amazon ECS creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes service tasks as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target tracking scaling policy also adjusts to the fluctuations in the metric due to a fluctuating load pattern and minimizes rapid fluctuations in the number of tasks running in your service.

You can create multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The service scales based on the policy that provides the largest task capacity. This enables you to cover multiple scenarios and ensure that there is always enough capacity to process your application workloads.

To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually.

Do not edit or delete the CloudWatch alarms that Amazon ECS manages for a target tracking scaling policy. Amazon ECS deletes the alarms automatically when you delete the target tracking scaling policy.

Considerations

Keep the following considerations in mind when creating a target tracking scaling policy:

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value.
- A target tracking scaling policy does not perform scaling when the specified metric has insufficient data. It does not perform scale in because it does not interpret insufficient data as low utilization. To scale in when a metric has insufficient data, create a step scaling policy and have an alarm invoke the scaling policy when it changes to the `INSUFFICIENT_DATA` state. For information, see Alarm States in the *Amazon CloudWatch User Guide*.
- You may see gaps between the target value and the actual metric data points. This is because Application Auto Scaling always acts conservatively by rounding up or down when it determines how much capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity. However, for a scalable target with small capacity, the actual metric data points might seem far from the target value. For a scalable target with larger capacity, adding or removing capacity causes less of a gap between the target value and the actual metric data points.
- We recommend that you scale based on metrics with a 1-minute frequency because that ensures a faster response to utilization changes. Scaling on metrics with a 5-minute frequency can result in slower response time and scaling on stale metric data.
- To ensure application availability, Application Auto Scaling scales out proportionally to the metric as fast as it can, but scales in more gradually.
- Do not edit or delete the CloudWatch alarms that Application Auto Scaling manages for a target tracking scaling policy. Application Auto Scaling deletes the alarms automatically when you delete the scaling policy.

Tutorial: Service Auto Scaling with Target Tracking

The following procedures help you to create an Amazon ECS cluster and a service that uses Application Auto Scaling to scale out (and in) using target tracking.

In this tutorial, you create a cluster and a service (that runs behind an Elastic Load Balancing load balancer) using the Amazon ECS first run wizard. Then you configure Service Auto Scaling on the service with CloudWatch alarms that use the `CPUUtilization` metric to scale your service up or down, depending on the current application load.

When the CPU utilization of your service rises above 75% (meaning that more than 75% of the CPU that is reserved for the service is being used), the scale-out alarm triggers Service Auto Scaling to add another task to your service to help out with the increased load. Conversely, when the CPU utilization of your service drops below 75%, the scale in alarm triggers a decrease in the service's desired count to free up those cluster resources for other tasks and services.

Prerequisites

This tutorial assumes that you have an AWS account, an IAM administrator with permissions to perform all of the actions described within, and an Amazon EC2 key pair in the current region. If you do not have these resources, or your are not sure, you can create them by following the steps in Setting Up with Amazon ECS (p. 7).

Step 1: Create a Cluster and a Service

After you have enabled CloudWatch metrics for your clusters and services, you can create a cluster and service using the Amazon ECS first-run wizard. The first-run wizard takes care of creating the necessary IAM roles and policies for this tutorial, an Auto Scaling group for your container instances, and a service that runs behind a load balancer. The wizard also makes the clean-up process much easier, because you can delete the entire AWS CloudFormation stack in one step.

For this tutorial, you create a cluster called `service-autoscaling` and a service called `sample-webapp`.

To create your cluster and service

1. Open the Amazon ECS console first run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.
2. From the navigation bar, choose the **US East (N. Virginia)** region.
3. On **Step 1: Container and Task**, for **Container definition**, select **sample-app**.
4. For **Task definition**, leave all of the default options and choose **Next**.
5. On **Step 2: Service**, for **Load balancer type**, choose **Application Load Balancer**, **Next**.

   Important
   Application Load Balancers do incur costs while they exist in your AWS resources. For more information, see Elastic Load Balancing Pricing.
6. On **Step 3: Cluster**, for **Cluster name**, enter `service-autoscaling` and choose **Next**.
7. Review your choices and then choose **Create**.

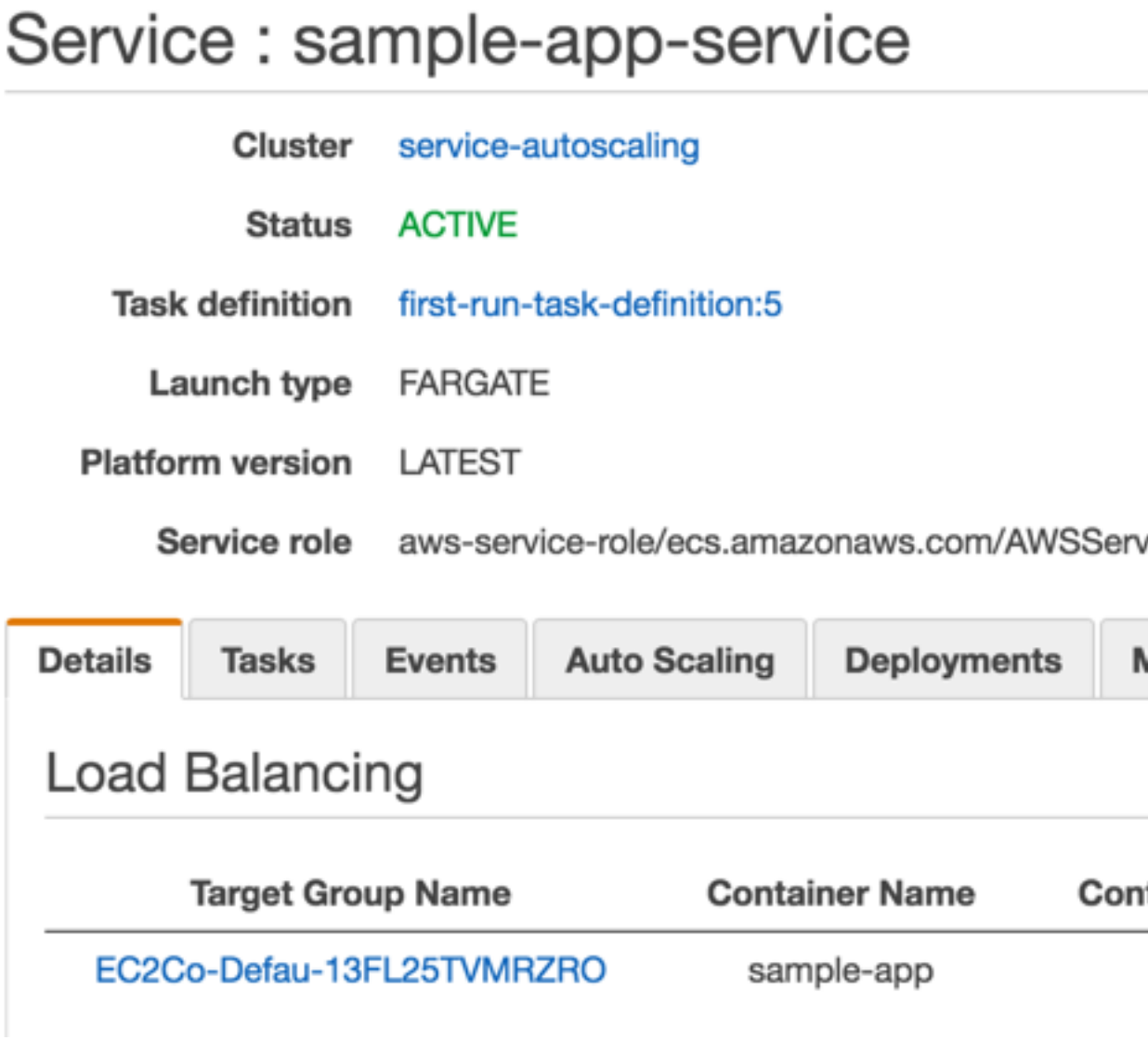
   You are directed to a **Launch Status** page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your cluster resources are created and populated).
8. When your cluster and service are created, choose **View service**.

Step 2: Configure Service Auto Scaling

Now that you have launched a cluster and created a service in that cluster that is running behind a load balancer, you can configure Service Auto Scaling by creating scaling policies to scale your service out and in response to CloudWatch alarms.

To configure basic Service Auto Scaling parameters

1. On the **Service: sample-app-service** page, your service configuration should look similar to the image below, although the task definition revision and load balancer name are likely to be different. Choose **Update** to update your new service.



2. On the **Update service** page, choose **Next step** until you get to **Step 3: Set Auto Scaling (optional)**.
3. For **Service Auto Scaling**, choose **Configure Service Auto Scaling to adjust your service's desired count**.
4. For **Minimum number of tasks**, enter `1` for the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted below this amount.

5. For **Desired number of tasks**, this field is pre-populated with the value that you entered earlier. This value must be between the minimum and maximum number of tasks specified on this page. Leave this value at `1`.
6. For **Maximum number of tasks**, enter `2` for the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted above this amount.
7. For **IAM role for Service Auto Scaling**, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose **Create new role** and the role is created for you. For future reference, the role that is created for you is called `ecsAutoscaleRole`. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 164).

To configure scaling policies for your service

These steps help you create scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale-out alarm to increase the desired count of your service, and a scale-in alarm to decrease the desired count of your service.

1. Choose **Add scaling policy** to configure your scaling policy.
2. On the **Add policy** page, update the following fields:
   a. For **Scaling policy type**, choose **Target tracking**.
   b. For **Policy name**, enter `TargetTrackingPolicy`.
   c. For **ECS service metric**, choose **CPUUtilization**.
   d. For **Target value**, enter `75`.
   e. For **Scale-out cooldown period**, enter `60`. This is the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
   f. For **Scale-in cooldown period**, enter `60`. This is the amount of time, in seconds, after a scale in activity completes before another scale in activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
   g. Choose **Save**.
3. Choose **Next step**.
4. Review all of your choices and then choose **Update Service**.
5. When your service status is finished updating, choose **View Service**.

Step 3: Trigger a Scaling Activity

After your service is configured with Service Auto Scaling, you can trigger a scaling activity by pushing your service's CPU utilization into the `ALARM` state. Because the example in this tutorial is a web application that is running behind a load balancer, you can send thousands of HTTP requests to your service (using the ApacheBench utility) to spike the service CPU utilization above the threshold amount. This spike should trigger the alarm, which in turn triggers a scaling activity to add one task to your service.

After the ApacheBench utility finishes the requests, the service CPU utilization should drop below your 25% threshold, triggering a scale in activity that returns the service's desired count to 1.

To trigger a scaling activity for your service

1. From your service's main view page in the console, choose the load balancer name to view its details in the Amazon EC2 console. You need the load balancer's DNS name, which should look something like `EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com`.
2. Use the ApacheBench (**ab**) utility to make thousands of HTTP requests to your load balancer in a short period of time.

Note
This command is installed by default on macOS, and it is available for many Linux distributions, as well. For example, you can install **ab** on Amazon Linux with the following command:

```
$ sudo yum install -y httpd24-tools
```

Run the following command, substituting your load balancer's DNS name.

```
$ ab -n 100000 -c 1000 http://EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-
east-1.elb.amazonaws.com/
```

3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
4. In the left navigation pane, choose **Alarms**.
5. Wait for your **ab** HTTP requests to trigger the scale-out alarm in the CloudWatch console. You should see your Amazon ECS service scale out and add one task to your service's desired count.
6. Shortly after your **ab** HTTP requests complete (between 1 and 2 minutes), your scale in alarm should trigger and the scale in policy reduces your service's desired count back to 1.

Step 4: Cleaning Up

When you have completed this tutorial, you may choose to keep your cluster, Auto Scaling group, load balancer, and CloudWatch alarms. However, if you are not actively using these resources, you should consider cleaning them up so that your account does not incur unnecessary charges.

To delete your cluster and CloudWatch alarms

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the left navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the **service-autoscaling** cluster.
4. Choose **Delete Cluster**, **Delete**. It may take a few minutes for the cluster AWS CloudFormation stack to finish cleaning up.
5. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
6. Choose **Alarms** and select the alarms that begin with `TargetTracking-service`.
7. Choose **Delete**, **Yes, Delete**.

Step Scaling Policies

Your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling is available in all regions that support Amazon ECS.

Service Auto Scaling Required IAM Permissions

Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling. IAM users must have the appropriate permissions for these services before they can use Service Auto Scaling in the AWS Management Console or with the AWS CLI or SDKs. In addition to the standard IAM permissions for creating and updating services, Service Auto Scaling requires the following permissions:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:*",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The Create Services (p. 176) and Update Services (p. 177) IAM policy examples show the permissions that are required for IAM users to use Service Auto Scaling in the AWS Management Console.

The Application Auto Scaling service needs permission to describe your ECS services and CloudWatch alarms, as well as permissions to modify your service's desired count on your behalf. You must create an IAM role (`ecsAutoscaleRole`) for your ECS services to provide these permissions and then associate that role with your service before it can use Application Auto Scaling. If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p.) or update service (p. 115) workflows, and then use the role for any other service later (in the console or with the CLI or SDKs). You can also create the role by following the procedures in Amazon ECS Service Auto Scaling IAM Role (p. 164).

Service Auto Scaling Concepts

- The ECS service scheduler respects the desired count at all times, but as long as you have active scaling policies and alarms on a service, Service Auto Scaling could change a desired count that was manually set by you.
- If a service's desired count is set below its minimum capacity value, and an alarm triggers a scale-out activity, Application Auto Scaling scales the desired count up to the minimum capacity value and then continues to scale out as required, based on the scaling policy associated with the alarm. However, a scale-in activity does not adjust the desired count, because it is already below the minimum capacity value.
- If a service's desired count is set above its maximum capacity value, and an alarm triggers a scale in activity, Application Auto Scaling scales the desired count down to the maximum capacity value and then continues to scale in as required, based on the scaling policy associated with the alarm. However, a scale- out activity does not adjust the desired count, because it is already above the maximum capacity value.
- During scaling activities, the actual running task count in a service is the value that Service Auto Scaling uses as its starting point, as opposed to the desired count, which is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that could not be satisfied, for example, if there are not enough container instance resources to place the additional tasks. If the container instance capacity is available later, the pending scaling activity may succeed, and then further scaling activities can continue after the cooldown period.

Amazon ECS Console Experience

The Amazon ECS console's service creation and service update workflows support Service Auto Scaling. The Amazon ECS console handles the `ecsAutoscaleRole` and policy creation, provided that the IAM user who is using the console has the permissions described in Service Auto Scaling Required IAM Permissions (p. 95), and that they can create IAM roles and attach policies to them.

When you configure a service to use Service Auto Scaling in the console, your service is automatically registered as a scalable target with Application Auto Scaling so that you can configure scaling policies that scale your service up and down. You can also create and update the scaling policies and CloudWatch alarms that trigger them in the Amazon ECS console.

To create a new ECS service that uses Service Auto Scaling, see Creating a Service (p. 108).

To update an existing service to use Service Auto Scaling, see Updating a Service (p. 115).

AWS CLI and SDK Experience

You can configure Service Auto Scaling by using the AWS CLI or the AWS SDKs, but you must observe the following considerations.

- Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling. For more information about these specific API operations, see the Amazon Elastic Container Service API Reference, the Amazon CloudWatch API Reference, and the Application Auto Scaling API Reference. For more information about the AWS CLI commands for these services, see the ecs, cloudwatch, and application-autoscaling sections of the AWS CLI Command Reference.
- Before your service can use Service Auto Scaling, you must register it as a scalable target with the Application Auto Scaling RegisterScalableTarget API operation.
- After your ECS service is registered as a scalable target, you can create scaling policies with the Application Auto Scaling PutScalingPolicy API operation to specify what should happen when your CloudWatch alarms are triggered.
- After you create the scaling policies for your service, you can create the CloudWatch alarms that trigger the scaling events for your service with the CloudWatch PutMetricAlarm API operation.

Service Discovery

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses Amazon Route 53 auto naming API actions to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see Using Auto Naming for Service Discovery in the *Amazon Route 53 API Reference*.

Service discovery is available in the following AWS Regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1

Region Name	Region
EU (Frankfurt)	eu-central-1
EU (Ireland)	eu-west-1
EU (London)	eu-west-2
EU (Paris)	eu-west-3
South America (São Paulo)	sa-east-1
Canada (Central)	ca-central-1

Topics

- Service Discovery Concepts (p. 98)
- Service Discovery Considerations (p. 98)
- Service Discovery Pricing (p. 99)
- Tutorial: Creating a Service Using Service Discovery (p. 99)

Service Discovery Concepts

Service discovery consists of the following components:

- **Service discovery namespace**: A logical group of services that share the same domain name, such as example.com. You need one namespace per Route 53 hosted zone and per VPC. If you are using service discovery from the Amazon ECS console, the workflow creates one private namespace per ECS cluster.
- **Service discovery service**: Exists within the service discovery namespace and consists of the service name and DNS configuration for the namespace. It provides the following core component:
  - **Service directory**: Allows you to look up a service via DNS or Route 53 auto naming API actions and get back one or more available endpoints that can be used to connect to the service.
- **Health checks**: Perform periodic container-level health checks. If an endpoint does not pass the health check, it is removed from DNS routing and marked as unhealthy. For more information, see How Amazon Route 53 Checks the Health of Your Resources.

Service Discovery Considerations

The following should be considered when using service discovery:

- Service discovery is supported for Fargate tasks if you are using platform version v1.1.0 or later. For more information, see AWS Fargate Platform Versions (p. 19).
- Public namespaces are supported but you must have an existing public hosted zone registered with Route 53 before creating your service discovery service.
- The DNS records created for a service discovery service always register with the private IP address for the task, rather than the public IP address, even when public namespaces are used.
- Service discovery requires that tasks specify either the `awsvpc`, `bridge`, or `host` network mode (`none` is not supported).
- If the task definition your service task specifies uses the `awsvpc` network mode, you can create any combination of A or SRV records for each service task. If you use SRV records, a port is required.
- If the task definition that your service task specifies uses the `bridge` or `host` network mode, an SRV record is the only supported DNS record type. Create an SRV record for each service task. The SRV record must specify a container name and container port combination from the task definition.

- DNS records for a service discovery service can be queried within your VPC. They use the following format: `<service discovery service name>.<service discovery namespace>`. For more information, see Step 5: Verify Service Discovery (p. 104).
- When doing a DNS query on the service name, A records return a set of IP addresses that correspond to your tasks. SRV records return a set of IP addresses and ports per task.
- You can configure service discovery for an ECS service that is behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.
- Service discovery does not support the use of Classic Load Balancers.
- When specifying health checks for your service discovery service, you must use either custom health checks managed by Amazon ECS or Route 53 health checks. The two options for health checks cannot be combined.
  - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and Elastic Load Balancing health checks, as well as your task state, to update the health with Route 53. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see HealthCheckCustomConfig in the *Amazon Route 53 API Reference*.
  - **HealthCheckConfig**—Route 53 creates health checks to monitor tasks. This requires the tasks to be publicly available. This is specified using the `--health-check-config` parameter when creating your service discovery service. For more information, see HealthCheckConfig in the *Amazon Route 53 API Reference*.
- If you are using the Amazon ECS console, the workflow creates one service discovery service per ECS service and maps all of the task IP addresses as A records, or task IP addresses and port as SRV records.
- Service discovery can only be configured when first creating a service. Updating existing services to configure service discovery for the first time or change the current configuration is not supported.
- The Route 53 resources created when service discovery is used must be cleaned up manually. For more information, see Step 6: Clean Up (p. 106) in the Tutorial: Creating a Service Using Service Discovery (p. 99) topic.

Service Discovery Pricing

Customers using Amazon ECS service discovery are charged for the usage of Route 53 auto naming APIs. This involves costs for creating the hosted zones and queries to the service registry. For more information, see Amazon Route 53 Pricing

Amazon ECS performs container level health checks and exposes this to Route 53 custom health check APIs and this is currently made available to customers at no extra cost. If you configure additional network health checks for publicly exposed tasks, you are charged for those health checks.

Tutorial: Creating a Service Using Service Discovery

Service discovery has been integrated into the Create Service wizard in the Amazon ECS console. For more information, see Creating a Service (p. 108).

The following tutorial shows how to create an ECS service containing a Fargate task that uses service discovery with the AWS CLI.

Note
Fargate tasks are only supported in the following regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2

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

Prerequisites (p. 100)

Step 1: Create the Service Discovery Namespace and Service (p. 100)

Step 2: Create a Cluster (p. 101)

Step 3: Register a Task Definition (p. 102)

Step 4: Create a Service (p. 103)

Step 5: Verify Service Discovery (p. 104)

Step 6: Clean Up (p. 106)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see Installing the AWS Command Line Interface.
- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 171) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: Create the Service Discovery Namespace and Service

Create a private service discovery namespace named `tutorial` within an existing VPC:

```
aws servicediscovery create-private-dns-namespace --name tutorial --vpc vpc-abcd1234 --
region us-east-1
```

Output:

```
{
    "OperationId": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e"
```

```
}
```

Using the `OperationId` from the previous output, verify that the private namespace was created successfully. Copy the namespace ID as it is used in subsequent commands.

```
aws servicediscovery get-operation --operation-id h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e
```

Output:

```
{
    "Operation": {
        "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",
        "Type": "CREATE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1519777852.502,
        "UpdateDate": 1519777856.086,
        "Targets": {
            "NAMESPACE": "ns-uejictsjen2i4eeg"
        }
    }
}
```

Using the `NAMESPACE` ID from the previous output, create a service discovery service named `myapplication`. Copy the service discovery service ID as it is used in subsequent commands.

```
aws servicediscovery create-service --name myapplication --dns-config 'NamespaceId="ns-
uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]' --health-check-custom-config
 FailureThreshold=1 --region us-east-1
```

Output:

```
{
    "Service": {
        "Id": "srv-utcrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

Step 2: Create a Cluster

Create an ECS cluster named `tutorial` to use.

```
aws ecs create-cluster --cluster-name tutorial --region us-east-1
```

Output:

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",
        "clusterName": "tutorial",
        "status": "ACTIVE",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": []
    }
}
```

Step 3: Register a Task Definition

Register a task definition that is compatible with Fargate. It requires the use of the `awsvpc` network mode. The following is the example task definition used for this tutorial.

First, create a file named `fargate-task.json` with the contents of the following task definition:

```
{
    "family": "tutorial-task-def",
        "networkMode": "awsvpc",
        "containerDefinitions": [
            {
                "name": "sample-app",
                "image": "httpd:2.4",
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "entryPoint": [
                    "sh",
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' >  /usr/local/apache2/htdocs/index.html && httpd-foreground\""
                ]
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

Then, register the task definition using the `fargate-task.json` file you created.

```
aws ecs register-task-definition --cli-input-json file://fargate-task.json --region us-
east-1
```

Step 4: Create a Service

Create a file named `ecs-service-discovery.json` with the contents of the ECS service that you are going to create. This example uses the task definition created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

```
{
    "cluster": "tutorial",
    "serviceName": "ecs-service-discovery",
    "taskDefinition": "tutorial-task-def",
    "serviceRegistries": [
       {
          "registryArn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk"
       }
    ],
    "launchType": "FARGATE",
    "platformVersion": "1.1.0",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
          "subnets": [ "subnet-abcd1234" ]
       }
    },
    "desiredCount": 1
}
```

Create your ECS service, specifying the Fargate launch type and the `1.1.0` platform version, which uses service discovery.

```
aws ecs create-service --cli-input-json file://ecs-service-discovery.json --region us-
east-1
```

Output:

```
{
    "service": {
        "serviceArn": "arn:aws:ecs:region:aws_account_id:service/ecs-service-discovery",
        "serviceName": "ecs-service-discovery",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [
            {
                "registryArn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk"
            }
        ],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "1.1.0",
        "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/tutorial-task-
def:1",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "deployments": [
```

```
            {
                "id": "ecs-svc/9223370516993140842",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
tutorial-task-def:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "createdAt": 1519861634.965,
                "updatedAt": 1519861634.965,
                "launchType": "FARGATE",
                "platformVersion": "1.1.0",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "subnets": [
                            "subnet-abcd1234"
                        ],
                        "securityGroups": [
                            "sg-abcd1234"
                        ],
                        "assignPublicIp": "ENABLED"
                    }
                }
            }
        ],
        "roleArn": "arn:aws:iam::aws_account_id:role/ECSServiceLinkedRole",
        "events": [],
        "createdAt": 1519861634.965,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-abcd1234"
                ],
                "securityGroups": [
                    "sg-abcd1234"
                ],
                "assignPublicIp": "ENABLED"
            }
        }
    }
}
```

Step 5: Verify Service Discovery

You can verify that everything has been created properly by querying your DNS information. After service discovery is configured, you can query it using either the Route 53 auto naming API actions or by using `dig` from within your VPC, as described below.

Using the service discovery service ID, list the service discovery instances:

```
aws servicediscovery list-instances --service-id srv-utcrh6wavdkggqtk --region us-east-1
```

Output:

```
{
    "Instances": [
        {
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",
            "Attributes": {
                "AWS_INSTANCE_IPV4": "172.31.87.2"
            }
```

```
        }
    ]
}
```

The DNS records created in the Route 53 hosted zone for the service discovery service can be queried with the following CLI commands.

First, using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID:

```
aws servicediscovery get-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

Output:

```
{
    "Namespace": {
        "Id": "ns-uejictsjen2i4eeg",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-
uejictsjen2i4eeg",
        "Name": "tutorial",
        "Type": "DNS_PRIVATE",
        "Properties": {
            "DnsProperties": {
                "HostedZoneId": "Z35JQ4ZFDRYPLV"
            }
        },
        "CreateDate": 1519777852.502,
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
    }
}
```

Then, using the Route 53 hosted zone ID, get the resource record set for the hosted zone:

```
aws route53 list-resource-record-sets --hosted-zone-id Z35JQ4ZFDRYPLV --region us-east-1
```

Output:

```
{
    "ResourceRecordSets": [
        {
            "Name": "tutorial.",
            "Type": "NS",
            "TTL": 172800,
            "ResourceRecords": [
                {
                    "Value": "ns-1536.awsdns-00.co.uk."
                },
                {
                    "Value": "ns-0.awsdns-00.com."
                },
                {
                    "Value": "ns-1024.awsdns-00.org."
                },
                {
                    "Value": "ns-512.awsdns-00.net."
                }
            ]
        },
        {
            "Name": "tutorial.",
            "Type": "SOA",
```

```
            "TTL": 900,
            "ResourceRecords": [
                {
                    "Value": "ns-1536.awsdns-00.co.uk. awsdns-hostmaster.amazon.com. 1 7200
 900 1209600 86400"
                }
            ]
        },
        {
            "Name": "myapplication.tutorial.",
            "Type": "A",
            "SetIdentifier": "16becc26-8558-4af1-9fbd-f81be062a266",
            "MultiValueAnswer": true,
            "TTL": 300,
            "ResourceRecords": [
                {
                    "Value": "172.31.87.2"
                }
            ]
        }
    ]
}
```

You can also query the DNS using `dig` from an instance within your VPC with the following command:

```
dig +short myapplication.tutorial
```

Output:

```
172.31.87.2
```

Step 6: Clean Up

When you are finished with this tutorial, you should clean up the resources associated with it to avoid incurring charges for resources that you are not using.

Cleaning up the service discovery and Amazon ECS resources

1. Deregister the service discovery service instances.

   ```
   aws servicediscovery deregister-instance --service-id srv-utcrh6wavdkggqtk --instance-
   id 16becc26-8558-4af1-9fbd-f81be062a266 --region us-east-1
   ```

   Output:

   ```
   {
       "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"
   }
   ```

2. Using the `OperationId` from the previous output, verify that the service discovery service instances were deregistered successfully.

   ```
   aws servicediscovery get-operation --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-
   jh0zimzv --region us-east-1
   ```

   Output:

   ```
   {
   ```

```
    "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "DEREGISTER_INSTANCE",
        "Status": "SUCCESS",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": {
            "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
            "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
            "SERVICE": "srv-utcrh6wavdkggqtk"
        }
    }
}
```

3. Delete the service discovery service.

```
aws servicediscovery delete-service --id srv-utcrh6wavdkggqtk --region us-east-1
```

4. Delete the service discovery namespace.

```
aws servicediscovery delete-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

Output:

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. Using the `OperationId` from the previous output, verify that the service discovery namespace was deleted successfully.

```
aws servicediscovery get-operation --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-
jh0ztysj --region us-east-1
```

Output:

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": {
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
        }
    }
}
```

6. Update the Amazon ECS service so the desired count is `0`, which allows you to delete it.

```
aws ecs update-service --cluster tutorial --service ecs-service-discovery --desired-
count 0 --force-new-deployment --region us-east-1
```

7. Delete the Amazon ECS service.

```
aws ecs delete-service --cluster tutorial --service ecs-service-discovery --region us-
east-1
```

8. Delete the Amazon ECS cluster.

```
aws ecs delete-cluster --cluster tutorial --region us-east-1
```

Creating a Service

When you create an Amazon ECS service, you specify the basic parameters that define what makes up your service and how it should behave. These parameters create a service definition.

You can optionally configure additional features, such as an Elastic Load Balancing load balancer to distribute traffic across the containers in your service. For more information, see Service Load Balancing (p. 81). You must verify that your container instances can receive traffic from your load balancers. You can allow traffic to all ports on your container instances from your load balancer's security group to ensure that traffic can reach any containers that use dynamically assigned ports.

Configuring Basic Service Parameters

All services require some basic configuration parameters that define the service, such as the task definition to use, which cluster the service should run on, how many tasks should be placed for the service, and so on; this is called the service definition. For more information about the parameters defined in a service definition, see Service Definition Parameters (p. 75).

This procedure covers creating a service with the basic service definition parameters that are required. After you have configured these parameters, you can create your service or move on to the procedures for optional service definition configuration, such as configuring your service to use a load balancer.

To configure the basic service definition parameters

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the region that your cluster is in.
3. In the navigation pane, choose **Task Definitions** and select the task definition from which to create your service.
4. On the **Task Definition name** page, select the revision of the task definition from which to create your service.
5. Review the task definition, and choose **Actions**, **Create Service**.
6. On the **Configure service** page, fill out the following parameters accordingly:

   - **Launch type**: Choose whether your service should run tasks on Fargate infrastructure, or Amazon EC2 container instances that you maintain. For more information, see Amazon ECS Launch Types (p. 50).
   - **Platform version**: If you selected the Fargate launch type, then select the platform version to use.
   - **Cluster**: Select the cluster in which to create your service.
   - **Service name**: Type a unique name for your service.
   - **Service type**: Select a scheduling strategy for your service. For more information, see Service Scheduler Concepts (p. 74).
   - **Number of tasks**: If you selected the replica service type, type the number of tasks to launch and maintain on your cluster.

     Note
     If your launch type is `EC2`, and your task definition uses static host port mappings on your container instances, then you need at least one container instance with the specified port

available in your cluster for each task in your service. This restriction does not apply if your task definition uses dynamic host port mappings with the `bridge` network mode. For more information, see portMappings (p. 34).

- **Minimum healthy percent**: Specify a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the service's desired number of tasks (rounded up to the nearest integer). For example, if your service has a desired number of four tasks and a minimum healthy percent of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that do not use a load balancer are considered healthy if they are in the `RUNNING` state. Tasks for services that do use a load balancer are considered healthy if they are in the `RUNNING` state and when the container instance on which it is hosted is reported as healthy by the load balancer. The default value for the minimum healthy percent is 50% in the console, and 100% with the AWS CLI or SDKs.
- **Maximum percent**: Specify an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the service's desired number of tasks (rounded down to the nearest integer). For example, if your service has a desired number of four tasks and a maximum percent value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for maximum percent is 200%.

7. In the **Task tagging configuration** section, complete the following steps:

   a. Select **Enable ECS managed tags** if you want Amazon ECS to automatically tag the tasks in the service with the Amazon ECS managed tags. For more information, see Tagging Your Amazon ECS Resources.

   b. For **Propagate tags from**, select one of the following:

      - **Do not propagate** – This option will not propagate any tags to the tasks in the service.
      - **Service** – This option will propagate the tags specified on your service to each of the tasks in the service.
      - **Task Definitions** – This option will propagate the tags specified in the task definition of a task to the tasks in the service.

      Note
      If you specify a tag with the same `key` in the **Tags** section, it will override the tag propagated from either the service or the task definition.

8. In the **Tags** section, specify the key and value for each tag to associate with the task. For more information, see Tagging Your Amazon ECS Resources.
9. Choose **Next step** to proceed.

Configure a Network

VPC and Security Groups

If your service's task definition uses the `awsvpc` network mode, you must configure a VPC, subnet, and security group settings for your service. For more information, see the section called “Task Networking” (p. 54).

To configure VPC and security group settings for your service

1. For **Cluster VPC**, if you selected the EC2 launch type, choose the VPC in which your container instances reside. If you selected the Fargate launch type, select the VPC that the Fargate tasks should use. Ensure that the VPC you choose was not configured to require dedicated hardware tenancy as that is not supported by Fargate tasks.

2. For **Subnets**, choose the available subnets for your service task placement.

   Important
   Only private subnets are supported for the `awsvpc` network mode. Because tasks do not receive public IP addresses, a NAT gateway is required for outbound internet access, and inbound internet traffic should be routed through a load balancer.

3. For **Security groups**, a security group has been created for your service's tasks, which allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.
4. For **Auto-assign Public IP**, choose whether to have your tasks receive a public IP address. If you are using Fargate tasks, a public IP address needs to be assigned to the task's elastic network interface, with a route to the internet, or a NAT gateway that can route requests to the internet, in order for the task to pull container images.

(Optional) Health Check Grace Period

If your service's tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 7,200 seconds during which the ECS service scheduler ignores health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. This is only valid if your service is configured to use a load balancer.

- **Health check grace period**: Enter the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started.

(Optional) Configuring Your Service to Use a Load Balancer

If you are not configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, (Optional) Configuring Your Service to Use Service Auto Scaling (p. 113).

If you have an available Elastic Load Balancing load balancer configured, you can attach it to your service with the following procedures, or you can configure a new load balancer. For more information, see Creating a Load Balancer (p. 84).

Note
You must create your Elastic Load Balancing load balancer resources before following these procedures.

First, you must choose the load balancer type to use with your service. Then you can configure your service to work with the load balancer.

To choose a load balancer type

1. If you have not done so already, follow the basic service creation procedures in Configuring Basic Service Parameters (p. 108).
2. For **Load balancer type**, choose the load balancer type to use with your service:

   Application Load Balancer

   Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

Network Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing.

Classic Load Balancer

Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of the advanced features available to them.

3. For **Select IAM role for service**, choose **Create new role** to create a new role for your service, or select an existing IAM role to use for your service (by default, this is `ecsServiceRole`).

   Important
   If you choose to use an existing `ecsServiceRole` IAM role, you must verify that the role has the proper permissions to use Application Load Balancers and Classic Load Balancers. For more information, see Amazon ECS Service Scheduler IAM Role (p. 162).

4. For **ELB Name**, choose the name of the load balancer to use with your service. Only load balancers that correspond to the load balancer type you selected earlier are visible here.
5. The next step depends on the load balancer type for your service. If you've chosen an Application Load Balancer, follow the steps in To configure an Application Load Balancer (p. 111). If you've chosen a Network Load Balancer, follow the steps in To configure a Network Load Balancer (p. 112).

To configure an Application Load Balancer

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.
2. For **Listener port**, choose the listener port and protocol of the listener that you created in Creating an Application Load Balancer (p. 85) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol in **Listener protocol**.
3. For **Target group name**, choose the target group that you created in Creating an Application Load Balancer (p. 85) (if applicable), or choose **create new** to create a new target group.

   Important
   If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), your target group must use `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:
   - For **Target group name**, enter a name for your target group.
   - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
   - For **Path pattern**, if your listener does not have any existing rules, the default path pattern (`/`) is used. If your listener already has a default rule, then you must enter a path pattern that matches traffic that you want to have sent to your service's target group. For example, if your service is a web application called `web-app`, and you want traffic that matches `http://my-elb-url/web-app` to route to your service, then you would enter `/web-app*` as your path pattern. For more information, see ListenerRules in the *User Guide for Application Load Balancers*.
   - For **Health check path**, enter the path to which the load balancer should send health check pings.
5. When you are finished configuring your Application Load Balancer, choose **Next step**.

To configure a Network Load Balancer

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.
2. For **Listener port**, choose the listener port and protocol of the listener that you created in Creating an Application Load Balancer (p. 85) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol in **Listener protocol**.
3. For **Target group name**, choose the target group that you created in Creating an Application Load Balancer (p. 85) (if applicable), or choose **create new** to create a new target group.

   Important
   If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), your target group must use `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
4. (Optional) If you chose to create a new target group, complete the following fields as follows:
   - For **Target group name**, enter a name for your target group.
   - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
   - For **Health check path**, enter the path to which the load balancer should send health check pings.
5. When you are finished configuring your Network Load Balancer, choose **Next Step**.

(Optional) Configuring Your Service to Use Service Discovery

Your Amazon ECS service can optionally enable service discovery integration, which allows your service to be discoverable via DNS. For more information, see Service Discovery (p. 97).

To configure service discovery

1. If you have not done so already, follow the basic service creation procedures in Configuring Basic Service Parameters (p. 108).
2. On the **Configure network** page, select **Enable service discovery integration**.
3. For **Namespace**, select an existing Amazon Route 53 namespace, if you have one, otherwise select **create new private namespace**.
4. If creating a new namespace, for **Namespace name** enter a descriptive name for your namespace. This is the name used for the Amazon Route 53 hosted zone.
5. For **Configure service discovery service**, select to either create a new service discovery service or select an existing one.
6. If creating a new service discovery service, for **Service discovery name** enter a descriptive name for your service discovery service. This is used as the prefix for the DNS records to be created.
7. Select **Enable ECS task health propagation** if you want health checks enabled for your service discovery service.
8. For **DNS record type**, select the DNS record type to create for your service. Amazon ECS service discovery only supports A and SRV records, depending on the network mode that your task definition specifies. For more information about these record types, see DnsRecord.
   - If the task definition that your service task specifies uses the `bridge` or `host` network mode, only type SRV records are supported. Choose a container name and port combination to associate with the record.
   - If the task definition that your service task specifies uses the `awsvpc` network mode, select either the A or SRV record type. If the type A DNS record is selected, skip to the next step. If the type

SRV is selected, specify either the port that the service can be found on or a container name and port combination to associate with the record.

9. For **TTL**, enter the resource record cache time to live (TTL), in seconds. This value determines how long a record set is cached by DNS resolvers and by web browsers.
10. Choose **Next step**.

(Optional) Configuring Your Service to Use Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms.

Amazon ECS Service Auto Scaling supports the following types of scaling policies:

- Target Tracking Scaling Policies (p. 91)—Increase or decrease the number of tasks that your service runs based on a target value for a specific metric. This is similar to the way that your thermostat maintains the temperature of your home. You select temperature and the thermostat does the rest.
- Step Scaling Policies (p. 95)—Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach.

For more information, see Service Auto Scaling (p. 90).

To configure basic Service Auto Scaling parameters

1. If you have not done so already, follow the basic service creation procedures in Configuring Basic Service Parameters (p. 108).
2. On the **Set Auto Scaling** page, select **Configure Service Auto Scaling to adjust your service's desired count**.
3. For **Minimum number of tasks**, enter the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted below this amount.
4. For **Desired number of tasks**, this field is pre-populated with the value that you entered earlier. You can change your service's desired count at this time, but this value must be between the minimum and maximum number of tasks specified on this page.
5. For **Maximum number of tasks**, enter the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted above this amount.
6. For **IAM role for Service Auto Scaling**, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose **Create new role** and the role is created for you. For future reference, the role that is created for you is called `ecsAutoscaleRole`. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 164).
7. The following procedures provide steps for creating either target tracking or step scaling policies for your service. Choose your desired scaling policy type.

To configure target tracking scaling policies for your service

These steps help you create target tracking scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale-out alarm to increase the desired count of your service, and a scale-in alarm to decrease the desired count of your service.

1. For **Scaling policy type**, choose **Target tracking**.
2. For **Policy name**, enter a descriptive name for your policy.
3. For **ECS service metric**, choose the metric you want to track.
4. For **Target value**, enter the metric value you want the policy to maintain.
5. For **Scale-out cooldown period**, enter the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
6. For **Scale-in cooldown period**, enter the amount of time, in seconds, after a scale in activity completes before another scale in activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
7. (Optional) Choose **Disable scale-in**, if you wish to disable the scale-in actions for this policy. This allows you to create a separate scaling policy for scale-in later if you would like.
8. Choose **Next step**.

To configure step scaling policies for your service

These steps help you create step scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a **Scale out** alarm to increase the desired count of your service, and a **Scale in** alarm to decrease the desired count of your service.

1. For **Scaling policy type**, choose **Step scaling**.
2. For **Policy name**, enter a descriptive name for your policy.
3. For **Execute policy when**, select the CloudWatch alarm that you want to use to scale your service up or down.

   You can use an existing CloudWatch alarm that you have previously created, or you can choose to create a new alarm. The **Create new alarm** workflow allows you to create CloudWatch alarms that are based on the `CPUUtilization` and `MemoryUtilization` of the service that you are creating. To use other metrics, you can create your alarm in the CloudWatch console and then return to this wizard to choose that alarm.
4. (Optional) If you've chosen to create a new alarm, complete the following steps.

   a. For **Alarm name**, enter a descriptive name for your alarm. For example, if your alarm should trigger when your service CPU utilization exceeds 75%, you could call the alarm `service_name-cpu-gt-75`.

   b.

   c. For **Alarm threshold**, enter the following information to configure your alarm:

      - Choose the CloudWatch statistic for your alarm (the default value of **Average** works in many cases). For more information, see Statistics in the *Amazon CloudWatch User Guide*.
      - Choose the comparison operator for your alarm and enter the value that the comparison operator checks against (for example, > and `75`).
      - Enter the number of consecutive periods before the alarm is triggered and the period length. For example, two consecutive periods of 5 minutes would take 10 minutes before the alarm triggered. Because your Amazon ECS tasks can scale up and down quickly, consider using a low number of consecutive periods and a short period duration to react to alarms as soon as possible.

   d. Choose **Save** to save your alarm.
5. For **Scaling action**, enter the following information to configure how your service responds to the alarm:

   - Choose whether to add to, subtract from, or set a specific desired count for your service.

- If you chose to add or subtract tasks, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is triggered. If you chose to set the desired count, enter the desired count that your service should be set to when the scaling action is triggered.
- (Optional) If you chose to add or subtract tasks, choose whether the previous value is used as an integer or a percent value of the existing desired count.
- Enter the lower boundary of your step scaling adjustment. By default, for your first scaling action, this value is the metric amount where your alarm is triggered. For example, the following scaling action adds 100% of the existing desired count when the CPU utilization is greater than 75%.

6. (Optional) You can repeat Step 5 (p. 114) to configure multiple scaling actions for a single alarm (for example, to add one task if CPU utilization is between 75-85%, and to add two tasks if CPU utilization is greater than 85%).
7. (Optional) If you chose to add or subtract a percentage of the existing desired count, enter a minimum increment value for **Add tasks in increments of** *N* **task(s)**.
8. For **Cooldown period**, enter the number of seconds between scaling actions.
9. Repeat Step 1 (p. 114) through Step 8 (p. 115) for the **Scale in** policy and choose **Save** to save your Service Auto Scaling configuration.
10. Choose **Next step**.

Review and Create Your Service

After you have configured your basic service definition parameters and optionally configured your service to use a load balancer, you can review your configuration. Then, choose **Create Service** to finish creating your service.

> **Note**
> After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

Updating a Service

You can update a running service to change the number of tasks that are maintained by a service, which task definition is used by the tasks, or if you are using Fargate tasks you can change the platform version your service uses. If you have an application that needs more capacity, you can scale up your service. If you have unused capacity that you would like to scale down, you can reduce the number of desired tasks in your service and free up resources.

If you have updated the Docker image of your application, you can create a new task definition with that image and deploy it to your service. The service scheduler uses the minimum healthy percent and maximum percent parameters (in the service's deployment configuration) to determine the deployment strategy.

> **Note**
> If your updated Docker image uses the same tag as what is in the existing task definition for your service (for example, `my_image:latest`), you do not need to create a new revision of your task definition. You can update the service using the procedure below, keep the current settings

for your service, and select **Force new deployment**. The new tasks launched by the deployment pull the current image/tag combination from your repository when they start. The **Force new deployment** option is also used when updating a Fargate task to use a more current platform version when you specify `LATEST`. For example, if you specified `LATEST` and your running tasks are using the `1.0.0` platform version and you want them to relaunch using a newer platform version.

The minimum healthy percent represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a desired number of four tasks and a minimum healthy percent of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that *do not* use a load balancer are considered healthy if they are in the `RUNNING` state; tasks for services that *do* use a load balancer are considered healthy if they are in the `RUNNING` state and the container instance on which it is hosted is reported as healthy by the load balancer. The default value for minimum healthy percent is 50% in the console and 100% for the AWS CLI, the AWS SDKs, and the APIs.

The maximum percent parameter represents an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer). This parameter enables you to define the deployment batch size. For example, if your service has a desired number of four tasks and a maximum percent value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default value for maximum percent is 200%.

When the service scheduler replaces a task during an update, if a load balancer is used by the service, the service first removes the task from the load balancer and waits for the connections to drain. Then the equivalent of **docker stop** is issued to the containers running in the task. This results in a `SIGTERM` signal and a 30-second timeout, after which `SIGKILL` is sent and the containers are forcibly stopped. If the container handles the `SIGTERM` signal gracefully and exits within 30 seconds from receiving it, no `SIGKILL` signal is sent. The service scheduler starts and stops tasks as defined by your minimum healthy percent and maximum percent settings.

> **Important**
> If you are changing the ports used by containers in a task definition, you may need to update your container instance security groups to work with the updated ports.
> If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.
> To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service.
> Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To update a running service

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the region that your cluster is in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the name of the cluster in which your service resides.
5. On the **Cluster: *name*** page, choose **Services**.
6. Check the box to the left of the service to update and choose **Update**.
7. On the **Configure service** page, your service information is pre-populated. Change the task definition, platform version, deployment configuration, or number of desired tasks (or any combination of these) and choose **Next step**.

Note
If you want your service to use a newly updated Docker image with the same tag as what is in the existing task definition (for example, `my_image:latest`), or keep the current settings for your service, select **Force new deployment**. The new tasks launched by the deployment pull the current image/tag combination from your repository when they start. The **Force new deployment** option is also used when updating a Fargate task to use a more current platform version when you specify `LATEST`. For example, if you specified `LATEST` and your running tasks are using the `1.0.0` platform version and you want them to relaunch using a newer platform version.

8. On the **Configure network** page, your network information is pre-populated. Change the health check grace period (if desired) and choose **Next step**.
9. (Optional) You can use Service Auto Scaling to scale your service up and down automatically in response to CloudWatch alarms.
   a. Under **Optional configurations**, choose **Configure Service Auto Scaling**.
   b. Proceed to (Optional) Configuring Your Service to Use Service Auto Scaling (p. 113).
   c. Complete the steps in that section and then return here.
10. Choose **Update Service** to finish and update your service.

Service Throttle Logic

The Amazon ECS service scheduler includes logic that throttles how often service tasks are launched if they repeatedly fail to start.

If tasks for an ECS service repeatedly fail to enter the `RUNNING` state (progressing directly from `PENDING` to `STOPPED`), then the time between subsequent restart attempts is incrementally increased up to 15 minutes. This maximum period is subject to change in the future and should not be considered permanent. This behavior reduces the effect that unstartable tasks have on your Amazon ECS cluster resources or Fargate infrastructure costs. If your service triggers the throttle logic, you receive the following service event message (p. 220):

```
(service service-name) is unable to consistently start tasks successfully.
```

Amazon ECS does not ever stop a failing service from retrying, nor does it attempt to modify it in any way other than increasing the time between restarts. The service throttle logic does not provide any user-tunable parameters.

If you update your service to use a new task definition, your service returns to a normal, non-throttled state immediately. For more information, see Updating a Service (p. 115).

The following are some common causes that trigger this logic:

- A lack of resources with which to host your task, such as ports, memory, or CPU units in your cluster. In this case, you also see the insufficient resource service event message (p. 219).
- The Amazon ECS container agent is unable to pull your task Docker image. This could be due to a bad container image name, image, or tag, or a lack of private registry authentication or permissions. In this case, you also see `CannotPullContainerError` in your stopped task errors (p. 216).

Important
Tasks that are stopped after they reach the `RUNNING` state do not trigger the throttle logic or the associated service event message. For example, if failed Elastic Load Balancing health checks for a service cause a task to be flagged as unhealthy, and Amazon ECS deregisters it and kills the task, this does not trigger the throttle. Even if a task's container command immediately

exits with a non-zero exit code, the task has already moved to the `RUNNING` state. Tasks that fail immediately due to command errors do not trigger the throttle or the service event message.

Resources and Tags

Amazon ECS resources, including task definitions, clusters, tasks, services, and container instances, are assigned an Amazon Resource Name (ARN) and a unique resource identifier (ID). These resources can be tagged with values that you define, to help you organize and identify them.

> **Note**
> Resource tagging is not available in the GovCloud (US-East) region.

The following topics describe resources and tags, and how you can work with them.

Contents

- Amazon Resource Names (ARNs) and IDs (p. 119)
- Tagging Your Amazon ECS Resources (p. 121)
- Amazon ECS Usage Reports (p. 126)

Amazon Resource Names (ARNs) and IDs

When Amazon ECS resources are created, we assign each resource a unique Amazon Resource Name (ARN) and resource identifier (ID). If you are using a command line tool or the Amazon ECS API to work with Amazon ECS, resource ARNs or IDs are required for certain commands. For example, if you are using the stop-task AWS CLI command to stop a task, you must specify the task ARN or ID in the command.

We're gradually introducing a new ARN and resource ID format for Amazon ECS tasks, services and container instances. The following sections describe how the formats are changing. For more information on the transition to the new formats, see Amazon Elastic Container Service FAQ.

Amazon Resource Name (ARN) Format

Some resources have a friendly name (for example, a service named `production`). However, sometimes you are required to specify a resource using the Amazon Resource Name (ARN) format. We're gradually introducing a new ARN format for Amazon ECS tasks, services and container instances which includes the cluster name. For details on how to opt in to the new ARN format, see Working with Account Settings (p. 120).

> **Note**
> The new ARN format is not available in the GovCloud (US-East) region.

The following table shows both the current (old) format and the new format for each resource type:

Resource type	ARN
Container instance	Old: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:container-instance/`*`container-instance-id`* New: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:container-instance/`*`cluster-name`*`/`*`container-instance-id`*
Amazon ECS service	Old: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`service-name`* New: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster-name`*`/`*`service-name`*

Resource type	ARN
Amazon ECS task	Old: `arn:aws:ecs:region:aws_account_id:task/task-id` New: `arn:aws:ecs:region:aws_account_id:task/cluster-name/task-id`

Resource ID Length

A resource ID takes the form of a unique combination of letters and numbers. We're gradually introducing shorter length IDs for Amazon ECS tasks and container instances. The length of the ID was in an 36-character format; the new IDs are in a 32-character format that will not include any hyphens. For details on how to opt in to the new resource ID format, see Working with Account Settings (p. 120).

> **Note**
> The new resource ID format is not available in the GovCloud (US-East) region.

Timeline

The new formats have an opt-in period, during which you can choose to accept the new formats. The following lists the important dates related to this change:

- Now until March 31, 2019 - The option to opt-in to the new Amazon Resource Name (ARN) and resource ID formats begins. The ability to opt-in and opt-out is provided on a per-region basis. Any new accounts created will be opted-out by default.
- April 1, 2019 - December 31, 2019 - All new accounts will be opted-in by default. The ability to opt-in and opt-out will continue to be available on a per-region basis.
- January 1, 2020 - All accounts will be opted-in by default. All new resources created will receive the new format.

You can opt-in or opt-out of the new Amazon Resource Name (ARN) and resource ID format at any time during the opt-in period. After you have opted in, any new resources that you create are created with the new format.

> **Note**
> A resource ID does not change after it's created. Therefore, opting in or out of the new format during the opt-in period does not affect your existing resource IDs.

Working with Account Settings

For each Region, you can opt in or opt out of the new ARN and resource ID format at the account-level or for a specific IAM user or role. Each IAM user or role can opt in or opt out for themselves. The root user has the ability to opt in or opt out any specific IAM role or user on the account. If the account setting for the root user is changed, it sets the default setting for all the IAM users and roles for which no individual account setting has been set.

The opt in and opt out account setting can be set for each Amazon ECS resource separately. The ARN and resource ID format of a resource will be defined by the opt-in status of the IAM user or role that created the resource.

Only resources launched after opting in will receive the new ARN and resource ID format. All existing resources keep their ARN and resource ID and are not affected. For example, if you opt in to the new format for all resources and have an existing Amazon ECS service, it will keep its old ARN. After you opt in, any new tasks launched by that service will receive the new ARN format.

Topics

- Viewing Account Settings (p. 121)

- Modifying Account Settings (p. 121)

Viewing Account Settings

You can use the AWS Management Console and AWS CLI tools to view the resource types that support the new ARN and ID formats.

To view your account settings using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation bar at the top of the screen, select the region for which to view your account settings.
3. From the dashboard, under **Clusters**, choose **Configure ECS ARN setting**.
4. On the **Amazon ECS ARN and resource ID settings** section, you can view your account settings for each resource type for the authenticated IAM user and role.

Modifying Account Settings

You can use the AWS Management Console and AWS CLI tools to modify account settings for resource types that are still within their opt-in period.

To modify account settings using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation bar at the top of the screen, select the region for which to modify the longer ID settings.
3. From the dashboard, choose **Configure ECS ARN setting**.
4. To enable the new ARN and ID format for each supported resource type, select the option in the **My IAM user/role opt-in setting** column. This will change the opt in setting for the authenticated IAM user.
5. On the confirmation screen, choose **Confirm** to save the selection.

Tagging Your Amazon ECS Resources

To help you manage your Amazon ECS tasks, services, task definitions, clusters, and container instances, you can optionally assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.

> **Important**
> To use this feature, it requires that you opt-in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see Amazon Resource Names (ARNs) and IDs (p. 119).

> **Note**
> Resource tagging is not available in the GovCloud (US-East) region.

Contents

- Tag Basics (p. 122)
- Tagging Your Resources (p. 122)
- Tag Restrictions (p. 123)
- Tagging Your Resources for Billing (p. 123)

- Working with Tags Using the Console (p. 123)
- Working with Tags Using the CLI or API (p. 124)

Tag Basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment. This is useful when you have many resources of the same type—you can quickly identify a specific resource based on the tags you've assigned to it. For example, you could define a set of tags for your account's Amazon ECS container instances that helps you track each container instance's owner and stack level.

We recommend that you devise a set of tag keys that meets your needs for each resource type. Using a consistent set of tag keys makes it easier for you to manage your resources. You can search and filter the resources based on the tags you add.

Tags don't have any semantic meaning to Amazon ECS and are interpreted strictly as a string of characters. Also, tags are not automatically assigned to your resources. You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value. If you delete a resource, any tags for the resource are also deleted.

You can work with tags using the AWS Management Console, the AWS CLI, and the Amazon ECS API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

Tagging Your Resources

You can tag new or existing Amazon ECS tasks, services, task definitions, and clusters.

If you're using the Amazon ECS console, you can apply tags to new resources when they are created or existing resources by using the **Tags** tab on the relevant resource page at any time.

If you're using the Amazon ECS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action or use the `TagResource` API action to apply tags to existing resources. For more information, see TagResource.

Additionally, some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, we roll back the resource creation process. This ensures that resources are either created with tags or not created at all, and that no resources are left untagged at any time. By tagging resources at the time of creation, you can eliminate the need to run custom tagging scripts after resource creation.

The following table describes the Amazon ECS resources that can be tagged, and the resources that can be tagged on creation.

Tagging Support for Amazon ECS Resources

Resource	Supports tags	Supports tagging on creation (Amazon ECS API, AWS CLI, AWS SDK)
Amazon ECS tasks	Yes	Yes

Resource	Supports tags	Supports tagging on creation (Amazon ECS API, AWS CLI, AWS SDK)
Amazon ECS services	Yes	Yes
Amazon ECS task definitions	Yes	Yes
Amazon ECS clusters	Yes	Yes

Tag Restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are: letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case-sensitive.
- Don't use the `aws:` prefix for either keys or values; it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags per resource limit.

Tagging Your Resources for Billing

When enabling Amazon ECS managed tags, Amazon ECS will automatically tag all newly launched tasks with the cluster name. For tasks that belong to a service, they will be tagged with the service name as well. These managed tags are helpful when reviewing cost allocation after enabling them in your Cost & Usage Report. For more information, see Amazon ECS Usage Reports (p. 126).

To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information about setting up a cost allocation report with tags, see The Monthly Cost Allocation Report in the *AWS Billing and Cost Management User Guide*.

> **Important**
> To use this feature, it requires that you opt-in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see Amazon Resource Names (ARNs) and IDs (p. 119).

> **Note**
> If you've just enabled reporting, data for the current month is available for viewing after 24 hours.

Working with Tags Using the Console

Using the Amazon ECS console, you can manage the tags associated with new or existing tasks, services, task definitions, clusters, or container instances.

When you select a resource-specific page in the Amazon ECS console, it displays a list of those resources. For example, if you select **Clusters** from the navigation pane, the console displays a list of Amazon ECS clusters. When you select a resource from one of these lists (for example, a specific cluster), if the resource supports tags, you can view and manage its tags on the **Tags** tab.

Contents

- Adding Tags on an Individual Resource During Launch (p. 124)
- Adding and Deleting Tags on an Individual Resource (p. 124)

Adding Tags on an Individual Resource During Launch

The following resources allow you to specify tags when you create the resource.

Task	Console
Run one or more tasks.	Running Tasks (p. 68)
Create a service.	Creating a Service (p. 108)
Register a task definition.	Creating a Task Definition (p. 26)
Create a cluster.	Creating a Cluster (p. 20)

Adding and Deleting Tags on an Individual Resource

Amazon ECS allows you to add or delete tags associated with your clusters, services, tasks, and task definitions directly from the resource's page.

To add a tag to an individual resource

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, select a resource type (for example, **Clusters**).
4. Select the resource from the resource list and choose **Tags**, **Edit**.
5. In the **Edit Tags** dialog box, specify the key and value for each tag, and then choose **Save**.

To delete a tag from an individual resource

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose a resource type (for example, **Clusters**).
4. Select the resource from the resource list and choose **Tags**, **Edit**.
5. On the **Edit Tags** page, select the **Delete** icon for each tag you want to delete, and choose **Save**.

Working with Tags Using the CLI or API

Use the following to add, update, list, and delete the tags for your resources. The corresponding documentation provides examples.

Tagging Support for Amazon ECS Resources

Task	AWS CLI	API Action
Add or overwrite one or more tags.		TagResource
Delete one or more tags.		UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws ecs tag-resource --resource-arn resource_ARN --tags key=stack,values=dev
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws ecs untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws ecs list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

Task	AWS CLI	AWS Tools for Windows PowerShell	API Action
Run one or more tasks.	run-task	Start-ECSTask	RunTask
Create a service.	create-service	New-ECSService	CreateService
Register a task definition.	register-task-definition	Register-ECSTaskDefinition	RegisterTaskDefinition
Create a cluster.	create-cluster	New-ECSCluster	CreateCluster

The following examples demonstrate how to apply tags when you create resources.

Example 1: Create a cluster and apply a tag

The following command creates a cluster named `devcluster` and adds a tag with key `team` and value `devs`.

```
aws ecs create-cluster --cluster-name devcluster --tags key=team,values=devs
```

Example 2: Create a service and apply a tag

The following command creates a service named `application` and adds a tag with key `stack` and value `dev`.

```
aws ecs create-service --service-name application --task-definition task-def-app --tags
 key=stack,values=dev
```

The `--propagateTags` parameter can be used to copy the tags from either a task definition or a service to the tasks in a service. The following command creates a service with tags and propagates them to the tasks in that service.

```
aws ecs create-service --service-name application --task-definition task-def-app --tags
 key=stack,values=dev --propagateTags Service
```

Amazon ECS Usage Reports

AWS provides a free reporting tool called Cost Explorer that enables you to analyze the cost and usage of your Amazon ECS resources.

Cost Explorer is a free tool that you can use to view charts of your usage and costs. You can view data from the last 13 months, and forecast how much you are likely to spend for the next three months. You can use Cost Explorer to see patterns in how much you spend on AWS resources over time, identify areas that need further inquiry, and see trends that you can use to understand your costs. You also can specify time ranges for the data, and view time data by day or by month.

The metering data in your Cost & Usage Report shows usage across all of your Amazon ECS tasks. The metering data includes `vCPU-Hours` and memory `GB-Hours` for each task that was run. For tasks using the Fargate launch type, you will also see the cost associated with those tasks. You can also use the Amazon ECS managed tags to identify the service or cluster that each task belongs to. For more information, see Tagging Your Resources for Billing (p. 123).

> **Important**
> The metering data is only viewable for tasks launched on or after November 16, 2018. Tasks launched prior to this date will not show metering data.

Here's an example of some of the fields you can sort cost allocation data by when using Cost Explorer:

- Cluster name
- Service name
- Resource tags
- Launch type
- Region
- Usage type

For more information about creating an AWS Cost and Usage Report, see AWS Cost and Usage Report in the *AWS Billing and Cost Management User Guide*.

Monitoring Amazon ECS

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon ECS and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon ECS; however, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

When you are using the Fargate launch type, you get CPU and memory utilization metrics for each of your services to assist in the monitoring of your environment.

The next step is to establish a baseline for normal Amazon ECS performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon ECS, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

Topics

- Monitoring Tools (p. 127)
- Amazon ECS CloudWatch Metrics (p. 128)
- Amazon ECS Event Stream for CloudWatch Events (p. 134)

Monitoring Tools

AWS provides various tools that you can use to monitor Amazon ECS. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use the following automated monitoring tools to watch Amazon ECS and report when something is wrong:

- Amazon CloudWatch alarms – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Amazon ECS CloudWatch Metrics (p. 128).

For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see Service Auto Scaling (p. 90).

- Amazon CloudWatch Logs – Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. This is the only supported method for accessing logs for tasks using the Fargate launch type. For more information, see Using the awslogs Log Driver (p. 55).
- Amazon CloudWatch Events – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see Amazon ECS Event Stream for CloudWatch Events (p. 134) in this guide and Using Events in the *Amazon CloudWatch User Guide*.
- AWS CloudTrail log monitoring – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Logging Amazon ECS API Calls with AWS CloudTrail (p. 213) in this guide, and Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.

Manual Monitoring Tools

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

- CloudWatch home page:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

  In addition, you can use CloudWatch to do the following:
  - Create customized dashboards to monitor the services you care about.
  - Graph metric data to troubleshoot issues and discover trends.
  - Search and browse all your AWS resource metrics.
  - Create and edit alarms to be notified of problems.
- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

Amazon ECS CloudWatch Metrics

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Topics

- Enabling CloudWatch Metrics (p. 129)
- Available Metrics and Dimensions (p. 129)

- Service Utilization (p. 131)
- Service RUNNING Task Count (p. 132)
- Viewing Amazon ECS Metrics (p. 133)

Enabling CloudWatch Metrics

Any task or service using the Fargate launch type is enabled for CloudWatch CPU and memory utilization metrics automatically, so there is no need to take any manual steps.

Available Metrics and Dimensions

The metrics and dimensions that Amazon ECS sends to Amazon CloudWatch are listed below.

Amazon ECS Metrics

Amazon ECS provides metrics for you to monitor the CPU and memory reservation and utilization across your cluster as a whole, and the CPU and memory utilization on the services in your clusters.

The metrics made available will depend on the launch type of the tasks and services in your clusters. If you are using the Fargate launch type for your services then CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the Amazon EC2 launch type you will own and need to monitor the EC2 instances that make your underlying infrastructure so additional CPU and memory reservation and utilization metrics are made available at the cluster, service, and task level.

Amazon ECS sends the following metrics to CloudWatch every minute. When Amazon ECS collects metrics, it collects multiple data points every minute. It then aggregates them to one data point before sending the data to CloudWatch. So in CloudWatch, one sample count is actually the aggregate of multiple data points during one minute.

The `AWS/ECS` namespace includes the following metrics.

Metric	Description
`CPUReservation`	The percentage of CPU units that are reserved by running tasks in the cluster. Cluster CPU reservation (this metric can only be filtered by `ClusterName`) is measured as the total CPU units that are reserved by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. This metric is only used for tasks using the EC2 launch type. Valid Dimensions: `ClusterName` Valid Statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent
`CPUUtilization`	The percentage of CPU units that are used in the cluster or service. Cluster CPU utilization (metrics that are filtered by `ClusterName` without `ServiceName`) is measured as the total CPU units in use by Amazon ECS tasks on

Metric	Description
	the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. Cluster CPU utilization metrics are only used for tasks using the EC2 launch type. 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. Service CPU utilization metrics are used for tasks using both the Fargate and the EC2 launch type. Valid Dimensions: `ClusterName`, `ServiceName` Valid Statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent
`MemoryReservation`	The percentage of memory that is reserved by running tasks in the cluster. Cluster memory reservation (this metric can only be filtered by `ClusterName`) is measured as the total memory that is reserved by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. This metric is only used for tasks using the EC2 launch type. Valid Dimensions: `ClusterName` Valid Statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent

Metric	Description
`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. Cluster memory utilization metrics are only used for tasks using the EC2 launch type. 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. Service memory utilization metrics are used for tasks using both the Fargate and the EC2 launch type. Valid Dimensions: `ClusterName`, `ServiceName` Valid Statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent

Note
If you are using tasks with the EC2 launch type and have Linux container instances, the Amazon ECS container agent relies on Docker `stats` metrics to gather CPU and memory data for each container running on the instance. If you are using an Amazon ECS agent prior to version 1.14.0, ECS includes filesystem cache usage when reporting memory utilization to CloudWatch so your CloudWatch graphs show a higher than actual memory utilization for tasks. To remediate this, starting with Amazon ECS agent version 1.14.0, the Amazon ECS container agent excludes the filesystem cache usage from the memory utilization metric. This change does not impact the out-of-memory behavior of containers.

Dimensions for Amazon ECS Metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions:

Dimension	Description
`ClusterName`	This dimension filters the data you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName`.
`ServiceName`	This dimension filters the data you request for all resources in a specified service within a specified cluster.

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 specified in

the service's task definition. This metric is supported for services with tasks using the Fargate launch type.

```
                                    (Total CPU units used by tasks in service) x 100
Service CPU utilization =
 ---------------------------------------------------------------------------
                          (Total CPU units specified 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 specified in task definition) x (number
 of tasks in service)
```

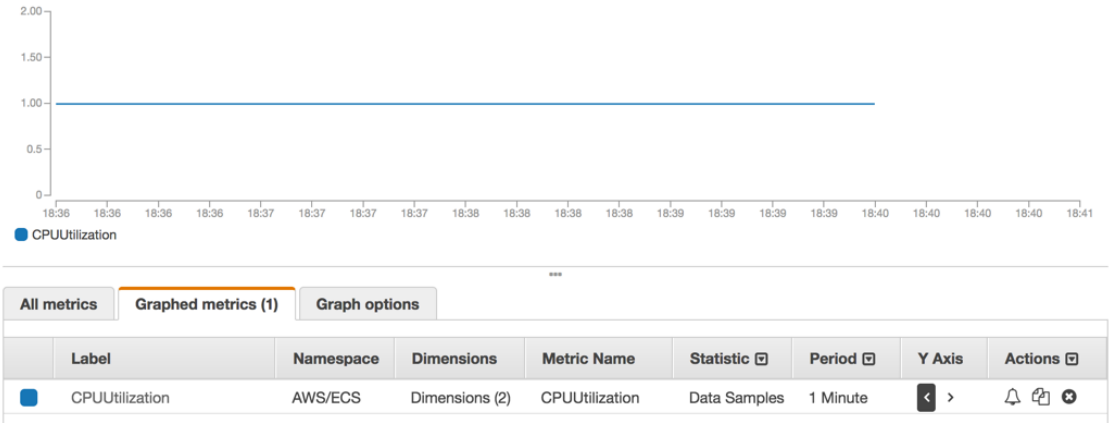
Each minute, the Amazon ECS container agent associated with each task calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks owned 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 specified for the service in the service's task definition. If you specify a soft limit (`memoryReservation`), then it is used to calculate the amount of reserved memory. Otherwise, the hard limit (`memory`) is used. For more information about hard and soft limits, see Task Definition Parameters.

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. On the navigation pane, choose **Metrics** section .
3. On the **All metrics** tab, choose **ECS**.
4. Choose **ClusterName**, **ServiceName** and then choose any metric (either `CPUUtilization` or `MemoryUtilization`) that corresponds to the service in which to view running tasks.
5. On the **Graphed metrics** tab, change **Period** to **1 Minute** and **Statistic** to **Sample Count**.
6. The value displayed in the graph indicates the number of `RUNNING` tasks in the service.



Viewing Amazon ECS Metrics

After you have enabled CloudWatch metrics for Amazon ECS, you can view those metrics in both the Amazon ECS and CloudWatch consoles. The Amazon ECS console provides a 24-hour maximum, minimum, and average view of your service metrics. 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 Service Metrics in the Amazon ECS Console (p. 133)
- Viewing Amazon ECS Metrics in the CloudWatch Console (p. 133)

Viewing Service Metrics in the Amazon ECS Console

Amazon ECS 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, see Service Utilization (p. 131).

To view service metrics in the console

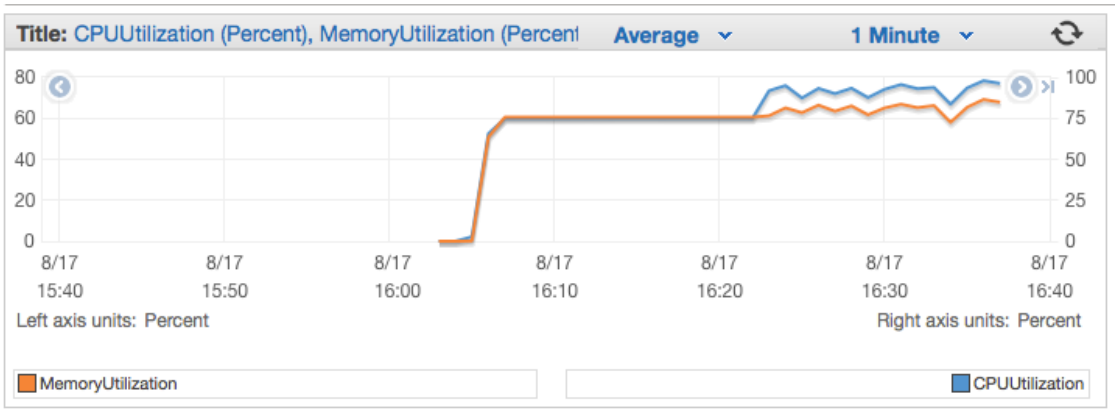
1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Select the cluster that contains the service for which to view metrics.
3. On the **Cluster: *`cluster-name`*** page, choose **Services**.
4. Choose the service for which to view metrics.
5. On the **Service: *`service-name`*** page, choose **Metrics**.

Viewing Amazon ECS Metrics in the CloudWatch Console

Amazon ECS 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 Service Utilization (p. 131) and the Service `RUNNING` Task Count (p. 132). For more information about CloudWatch, see the Amazon CloudWatch User Guide.

To view metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the **Metrics** section in the left navigation, choose **ECS**.
3. Choose the metrics to view. Cluster metrics are scoped as **ECS > ClusterName** and service utilization metrics are scoped as **ECS > ClusterName, ServiceName**. The example below shows cluster CPU and memory utilization.



Amazon ECS Event Stream for CloudWatch Events

You can use Amazon ECS event stream for CloudWatch Events to receive near real-time notifications regarding the current state of your Amazon ECS clusters. When using the Fargate launch type, you can see the state of your tasks.

Using CloudWatch Events, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters, and to monitor the state of clusters in near-real time. You can eliminate scheduling and monitoring code that continuously polls the Amazon ECS service for status changes, and instead handle Amazon ECS state changes asynchronously using any CloudWatch Events target, such as AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, and Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see Handling Events (p. 136).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics

- Amazon ECS Events (p. 134)
- Handling Events (p. 136)
- Tutorial: Listening for Amazon ECS CloudWatch Events (p. 137)
- Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events (p. 139)

Amazon ECS Events

Amazon ECS tracks the state of each of your task. If the state of a task changes, an event is triggered and is sent to CloudWatch Events. These events are classified as task state change events. These events and their possible causes are described in greater detail in the following sections.

> **Note**
> Amazon ECS may add other event types, sources, and details in the future. If you are programmatically deserializing event JSON data, make sure that your application is prepared to handle unknown properties to avoid issues if and when these additional properties are added.

Events contain two `version` fields; one in the main body of the event, and one in the `detail` object of the event.

- The version in the main body of the event is set to `0` on all events. For more information about CloudWatch Events parameters, see Events and Event Patterns in the *Amazon CloudWatch Events User Guide*.
- The version in the `detail` object of the event describes the version of the associated resource. Each time a resource changes state, this version is incremented. Because events can be sent multiple times, this field allows you to identify duplicate events (they have the same version in the `detail` object). If you are replicating your task state with CloudWatch Events, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in CloudWatch Events for the resource (inside the `detail` object) to verify that the version in your event stream is current.

Topics

- Task State Change Events (p. 135)

Task State Change Events

The following scenarios trigger task state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations (either directly, or with the AWS Management Console, AWS CLI, or SDKs).

Starting or stopping tasks creates new task resources or modifies the state of existing task resources.

The Amazon ECS service scheduler starts or stops a task.

Starting or stopping tasks creates new task resources or modifies the state of existing task resources.

The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation.

The Amazon ECS container agent monitors the state of your tasks and it reports any state changes (for example, from `PENDING` to `RUNNING`, or from `RUNNING` to `STOPPED`.

A container in the task changes state.

The Amazon ECS container agent monitors the state of containers within tasks. For example, if a container that is running within a task stops, this container state change triggers an event.

Example Task State Change Event

Task state change events are delivered in the following format (the `detail` section below resembles the Task object that is returned from a DescribeTasks API operation in the *Amazon Elastic Container Service API Reference*). For more information about CloudWatch Events parameters, see Events and Event Patterns in the *Amazon CloudWatch Events User Guide*.

```
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2016-12-06T16:41:06Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
  ],
  "detail": {
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315",
```

```
    "containers": [
      {
        "containerArn": "arn:aws:ecs:us-east-1:111122223333:container/3305bea1-
bd16-4217-803d-3e0482170a17",
        "exitCode": 0,
        "lastStatus": "STOPPED",
        "name": "xray",
        "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
      }
    ],
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "RUNNING",
    "group": "task-group",
    "lastStatus": "RUNNING",
    "overrides": {
      "containerOverrides": [
        {
          "name": "xray"
        }
      ]
    },
    "startedAt": "2016-12-06T16:41:06.8Z",
    "startedBy": "ecs-svc/9223370556150183303",
    "updatedAt": "2016-12-06T16:41:06.975Z",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6f1cebef",
    "taskDefinitionArn": "arn:aws:ecs:us-east-1:111122223333:task-definition/xray:2",
    "version": 4
  }
}
```

Handling Events

Amazon ECS sends events on an "at least once" basis. This means you may receive more than a single copy of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred.

To enable proper ordering of events, the `detail` section of each event contains a `version` property. Events with a higher version property number should be treated as occurring later than events with lower version numbers. Events with matching version numbers can be treated as duplicates.

Example: Handling Events in an AWS Lambda Function

The following example shows a Lambda function written in Python 2.7 that captures task state change events, and saves them to the following Amazon DynamoDB table:

- *ECSTaskState*: Stores the latest state for a task. The table ID is the `taskArn` value of the task.

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print(json.dumps(event))
```

```
    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type of:
aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    elif event["detail-type"] == "ECS Container Instance State Change":
        table_name = "ECSCtrInstanceState"
        id_name =  "containerInstanceArn"
        event_id = event["detail"]["containerInstanceArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting without
saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling")
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event - ignoring")
    else:
        print("Saving new event - ID " + event_id)

        table.put_item(
            Item=new_record
        )
```

Tutorial: Listening for Amazon ECS CloudWatch Events

In this tutorial, you set up a simple AWS Lambda function that listens for Amazon ECS task events and writes them out to a CloudWatch Logs log stream.

Prerequisite: Set Up a Test Cluster

If you do not have a running cluster to capture events from, follow the steps in Getting Started with Amazon ECS using Fargate (p. 16) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Lambda function correctly.

Step 1: Create the Lambda Function

In this procedure, you create a simple Lambda function to serve as a target for Amazon ECS event stream messages.

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create a function**.
3. On the **Author from scratch** screen, do the following:
   a. For **Name**, enter a value.
   b. For **Runtime**, choose **Python 2.7**.
   c. For **Role**, choose **Create a custom role**. A new window pops up enabling you to create a new role for your Lambda function.
   d. On the **AWS Lambda requires access to your resources** screen, accept the defaults and choose **Allow**.
4. Choose **Create function**.
5. In the **Function code** section, edit the sample code to match the following example:

```
import json

def lambda_handler(event, context):
    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

   This is a simple Python 2.7 function that prints the event sent by Amazon ECS. If everything is configured correctly, at the end of this tutorial, you see that the event details appear in the CloudWatch Logs log stream associated with this Lambda function.
6. In the **Function code** section, edit the value of **Handler** to be **eventstream-handler.lambda_handler**.
7. Choose **Save**.

Step 2: Register Event Rule

Next, you create a CloudWatch Events event rule that captures task events coming from your Amazon ECS clusters. This rule captures all events coming from all clusters within the account where it is defined. The task messages themselves contain information about the event source, including the cluster on which it resides, that you can use to filter and sort events programmatically.

> **Note**
> When you use the AWS Management Console to create an event rule, the console automatically adds the IAM permissions necessary to grant CloudWatch Events permission to call your Lambda function. If you are creating an event rule using the AWS CLI, you need to grant this permission explicitly. For more information, see Events and Event Patterns in the *Amazon CloudWatch Events User Guide*.

To route events to your Lambda function

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Events**, **Create rule**.
3. For **Event Source**, choose **ECS** as the event source. By default, the rule applies to all Amazon ECS events for all of your Amazon ECS groups. Alternatively, you can select specific events or a specific Amazon ECS group.
4. For **Targets**, choose **Add target**, for **Target type**, choose **Lambda function**, and then select your Lambda function.
5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for your rule and choose **Create rule**.

Step 3: Test Your Rule

Finally, you create a CloudWatch Events event rule that captures task events coming from your Amazon ECS clusters. This rule captures all events coming from all clusters within the account where it is defined. The task messages themselves contain information about the event source, including the cluster on which it resides, that you can use to filter and sort events programmatically.

To test your rule

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose **Clusters**, **default**.
3. On the **Cluster : default** screen, choose **Tasks**, **Run new Task**.
4. For **Task Definition**, select the latest version of **console-sample-app-static** and choose **Run Task**.
5. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
6. On the navigation pane, choose **Logs** and select the log group for your Lambda function (for example, **/aws/lambda/***my-function*).
7. Select a log stream to view the event data.

Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events

In this tutorial, you configure a CloudWatch Events event rule that only captures task events where the task has stopped running because one of its essential containers has terminated. The event sends only task events with a specific `stoppedReason` property to the designated Amazon SNS topic.

Prerequisite: Set Up a Test Cluster

If you do not have a running cluster to capture events from, follow the steps in Getting Started with Amazon ECS using Fargate (p. 16) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Amazon SNS topic and CloudWatch Events event rule correctly.

Step 1: Create and Subscribe to an Amazon SNS Topic

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.

To create an Amazon SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v2/home.
2. Choose **Topics**, **Create new topic**.

3. On the **Create new topic** window, for **Topic name**, enter **TaskStoppedAlert** and choose **Create topic**.
4. On the **Topics** window, select the topic that you just created. On the **Topic details: TaskStoppedAlert** screen, choose **Create subscription**.
5. On the **Create Subscription** window, for **Protocol**, choose **Email**. For **Endpoint**, enter an email address to which you currently have access and choose **Create subscription**.
6. Check your email account, and wait to receive a subscription confirmation email message. When you receive it, choose **Confirm subscription**.

Step 2: Register Event Rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

To create an event rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Events**, **Create rule**.
3. Choose **Show advanced options**, **edit**.
4. For **Build a pattern that selects events for processing by your targets**, replace the existing text with the following text:

```
{
"source": [
  "aws.ecs"
],
"detail-type": [
  "ECS Task State Change"
],
"detail": {
  "lastStatus": [
    "STOPPED"
  ],
  "stoppedReason" : [
    "Essential container in task exited"
  ]
}
}
```

   This code defines a CloudWatch Events event rule that matches any event where the `lastStatus` and `stoppedReason` fields match the indicated values. For more information about event patterns, see Events and Event Patterns in the *Amazon CloudWatch User Guide*.
5. For **Targets**, choose **Add target**. For **Target type**, choose **SNS topic**, and then choose **TaskStoppedAlert**.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for your rule and then choose **Create rule**.

Step 3: Test Your Rule

To test your rule, you attempt to run a task that exits shortly after it starts. If your event rule is configured correctly, you receive an email message within a few minutes with the event text.

To test a rule

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

2. Choose **Task Definitions**, **Create new Task Definition**.
3. For **Task Definition Name**, type **WordPressFailure** and choose **Add Container**.
4. For **Container name**, type **Wordpress**, for **Image**, type **wordpress**, and for **Maximum memory (MB)**, type **128**.
5. Choose **Add**, **Create**.
6. On the **Task Definition** screen, choose **Actions**, **Run Task**.
7. For **Cluster**, choose **default**. Choose **Run Task**.
8. On the **Tasks** tab for your cluster, periodically choose the refresh icon until you no longer see your task running. To verify that your task has stopped, for **Desired task status**, choose **Stopped**.
9. Check your email to confirm that you have received an email alert for the stopped notification.

Amazon ECS IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify Amazon ECS resources, or perform tasks using the Amazon ECS API. This means that they also can't do so using the Amazon ECS console or the AWS CLI. To allow IAM users to create or modify resources and perform tasks, you must create IAM policies. Policies grant IAM users permissions to use specific resources and API actions. Then, attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more general information about IAM policies, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Getting Started

An IAM policy must grant or deny permission to use one or more Amazon ECS actions. It must also specify the resources that can be used with the action, which can be all resources, or in some cases, specific resources. The policy can also include conditions that you apply to the resource.

Amazon ECS partially supports resource-level permissions. This means that for some Amazon ECS API actions, you cannot specify which resource a user is allowed to work with for that action; instead, you have to allow users to work with all resources for that action.

Topics

- Policy Structure (p. 142)
- Supported Resource-Level Permissions for Amazon ECS API Actions (p. 146)
- Creating Amazon ECS IAM Policies (p. 149)
- Managed Policies and Trust Relationships (p. 149)
- Amazon ECS Task Execution IAM Role (p. 156)
- Using Service-Linked Roles for Amazon ECS (p. 157)
- Amazon ECS Service Scheduler IAM Role (p. 162)
- Amazon ECS Service Auto Scaling IAM Role (p. 164)
- Amazon ECS Task Role (p. 165)
- CloudWatch Events IAM Role (p. 166)
- IAM Roles for Tasks (p. 168)
- Amazon ECS IAM Policy Examples (p. 171)

Policy Structure

The following topics explain the structure of an IAM policy.

Topics

- Policy Syntax (p. 143)
- Actions for Amazon ECS (p. 143)
- Amazon Resource Names for Amazon ECS (p. 144)
- Condition Keys for Amazon ECS (p. 145)
- Checking that Users Have the Required Permissions (p. 146)

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

There are various elements that make up a statement:

- **Effect:** The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action**: The *action* is the specific API action for which you are granting or denying permission. To learn about specifying *action*, see Actions for Amazon ECS (p. 143).
- **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, use its Amazon Resource Name (ARN). For more information about specifying the *arn* value, see Amazon Resource Names for Amazon ECS (p. 144). For more information about which API actions support which ARNs, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 146). 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 is in effect. For more information about specifying conditions for Amazon ECS, see Condition Keys for Amazon ECS (p. 145).

For more information about example IAM policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 149).

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 Elastic 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. 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. 146).

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/*task-definition-family-name*:*task-definition-revision-number*
Service	arn:aws:ecs:*region*:*account*:service/*service-name*
Task	arn:aws:ecs:*region*:*account*:task/*task-id*
Container	arn:aws:ecs:*region*:*account*:container/*container-id*

Many Amazon ECS API actions accept multiple resources. To specify multiple resources in a single statement, separate their ARNs with commas, as follows:

```
"Resource": ["arn1", "arn2"]
```

For more general information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the *Amazon Web Services General Reference*.

Condition Keys for Amazon ECS

In a policy statement, you can optionally specify conditions that control when it is in effect. Each condition contains one or more key-value pairs. Condition keys are not case-sensitive. We've defined AWS-wide condition keys, plus additional service-specific condition keys.

If you specify multiple conditions, or multiple keys in a single condition, we evaluate them using a logical AND operation. If you specify a single condition with multiple values for one key, we evaluate the condition using a logical OR operation. For permission to be granted, all conditions must be met.

You can also use placeholders when you specify conditions. For more information, see Policy Variables in the *IAM User Guide*.

Amazon ECS implements the AWS-wide condition keys (see Available Keys), plus the following service-specific condition keys.

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*"	ARN, Null

Condition Key	Key/Value Pair	Evaluation Types
	Where *container-instance-arns* is one or more container instance ARNs.	

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. 146). For example policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 149).

Checking that Users Have the Required Permissions

After you've created an IAM policy, we recommend that you check whether it grants users the permissions to use the particular API actions and resources they need before you put the policy into production.

First, create an IAM user for testing purposes, and then attach the IAM policy that you created to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

> **Note**
> You can also test your policies with the IAM Policy Simulator. For more information on the policy simulator, see Working with the IAM Policy Simulator in the *IAM User Guide*.

If the action that you are testing creates or modifies a resource, you should make the request using the `DryRun` parameter (or run the AWS CLI command with the `--dry-run` option). In this case, the call completes the authorization check, but does not complete the operation. For example, you can check whether the user can terminate a particular instance without actually terminating it. If the test user has the required permissions, the request returns `DryRunOperation`; otherwise, it returns `UnauthorizedOperation`.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed and retest until you get the desired results.

> **Important**
> It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the `DecodeAuthorizationMessage` action. For more information, see DecodeAuthorizationMessage in the *AWS Security Token Service API Reference*, and decode-authorization-message in the *AWS CLI Command Reference*.

Supported Resource-Level Permissions for Amazon ECS API Actions

The term *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
DeleteAttributes	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
DeleteCluster	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
DeregisterContainerInstance	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
DescribeClusters	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster1*, arn:aws:ecs:*region*:*account*:cluster/*my-cluster2*	N/A
DescribeContainerInstances	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id1*, arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id2*	ecs:cluster
DescribeTasks	Task arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252a*, arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252b*	ecs:cluster
ListAttributes	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
ListContainerInstances	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
ListTasks	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
Poll	Container instance	ecs:cluster

API action	Resource	Condition keys
	arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	
PutAttributes	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
RegisterContainerInstance	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
RunTask	Task definition arn:aws:ecs:*region*:*account*:task-definition/*hello_world:8*	ecs:cluster
StartTask	Task definition arn:aws:ecs:*region*:*account*:task-definition/*hello_world:8*	ecs:cluster ecs:container-instances
StartTelemetrySession	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
StopTask	Task arn:aws:ecs:*region*:*account*:task/*1abf0f6d-a411-4033-b8eb-a4eed3ad252a*	ecs:cluster
SubmitContainerStateChange	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
SubmitTaskStateChange	Cluster arn:aws:ecs:*region*:*account*:cluster/*my-cluster*	N/A
UpdateContainerAgent	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster
UpdateContainerInstancesState	Container instance arn:aws:ecs:*region*:*account*:container-instance/*container-instance-id*	ecs:cluster

Creating Amazon ECS IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information about IAM policies, see Permissions and Policies in the *IAM User Guide*. For more information about 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**, **Create policy**.
3. On the **Visual editor** tab, choose **Choose a Service**, **Elastic Container Service**.
4. Choose **Select actions** and select the actions to add to the policy. For more information, see Amazon ECS IAM Policy Examples (p. 171).
5. (Optional) Choose **Specify request conditions (optional)** to add conditions to the policy that you are creating. Conditions limit a JSON policy statement's effect. For example, you can specify that a user is allowed to perform the actions on the resources only when that user's request happens within a certain time range. You can also use commonly used conditions to limit whether a user must be authenticated using a multi-factor authentication (MFA) device, or if the request must originate from within a certain range of IP addresses. For lists of all of the context keys that you can use in a policy condition, see AWS Service Actions and Condition Context Keys for Use in IAM Policies.
6. Choose **Review policy**.
7. In the **Name** field, type your own unique name, such as `AmazonECSUserPolicy`.
8. 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. Choose **Permissions**, **Add permissions**.
4. In the **Grant permissions** section, choose **Attach existing policies directly**.
5. Select the custom policy that you created in the previous procedure and choose **Next: Review**.
6. Review your details and choose **Add permissions**.

Managed Policies and Trust Relationships

Amazon ECS and Amazon ECR provide several managed policies and trust relationships that you can attach to IAM users, EC2 instances, and Amazon ECS tasks that allow differing levels of control over resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies.

Topics

- Amazon ECS Managed Policies and Trust Relationships (p. 150)
- Amazon ECR Managed Policies (p. 154)

Amazon ECS Managed Policies and Trust Relationships

Amazon ECS provides several managed policies and trust relationships that you can attach to IAM users, EC2 instances, or Amazon ECS tasks that allow differing levels of control over Amazon ECS resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies. For more information about each API operation mentioned in these policies, see Actions in the *Amazon Elastic Container Service API Reference*.

Topics

- AmazonECS_FullAccess (p. 150)
- AmazonEC2ContainerServiceFullAccess (p. 153)
- AmazonEC2ContainerServiceRole (p. 153)
- AmazonEC2ContainerServiceAutoscaleRole (p. 154)

AmazonECS_FullAccess

This managed policy provides administrative access to Amazon ECS resources and enables ECS features through access to other AWS service resources, including VPCs, Auto Scaling groups, and AWS CloudFormation stacks.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:Describe*",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:PutMetricAlarm",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CancelSpotFleetRequests",
                "ec2:CreateInternetGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
```

```
                "ec2:DeleteSubnet",
                "ec2:DeleteVpc",
                "ec2:Describe*",
                "ec2:DetachInternetGateway",
                "ec2:DisassociateRouteTable",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:RequestSpotFleet",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateRule",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteRule",
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:DescribeTargetGroups",
                "ecs:*",
                "events:DescribeRule",
                "events:DeleteRule",
                "events:ListRuleNamesByTarget",
                "events:ListTargetsByRule",
                "events:PutRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfiles",
                "iam:ListRoles",
                "logs:CreateLogGroup",
                "logs:DescribeLogGroups",
                "logs:FilterLogEvents",
                "route53:GetHostedZone",
                "route53:ListHostedZonesByName",
                "route53:CreateHostedZone",
                "route53:DeleteHostedZone",
                "route53:GetHealthCheck",
                "servicediscovery:CreatePrivateDnsNamespace",
                "servicediscovery:CreateService",
                "servicediscovery:GetNamespace",
                "servicediscovery:GetOperation",
                "servicediscovery:GetService",
                "servicediscovery:ListNamespaces",
                "servicediscovery:ListServices",
                "servicediscovery:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParametersByPath",
                "ssm:GetParameters",
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/aws/service/ecs*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DeleteInternetGateway",
                "ec2:DeleteRoute",
```

```
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "ec2:ResourceTag/aws:cloudformation:stack-name": "EC2ContainerService-
*"
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsInstanceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com",
                        "ec2.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsAutoscaleRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "application-autoscaling.amazonaws.com",
                        "application-autoscaling.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "ecs.amazonaws.com",
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "ecs.application-autoscaling.amazonaws.com"
```

```
                    ]
                }
            }
        }
    ]
}
```

AmazonEC2ContainerServiceFullAccess

This managed policy allows full administrator access to Amazon ECS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:Describe*",
        "autoscaling:UpdateAutoScalingGroup",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStack*",
        "cloudformation:UpdateStack",
        "cloudwatch:GetMetricStatistics",
        "ec2:Describe*",
        "elasticloadbalancing:*",
        "ecs:*",
        "events:DescribeRule",
        "events:DeleteRule",
        "events:ListRuleNamesByTarget",
        "events:ListTargetsByRule",
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "iam:ListInstanceProfiles",
        "iam:ListRoles",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

AmazonEC2ContainerServiceRole

This managed policy allows Elastic Load Balancing load balancers to register and deregister Amazon ECS container instances on your behalf. For more information, see Amazon ECS Service Scheduler IAM Role (p. 162).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:Describe*",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets"
```

```
      ],
      "Resource": "*"
    }
  ]
}
```

AmazonEC2ContainerServiceAutoscaleRole

This managed policy allows Application Auto Scaling to scale your Amazon ECS service's desired count up and down in response to CloudWatch alarms on your behalf. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 164).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1456535218000",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Stmt1456535243000",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Amazon ECR Managed Policies

Amazon ECR provides several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over Amazon ECR resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies. For more information about each API operation mentioned in these policies, see Actions in the *Amazon Elastic Container Registry API Reference*.

Topics

- AmazonEC2ContainerRegistryFullAccess (p. 154)
- AmazonEC2ContainerRegistryPowerUser (p. 155)
- AmazonEC2ContainerRegistryReadOnly (p. 155)

AmazonEC2ContainerRegistryFullAccess

This managed policy allows full administrator access to Amazon ECR.

```
{
```

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

AmazonEC2ContainerRegistryPowerUser

This managed policy allows power user access to Amazon ECR, which allows read and write access to repositories, but does not allow users to delete repositories or change the policy documents applied to them.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": [
   "ecr:GetAuthorizationToken",
   "ecr:BatchCheckLayerAvailability",
   "ecr:GetDownloadUrlForLayer",
   "ecr:GetRepositoryPolicy",
   "ecr:DescribeRepositories",
   "ecr:ListImages",
   "ecr:DescribeImages",
   "ecr:BatchGetImage",
   "ecr:InitiateLayerUpload",
   "ecr:UploadLayerPart",
   "ecr:CompleteLayerUpload",
   "ecr:PutImage"
  ],
  "Resource": "*"
 }]
}
```

AmazonEC2ContainerRegistryReadOnly

This managed policy allows read-only access to Amazon ECR, such as the ability to list repositories and the images within the repositories, and also to pull images from Amazon ECR with the Docker CLI.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": [
   "ecr:GetAuthorizationToken",
   "ecr:BatchCheckLayerAvailability",
   "ecr:GetDownloadUrlForLayer",
   "ecr:GetRepositoryPolicy",
   "ecr:DescribeRepositories",
   "ecr:ListImages",
   "ecr:DescribeImages",
   "ecr:BatchGetImage"
  ],
  "Resource": "*"
 }]
```

```
}
```

Amazon ECS Task Execution IAM Role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. The following actions are covered by the `AmazonECSTaskExecutionRolePolicy` policy in the task execution role:

- Calls to Amazon ECR to pull the container image
- Calls to CloudWatch to store container application logs

> **Note**
> The task execution role is supported by ECS Agent version 1.16.0 and later.

If you want to use the private registry authentication for tasks feature, it requires you to add additional permissions as an inline policy to the task execution role. For more information, see Private Registry Authentication Required IAM Permissions (p. 62).

For tasks that use the Fargate launch type, the task execution role is required to pull container images from Amazon ECR or to use the awslogs log driver, which is currently the only supported logging option for this launch type. If you are using a public container image, for example a public image from Docker Hub, and are not using a logging configuration then the task execution role is not needed.

The `AmazonECSTaskExecutionRolePolicy` policy is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

The Amazon ECS task execution role is automatically created for you in the console first-run experience; however, you should manually attach the managed IAM policy for tasks 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 task execution role and to attach the managed IAM policy if needed.

To check for the `ecsTaskExecutionRole` 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 `ecsTaskExecutionRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose **Permissions**. Ensure that the **AmazonECSTaskExecutionRolePolicy** managed policy is attached to the role. If the policy is attached, your Amazon ECS task execution role is properly configured. If not, follow the substeps below to attach the policy.

   a. Choose **Attach policy**.

   b. To narrow the available policies to attach, for **Filter**, type **AmazonECSTaskExecutionRolePolicy**.

   c. Check the box to the left of the **AmazonECSTaskExecutionRolePolicy** policy and choose **Attach policy**.

5. Choose **Trust relationships**, **Edit trust relationship**.

6. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

To create the `ecsTaskExecutionRole` IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create role**.
3. In the **Select type of trusted entity** section, choose **Elastic Container Service**.
4. For **Select your use case**, choose **Elastic Container Service Task**, then choose **Next: Permissions**.
5. In the **Attach permissions policy** section, search for **AmazonECSTaskExecutionRolePolicy**, select the policy, and then choose **Next: Review**.
6. For **Role Name**, type `ecsTaskExecutionRole` and choose **Create role**.

Using Service-Linked Roles for Amazon ECS

Amazon Elastic Container Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon ECS. Service-linked roles are predefined by Amazon ECS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon ECS easier because you don't have to manually add the necessary permissions. Amazon ECS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon ECS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Amazon ECS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon ECS

Amazon ECS uses the service-linked role named **AWSServiceRoleForECS** – Role to enable Amazon ECS to manage your cluster..

The AWSServiceRoleForECS service-linked role trusts the following services to assume the role:

- `ecs.amazonaws.com`

The role permissions policy allows Amazon ECS to complete the following actions on the specified resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:Describe*",
                "ec2:DetachNetworkInterface",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:Describe*",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets",
                "route53:ChangeResourceRecordSets",
                "route53:CreateHealthCheck",
                "route53:DeleteHealthCheck",
                "route53:Get*",
                "route53:List*",
                "route53:UpdateHealthCheck",
                "servicediscovery:DeregisterInstance",
                "servicediscovery:Get*",
                "servicediscovery:List*",
                "servicediscovery:RegisterInstance"
            ],
            "Resource": "*"
        }
    ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role.

To allow an IAM entity to create the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to create the service-linked role:

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

To allow an IAM entity to edit the description of the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to edit the description of a service-linked role:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

To allow an IAM entity to delete the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete a service-linked role:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

Creating a Service-Linked Role for Amazon ECS

Under most circumstances, you don't need to manually create a service-linked role. For example, when you create a new cluster (for example, with the Amazon ECS first run, the cluster creation wizard, or the AWS CLI or SDKs), or create or update a service in the AWS Management Console, Amazon ECS creates the service-linked role for you, if it does not already exist.

> **Important**
> The IAM entity that is creating the cluster must have the appropriate IAM permissions to create the service-linked role and apply a policy to it. Otherwise, the automatic creation fails.

Creating a Service-Linked Role in IAM (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (CLI)

Use the following command:

```
$ aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

Editing a Service-Linked Role for Amazon ECS

Amazon ECS does not allow you to edit the AWSServiceRoleForECS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. You can, however, edit the description of the role. For more information, see Modifying a Role in the *IAM User Guide*.

Deleting a Service-Linked Role for Amazon ECS

If you no longer use Amazon ECS, we recommend that you delete the role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all Amazon ECS clusters in all regions before you can delete the service-linked role.

Cleaning up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and delete all Amazon ECS clusters in all AWS Regions.

To check whether the service-linked role has an active session

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and choose the AWSServiceRoleForECS name (not the check box).
3. On the **Summary** page, choose **Access Advisor** and review recent activity for the service-linked role.

   Note
   If you are unsure whether Amazon ECS is using the AWSServiceRoleForECS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

To remove Amazon ECS resources used by the AWSServiceRoleForECS service-linked role

You must delete all Amazon ECS clusters in all AWS Regions before you can delete the AWSServiceRoleForECS role.

- Delete all Amazon ECS clusters in all regions. For more information, see Deleting a Cluster (p. 21).

Deleting a Service-Linked Role in IAM (Console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (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 of the IAM console, choose **Roles**. Then select the check box next to AWSServiceRoleForECS, not the name or row itself.
3. For **Role actions** at the top of the page, choose **Delete role**.

4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. If you want to proceed, choose **Yes, Delete** to submit the service-linked role for deletion.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, after you submit the role for deletion, the deletion task can succeed or fail.

   - If the task succeeds, then the role is removed from the list and a notification of success appears at the top of the page.
   - If the task fails, you can choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

     Note
     You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources.
   - If the task fails and the notification does not include a list of resources, then the service might not return that information. To learn how to clean up the resources for that service, see AWS Services That Work with IAM. Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Deleting a Service-Linked Role in IAM (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name AWSServiceRoleForECS+OPTIONAL-SUFFIX
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   Note
   You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not

report any resources, see AWS Services That Work with IAM. Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Deleting a Service-Linked Role in IAM (AWSAPI)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked roll, call DeleteServiceLinkedRole. In the request, specify the AWSServiceRoleForECS role name.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.
2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   Note
   You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not report any resources, see AWS Services That Work with IAM. Find your service in the table, and choose the **Yes** link to view the service-linked role documentation for that service.

Amazon ECS Service Scheduler IAM Role

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

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

The `AmazonEC2ContainerServiceRole` policy is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
```

```
        "ec2:Describe*",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets"
      ],
      "Resource": "*"
    }
  ]
}
```

Note
The `ec2:AuthorizeSecurityGroupIngress` rule is reserved for future use. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To check for the `ecsServiceRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsServiceRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.
4. Choose the **Permissions** tab.
5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Attach Policy**.
   b. To narrow the available policies to attach, for **Filter**, type **AmazonEC2ContainerServiceRole**.
   c. Check the box to the left of the **AmazonEC2ContainerServiceRole** policy and choose **Attach Policy**.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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**, **Create role**.
3. In the **Select type of trusted entity** section, choose **Elastic Container Service**.

4. In the **Select your use case** section, choose **Elastic Container Service** and choose **Next: Permissions**.
5. In the **Attached permissions policy** section, select the **AmazonEC2ContainerServiceRole** policy and choose **Next: Review**.
6. For **Role Name**, type `ecsServiceRole`, enter a **Role description** and then choose **Create role**.

Amazon ECS Service Auto Scaling IAM Role

Before you can use Service Auto Scaling with Amazon ECS, the Application Auto Scaling service needs permissions to describe your CloudWatch alarms and registered services, as well as permissions to update your Amazon ECS service's desired count on your behalf. These permissions are provided by the Service Auto Scaling IAM role (`ecsAutoscaleRole`).

> **Note**
> IAM users also require permissions to use Service Auto Scaling; these permissions are described in Service Auto Scaling Required IAM Permissions (p. 95). If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p.) or update service (p. 115) workflows, and then use the role for any other service later (in the console or with the AWS CLI or SDKs).

You can use the following procedure to check and see if your account already has Service Auto Scaling.

The `AmazonEC2ContainerServiceAutoscaleRole` policy is shown below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1456535218000",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Stmt1456535243000",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

To check for the Service Auto Scaling role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsAutoscaleRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose the **Permissions** tab.
5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceAutoscaleRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Attach Policy**.
   b. To narrow the available policies to attach, for **Filter**, type `AmazonEC2ContainerServiceAutoscaleRole`.
   c. Select the box to the left of the **AmazonEC2ContainerAutoscaleRole** policy and choose **Attach Policy**.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "application-autoscaling.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create an IAM role for Service Auto Scaling

1. Open the 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 **Select Role Type** section, scroll down and choose **Select** next to the **Amazon Elastic Container Service Autoscale Role** service role.
4. In the **Attach Policy** section, select the **AmazonEC2ContainerServiceAutoscaleRole** policy and then choose **Next Step**.
5. In the **Role Name** field, type `ecsAutoscaleRole` to name the role, and then choose **Next Step**.
6. Review your role information and then choose **Create Role** to finish.

Amazon ECS Task Role

Before you can use IAM roles for tasks, Amazon ECS needs permission to make calls to the AWS APIs on your behalf. These permissions are provided by the Amazon ECS Task Role.

You can create a task IAM role for each task definition that needs permission to call AWS APIs. You simply create an IAM policy that defines which permissions your task should have, and then attach that policy to a role that uses the Amazon ECS Task Role trust relationship policy. For more information, see Creating an IAM Role and Policy for your Tasks (p. 169).

The Amazon ECS Task Role trust relationship is shown below.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

CloudWatch Events IAM Role

Before you can use Amazon ECS scheduled tasks with CloudWatch Events rules and targets, the CloudWatch Events service needs permissions to run Amazon ECS tasks on your behalf. These permissions are provided by the CloudWatch Events IAM role (`ecsEventsRole`).

The CloudWatch Events role is automatically created for you in the AWS Management Console when you configure a scheduled task. For more information, see Scheduled Tasks (`cron`) (p. 70).

The `AmazonEC2ContainerServiceEventsRole` policy is shown below.

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

If your scheduled tasks require the use of the task execution role or a task role override, then you must add `iam:PassRole` permissions for each task execution role or task role override to the CloudWatch IAM role. For more information about the task execution role, see Amazon ECS Task Execution IAM Role (p. 156).

Note
Specify the full ARN of your task execution role or task role override.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/<ecsTaskExecutionRole_name>"
            ]
        }
    ]
}
```

You can use the following procedure to check that your account already has the CloudWatch Events IAM role, and manually create it if needed.

To check for the CloudWatch Events IAM role 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 `ecsEventsRole`. If the role does not exist, use the next procedure to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceEventsRole** 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. To narrow the available policies to attach, for **Filter**, type `AmazonEC2ContainerServiceEventsRole`.
   c. Select the box to the left of the **AmazonEC2ContainerServiceEventsRole** policy and choose **Attach Policy**.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

To create an IAM role for CloudWatch Events

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. In the **Select type of trusted entity** section, choose **Elastic Container Service**. For **Select your use case** choose **Elastic Container Service Task**. Choose **Next: Permissions**.
4. In the **Attach permissions policy** section, select the **AmazonEC2ContainerServiceEventsRole** policy and choose **Next: Review**.
5. For **Role name**, type `ecsEventsRole` to name the role, optionally enter a description, and then choose **Create role**.
6. Review your role information and choose **Create Role**.

To add permissions for the task execution role to the CloudWatch Events IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies**, **Create policy**.
3. Choose **JSON**, paste the following policy, and then choose **Review policy**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/<ecsTaskExecutionRole_name>"
            ]
        }
    ]
}
```

4. For **Name**, type `AmazonECSEventsTaskExecutionRole`, optionally enter a description, and then choose **Create policy**.
5. In the navigation pane, choose **Roles**.
6. Search the list of roles for `ecsEventsRole` and select the role to view the attached policies.
7. Choose **Attach policy**.
8. In the **Attach policy** section, select the **AmazonECSEventsTaskExecutionRole** policy and choose **Attach policy**.

IAM Roles for Tasks

With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or `RunTask` API operation. The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

You define the IAM role to use in your task definitions, or you can use a `taskRoleArn` override when running a task manually with the `RunTask` API operation. The Amazon ECS agent receives a payload message for starting the task with additional fields that contain the role credentials. The Amazon ECS agent sets a unique task credential ID as an identification token and updates its internal credential cache so that the identification token for the task points to the role credentials that are received in the payload. The Amazon ECS agent populates the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable in the `Env` object (available with the **docker inspect** ***`container_id`*** command) for all containers that belong to this task with the following relative URI: `/`*`credential_provider_version`*`/credentials?id=`*`task_credential_id`*.

> **Note**
> When you specify an IAM role for a task, the AWS CLI or other SDKs in the containers for that task use the AWS credentials provided by the task role exclusively and they no longer inherit any IAM permissions from the container instance.

From inside the container, you can query the credentials with the following command:

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

Output:

```
{
    "AccessKeyId": "ACCESS_KEY_ID",
    "Expiration": "EXPIRATION_DATE",
    "RoleArn": "TASK_ROLE_ARN",
    "SecretAccessKey": "SECRET_ACCESS_KEY",
    "Token": "SECURITY_TOKEN_STRING"
}
```

Topics

- Benefits of Using IAM Roles for Tasks (p. 169)
- Creating an IAM Role and Policy for your Tasks (p. 169)
- Using a Supported AWS SDK (p. 170)
- Specifying an IAM Role for your Tasks (p. 171)

Benefits of Using IAM Roles for Tasks

- **Credential Isolation:** A container can only retrieve credentials for the IAM role that is defined in the task definition to which it belongs; a container never has access to credentials that are intended for another container that belongs to another task.
- **Authorization:** Unauthorized containers cannot access IAM role credentials defined for other tasks.
- **Auditability:** Access and event logging is available through CloudTrail to ensure retrospective auditing. Task credentials have a context of `taskArn` that is attached to the session, so CloudTrail logs show which task is using which role.

Creating an IAM Role and Policy for your Tasks

You must create an IAM policy for your tasks to use that specifies the permissions that you would like the containers in your tasks to have. You have several ways to create a new IAM permission policy. You can copy a complete AWS managed policy that already does some of what you're looking for and then customize it to your specific requirements. For more information, see Creating a New Policy in the *IAM User Guide*.

You must also create a role for your tasks to use before you can specify it in your task definitions. You can create the role using the **Amazon Elastic Container Service Task Role** service role in the IAM console. Then you can attach your specific IAM policy to the role that gives the containers in your task the permissions you desire. The procedures below describe how to do this.

> **Note**
> To view the trust relationship for this role, see **Amazon ECS Task Role** (p. 165).

If you have multiple task definitions or services that require IAM permissions, you should consider creating a role for each specific task definition or service with the minimum required permissions for the tasks to operate so that you can minimize the access that you provide for each task.

To create an IAM policy for your tasks

In this example, we create a policy to allow read-only access to an Amazon S3 bucket. You could store database credentials or other secrets in this bucket, and the containers in your task can read the credentials from the bucket and load them into your application.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Follow the steps under one of the following tabs, which shows you how to use the visual or JSON editors.

Using the visual editor

1. For **Service**, choose **S3**.
2. For **Actions**, expand the **Read** option and select **GetObject**.
3. For **Resources**, select **Add ARN** and enter the full ARN of your Amazon S3 bucket.
4. Choose **Review policy**.
5. In the **Review policy** section, for **Name** type your own unique name, such as `AmazonECSTaskS3BucketPolicy`.
6. Choose **Create policy** to finish.

Using the JSON editor

1. In the **Policy Document** field, paste the policy to apply to your tasks. The example below allows permission to the *`my-task-secrets-bucket`* Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-task-secrets-bucket/*"
      ]
    }
  ]
}
```

2. Choose **Create Policy**.

To create an IAM role for your tasks

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create New Role**.
3. In the **Select Role Type** section, for the **Amazon Elastic Container Service Task Role** service role, choose **Select**.

   Note
   To view the trust relationship for this role, see **Amazon ECS Task Role** (p. 165).
4. In the **Attach Policy** section, select the policy to use for your tasks (in this example `AmazonECSTaskS3BucketPolicy`, and then choose **Next Step**.
5. For **Role Name**, enter a name for your role. For this example, type `AmazonECSTaskS3BucketRole` to name the role, and then choose **Create Role** to finish.

Using a Supported AWS SDK

Support for IAM roles for tasks was added to the AWS SDKs on July 13th, 2016. The containers in your tasks must use an AWS SDK version that was created on or after that date. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at Tools for Amazon Web Services when you are building your containers to get the latest version.

Specifying an IAM Role for your Tasks

After you have created a role and attached a policy to that role, you can run tasks that assume the role. You have several options to do this:

- Specify an IAM role for your tasks in the task definition. You can create a new task definition or a new revision of an existing task definition and specify the role you created previously. If you use the console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter. For more information, see Creating a Task Definition (p. 26).

  Note
  This option is required if you want to use IAM task roles in an Amazon ECS service.
- Specify an IAM task role override when running a task. You can specify an IAM task role override when running a task. If you use the console to run your task, choose **Advanced Options** and then choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter in the `overrides` JSON object. For more information, see Running Tasks (p. 68).

Note
In addition to the standard Amazon ECS permissions required to run tasks and services, IAM users also require `iam:PassRole` permissions to use IAM roles for tasks.

Amazon ECS IAM Policy Examples

The following examples show policy statements that you could use to control the permissions that IAM users have for Amazon ECS.

Topics

- Amazon ECS First Run Wizard (p. 171)
- Clusters (p. 174)
- List and Describe Tasks (p. 176)
- Create Services (p. 176)
- Update Services (p. 177)

Amazon ECS First Run Wizard

The Amazon ECS first-run wizard simplifies the process of creating a cluster and running your tasks and services. However, users require permissions to many API operations from multiple AWS services to complete the wizard. The AmazonECS_FullAccess (p. 150) managed policy below shows the required permissions to complete the Amazon ECS first-run wizard.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
```

```
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:Describe*",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:PutMetricAlarm",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CancelSpotFleetRequests",
                "ec2:CreateInternetGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
                "ec2:DeleteSubnet",
                "ec2:DeleteVpc",
                "ec2:Describe*",
                "ec2:DetachInternetGateway",
                "ec2:DisassociateRouteTable",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:RequestSpotFleet",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateRule",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteRule",
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:DescribeTargetGroups",
                "ecs:*",
                "events:DescribeRule",
                "events:DeleteRule",
                "events:ListRuleNamesByTarget",
                "events:ListTargetsByRule",
                "events:PutRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfiles",
                "iam:ListRoles",
                "logs:CreateLogGroup",
                "logs:DescribeLogGroups",
                "logs:FilterLogEvents",
                "route53:GetHostedZone",
                "route53:ListHostedZonesByName",
                "route53:CreateHostedZone",
                "route53:DeleteHostedZone",
                "route53:GetHealthCheck",
                "servicediscovery:CreatePrivateDnsNamespace",
```

```
                "servicediscovery:CreateService",
                "servicediscovery:GetNamespace",
                "servicediscovery:GetOperation",
                "servicediscovery:GetService",
                "servicediscovery:ListNamespaces",
                "servicediscovery:ListServices",
                "servicediscovery:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParametersByPath",
                "ssm:GetParameters",
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/aws/service/ecs*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DeleteInternetGateway",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "ec2:ResourceTag/aws:cloudformation:stack-name": "EC2ContainerService-
*"
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsInstanceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com",
                        "ec2.amazonaws.com.cn"
                    ]
                }
            }
        },
```

```
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsAutoscaleRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "application-autoscaling.amazonaws.com",
                        "application-autoscaling.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "ecs.amazonaws.com",
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "ecs.application-autoscaling.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

The first run wizard also attempts to automatically create different IAM roles depending on the launch type of the tasks used. Examples are the Amazon ECS service role, container instance IAM role, and the task execution IAM role. To ensure that the first-run experience is able to create these IAM roles, one of the following must be true:

- Your user has administrator access. For more information, see Setting Up with Amazon ECS (p. 7).
- Your user has the IAM permissions to create a service role. For more information, see Creating a Role to Delegate Permissions to an AWS Service.
- You have a user with administrator access manually create the required IAM role so it is available on the account to be used. For more information, see the following:
  - Amazon ECS Service Scheduler IAM Role (p. 162)
  - Amazon ECS Task Execution IAM Role (p. 156)

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"
        }
      }
    }
  ]
}
```

List and Describe Tasks

The following IAM policy allows a user to list tasks for a specified cluster:

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

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

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

Create Services

The following IAM policy allows a user to create Amazon ECS services in the AWS Management Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs:List*",
        "ecs:Describe*",
        "ecs:CreateService",
        "elasticloadbalancing:Describe*",
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:ListRoles",
        "iam:ListGroups",
        "iam:ListUsers"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Update Services

The following IAM policy allows a user to update Amazon ECS services in the AWS Management Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs:List*",
        "ecs:Describe*",
        "ecs:UpdateService",
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:ListRoles",
        "iam:ListGroups",
        "iam:ListUsers"
      ],
      "Resource": [
        "*"
```

```
        ]
      }
    ]
}
```

Using the Amazon ECS Command Line Interface

The Amazon Elastic Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose files (Version 1, Version 2, and Version 3), a popular open-source specification for defining and running multi-container applications. Use the CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

The latest version of the Amazon ECS CLI is 1.10.0. For release notes, see Changelog.

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

Topics

- Installing the Amazon ECS CLI (p. 179)
- Configuring the Amazon ECS CLI (p. 184)
- Migrating Configuration Files (p. 186)
- Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 187)
- Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 191)

Installing the Amazon ECS CLI

Follow these instructions to install the Amazon ECS CLI on your macOS, Linux, or Windows system.

Step 1: Download the Amazon ECS CLI

Download the Amazon ECS CLI binary.

- For macOS:

```
sudo curl -o /usr/local/bin/ecs-cli https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-
darwin-amd64-latest
```

- For Linux systems:

```
sudo curl -o /usr/local/bin/ecs-cli https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-
linux-amd64-latest
```

- For Windows systems:

  Open Windows PowerShell and run the following commands:

```
PS C:\> New-Item 'C:\Program Files\Amazon\ECSCLI' -type
 directory
```

```
PS C:\> Invoke-WebRequest -OutFile 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe' https://
s3.amazonaws.com/amazon-ecs-cli/ecs-cli-windows-amd64-latest.exe
```

Note
If you encounter permissions issues, ensure that you are running PowerShell as an administrator.

Step 2: (Optional) Verify the Amazon ECS CLI

To verify the validity of the Amazon ECS CLI file, you can either use the provided MD5 sum or the PGP signatures. Both methods are described in the following sections.

Verify Using the MD5 Sum

Verify the downloaded binary with the MD5 sum provided.

- For macOS (compare the two output strings to verify that they match):

```
curl -s https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-darwin-amd64-latest.md5 && md5 -
q /usr/local/bin/ecs-cli
```

- For Linux systems (look for an `OK` in the output string):

```
echo "$(curl -s https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-linux-amd64-latest.md5) /
usr/local/bin/ecs-cli" | md5sum -c -
```

- For Windows systems:

  Open Windows PowerShell and find the md5 hash of the executable that you downloaded:

```
PS C:\> Get-FileHash ecs-cli.exe -Algorithm MD5
```

  Compare that with this md5 hash:

```
PS C:\> Invoke-WebRequest -OutFile md5.txt https://
s3.amazonaws.com/amazon-ecs-cli/ecs-cli-windows-amd64-
latest.md5
PS C:\> Get-Content md5.txt
```

Verify Using the PGP Signature

The Amazon ECS CLI executables are cryptographically signed using PGP signatures. You can use the following steps to verify the signatures using the GnuPG tool.

1. Download and install GnuPG. For more information, see the GnuPG website.

   - For macOS, we recommend using Homebrew. Install Homebrew using the instructions from their website. For more information, see Homebrew. After Homebrew is installed, use the following command from your macOS terminal:

     ```
     brew install gnupg
     ```

   - For Linux systems, install `gpg` using the package manager on your flavor of Linux.

- For Windows systems, download and use the Windows simple installer from the GnuPG website. For more information, see GnuPG Download.

2. Retrieve the Amazon ECS PGP public key. You can use a command to do this or manually create the key and then import it.

   a. Option 1: Retrieve the key with the following command.

```
gpg --keyserver hkp://keys.gnupg.net --recv BCE9D9A42D51784F
```

   b. Option 2: Create a file with the following contents of the Amazon ECS PGP public key and then import it:


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
```

X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlW8qsA
EQEAAYkEPgQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4tOLRHFHXuKUcxgGaubUcVtsFrwBKma1cYjqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53MEl8zQFw9Y5AmRoW4PZXER
ujs5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAte8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbzcqWaE51tJvgUR/nZFo6Ta3O5Ezhs
3VlEJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd24lRA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua

```
KohVD/0R4QxiSwQmICNtm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6MlI2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMa+dlE6kHx0rS0dPiuJ4O7NtPeYDKkoQtNagspsDvh
cK7CSqAiKMq06UBTxqlTSRkm62eOCtcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj2O+X+J
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

The details of the Amazon ECS PGP public key for reference:

```
Key ID: BCE9D9A42D51784F
Type: RSA
Size: 4096/4096
Expires: Never
User ID: Amazon ECS
Key fingerprint: F34C 3DDA E729 26B0 79BE AEC6 BCE9 D9A4 2D51 784F
```

Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

3. Download the Amazon ECS CLI signatures. The signatures are ASCII detached PGP signatures stored in files with the extension `.asc`. The signatures file has the same name as its corresponding executable, with `.asc` appended.

   - For macOS systems:

     ```
     curl -o ecs-cli.asc https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-darwin-amd64-
     latest.asc
     ```

   - For Linux systems:

     ```
     curl -o ecs-cli.asc https://s3.amazonaws.com/amazon-ecs-cli/ecs-cli-linux-amd64-
     latest.asc
     ```

   - For Windows systems:

     ```
     PS C:\> Invoke-WebRequest -OutFile ecs-cli.asc https://s3.amazonaws.com/amazon-ecs-
     cli/ecs-cli-windows-amd64-latest.exe.asc
     ```

4. Verify the signature.

   - For macOS and Linux systems:

     ```
     gpg --verify ecs-cli.asc /usr/local/bin/ecs-cli
     ```

   - For Windows systems:

     ```
     PS C:\> gpg --verify ecs-cli.asc 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe'
     ```

   Expected output:

```
gpg: Signature made Tue Apr  3 13:29:30 2018 PDT
gpg:                using RSA key DE3CBD61ADAF8B8E
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: EB3D F841 E2C9 212A 2BD4  2232 DE3C BD61 ADAF 8B8E
```

Important
The warning in the output is expected and is not problematic. It occurs because there is not a chain of trust between your personal PGP key (if you have one) and the Amazon ECS PGP key. For more information, see Web of trust.

Step 3: Apply Execute Permissions to the Binary

Apply execute permissions to the binary.

- For macOS and Linux systems:

```
sudo chmod +x /usr/local/bin/ecs-cli
```

- For Windows systems:

  Edit the environment variables and add `C:\Program Files\Amazon\ECSCLI` to the `PATH` variable field, separated from existing entries by using a semicolon. For example:

```
PS C:\> C:\existing\path;C:\Program Files\Amazon\ECSCLI
```

  Restart PowerShell (or the command prompt) so the changes go into effect.

  Note
  Once the `PATH` variable is set, the Amazon ECS CLI can be used from either Windows PowerShell or the command prompt.

Step 4: Complete the Installation

Verify that the CLI is working properly.

```
ecs-cli --version
```

Proceed to Configuring the Amazon ECS CLI (p. 184).

Important
You must configure the Amazon ECS CLI with your AWS credentials, an AWS region, and an Amazon ECS cluster name before you can use it.

Configuring the Amazon ECS CLI

The Amazon ECS CLI requires some basic configuration information before you can use it, such as your AWS credentials, the AWS Region in which to create your cluster, and the name of the Amazon ECS cluster to use. Configuration information is stored in the `~/.ecs` directory on macOS and Linux systems and in `C:\Users\<username>\AppData\local\ecs` on Windows systems.

To configure the Amazon ECS CLI

1. Set up a CLI profile with the following command, substituting *profile_name* with your desired profile name, *$AWS_ACCESS_KEY_ID* and *$AWS_SECRET_ACCESS_KEY* environment variables with your AWS credentials.

```
ecs-cli configure profile --profile-name profile_name --access-key $AWS_ACCESS_KEY_ID
 --secret-key $AWS_SECRET_ACCESS_KEY
```

2. Complete the configuration with the following command, substituting *launch_type* with the task launch type you want to use by default, *region_name* with your desired AWS region, *cluster_name* with the name of an existing Amazon ECS cluster or a new cluster to use, and *configuration_name* for the name you'd like to give this configuration.

```
ecs-cli configure --cluster cluster_name --default-launch-type launch_type --
region region_name --config-name configuration_name
```

After you have installed and configured the CLI, you can try the Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 187). For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.

Profiles

The Amazon ECS CLI supports the configuring of multiple sets of AWS credentials as named *profiles* using the **ecs-cli configure profile** command. A default profile can be set by using the **ecs-cli configure profile default** command. These profiles can then be referenced when you run Amazon ECS CLI commands that require credentials using the `--ecs-profile` flag otherwise the default profile is used.

For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.

Cluster Configurations

A cluster configuration is a set of fields that describes an Amazon ECS cluster including the name of the cluster and the region. A default cluster configuration can be set by using the **ecs-cli configure default** command. The Amazon ECS CLI supports the configuring of multiple named cluster configurations using the `--config-name` option.

For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.

Order of Precedence

There are multiple methods for passing both the credentials and the region in an Amazon ECS CLI command. The following is the order of precedence for each of these.

The order of precedence for credentials is:

1. Amazon ECS CLI profile flags:
   a. ECS profile (`--ecs-profile`)
   b. AWS profile (`--aws-profile`)
2. Environment variables:
   a. `ECS_PROFILE`
   b. `AWS_PROFILE`
   c. `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`

3. ECS config-attempts to fetch credentials from the default ECS profile.
4. Default AWS profile—Attempts to use credentials (`aws_access_key_id`, `aws_secret_access_key`) or `assume_role` (`role_arn`, `source_profile`) from the AWS profile name.
   a. `AWS_DEFAULT_PROFILE` environment variable (defaults to `default`).
5. EC2 instance role

The order of precedence for Region is:

1. Amazon ECS CLI flags:
   a. Region flag (`--region`)
   b. Cluster config flag (`--cluster-config`)
2. ECS config-attempts to fetch the Region from the default ECS profile.
3. Environment variables—Attempts to fetch the region from the following environment variables:
   a. `AWS_REGION`
   b. `AWS_DEFAULT_REGION`
4. AWS profile-attempts to use the region from the AWS profile name:
   a. `AWS_PROFILE` environment variable
   b. `AWS_DEFAULT_PROFILE` environment variable (defaults to `default`)

Migrating Configuration Files

The process of configuring the Amazon ECS CLI has changed significantly in the latest version (v1.0.0) to allow the addition of new features. A migration command has been introduced that converts an older (v0.6.6 and older) configuration file to the current format. The old configuration files are deprecated, so we recommend converting your configuration to the newest format to take advantage of the new features. The configuration-related changes and new features introduced in v1.0.0 in the new YAML formatted configuration files include:

- Splitting up of credential and cluster-related configuration information into two separate files. Credential information is stored in `~/.ecs/credentials` and cluster configuration information is stored in `~/.ecs/config`.
- The configuration files are formatted in YAML.
- Support for storing multiple named configurations.
- Deprecation of the field `compose-service-name-prefix` (name used for creating a service `<compose_service_name_prefix> + <project_name>`). This field can still be configured. However, if it is not configured, there is no longer a default value assigned. For Amazon ECS CLI v0.6.6 and earlier, the default was `ecscompose-service-`.
- Removal of the field `compose-project-name-prefix` (name used for creating a task definition `<compose_project_name_prefix> + <project_name>`). Amazon ECS CLI v1.0.0 and later can still read old configuration files; so if this field is present then it is still read and used. However, configuring this field is not supported in v1.0.0+ with the `ecs-cli configure` command, and if the field is manually added to a v1.0.0+ configuration file it causes the Amazon ECS CLI to throw an error.
- The field `cfn-stack-name-prefix` (name used for creating CFN stacks `<cfn_stack_name_prefix> + <cluster_name>`) has been changed to `cfn-stack-name`. Instead of specifying a prefix, the exact name of a CloudFormation template can be configured.
- Amazon ECS CLI v0.6.6 and earlier allowed configuring credentials using a named AWS profile from the `~/.aws/credentials` file on your system. This functionality has been removed. However, a new flag, `--aws-profile`, has been added which allows the referencing of an AWS profile inline in all commands that require credentials.

Note
The `--project-name` flag can be used to set the project name.

Migrating Older Configuration Files to the v1.0.0+ Format

While all versions of the Amazon ECS CLI support reading from the older configuration file format, upgrading to the new format is required to take advantage of some new features, for example using multiple named cluster profiles. Migrating your legacy configuration file to the new format is easy with the `ecs-cli configure migrate` command. The command takes the configuration information stored in the old format in `~/.ecs/config` and converts it to a pair of files in the new format, overwriting your old configuration file in the process.

When running the `ecs-cli configure migrate` command there is a warning message displayed with the old configuration file, and a preview of the new configuration files. User confirmation is required before the migration proceeds. If the `--force` flag is used, then the warning message is not displayed, and the migration proceeds without any confirmation. If `cfn-stack-name-prefix` is used in the legacy file, then `cfn-stack-name` is stored in the new file as `<cfn_stack_name_prefix> + <cluster_name>`.

For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.

Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI

This tutorial shows you how to set up a cluster and deploy a task using the Fargate launch type.

Prerequisites

Complete the following prerequisites:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 179).
- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.

Step 1: Create the Task Execution IAM Role

Amazon ECS needs permissions so that your Fargate task can store logs in CloudWatch. These permissions are covered by the task execution IAM role. For more information, see Amazon ECS Task Execution IAM Role (p. 156).

To create the task execution IAM role using the AWS CLI

1. Create a file named `task-execution-assume-role.json` with the following contents:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create the task execution role:

```
aws iam --region us-east-1 create-role --role-name ecsTaskExecutionRole --assume-role-
policy-document file://task-execution-assume-role.json
```

3. Attach the task execution role policy:

```
aws iam --region us-east-1 attach-role-policy --role-name ecsTaskExecutionRole --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

Step 2: Configure the Amazon ECS CLI

The Amazon ECS CLI requires credentials in order to make API requests on your behalf. It can pull credentials from environment variables, an AWS profile, or an Amazon ECS profile. For more information, see Configuring the Amazon ECS CLI (p. 184).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration, which defines the AWS region to use, resource creation prefixes, and the cluster name to use with the Amazon ECS CLI:

```
ecs-cli configure --cluster tutorial --region us-east-1 --default-launch-type FARGATE
 --config-name tutorial
```

2. Create a CLI profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-
key AWS_SECRET_ACCESS_KEY --profile-name tutorial
```

> **Note**
> If this is the first time that you are configuring the Amazon ECS CLI, these configurations are marked as default. If this is not your first time configuring the Amazon ECS CLI, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide* to set this as the default configuration and profile.

Step 3: Create a Cluster and Security Group

To create an ECS cluster and security group

1. Create an Amazon ECS cluster with the **ecs-cli up** command. Because you specified Fargate as your default launch type in the cluster configuration, this command creates an empty cluster and a VPC configured with two public subnets.

```
ecs-cli up
```

Note
This command may take a few minutes to complete as your resources are created. Take note of the VPC and subnet IDs that are created as they are used later.

2. Using the AWS CLI, create a security group using the VPC ID from the previous output:

```
aws ec2 create-security-group --group-name "my-sg" --description "My security group" --
vpc-id "VPC_ID"
```

3. Using AWS CLI, add a security group rule to allow inbound access on port 80:

```
aws ec2 authorize-security-group-ingress --group-id "security_group_id" --protocol tcp
 --port 80 --cidr 0.0.0.0/0
```

Step 4: Create a Compose File

For this step, create a simple Docker compose file that creates a WordPress application. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1, 2, and 3. This tutorial uses Docker compose v3.

Here is the compose file, which you can name `docker-compose.yml`. The `wordpress` container exposes port 80 for inbound traffic to the web server. It also configures container logs to go to the CloudWatch log group created earlier. This is the recommended best practice for Fargate tasks.

```
version: '3'
services:
  wordpress:
    image: wordpress
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: wordpress
```

In addition to the Docker compose information, there are some parameters specific to Amazon ECS that you must specify for the service. Using the VPC, subnet, and security group IDs from the previous step, create a file named `ecs-params.yml` with the following content:

```
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - "subnet ID 1"
        - "subnet ID 2"
      security_groups:
        - "security group ID"
      assign_public_ip: ENABLED
```

Step 5: Deploy the Compose File to a Cluster

After you create the compose file, you can deploy it to your cluster with **ecs-cli compose service up**. By default, the command looks for files called `docker-compose.yml` and `ecs-params.yml` in the current directory; you can specify a different docker compose file with the `--file` option, and a different ECS Params file with the `--ecs-params` option. By default, the resources created by this command have the current directory in their titles, but you can override that with the `--project-name` option. The `--create-log-groups` option creates the CloudWatch log groups for the container logs.

```
ecs-cli compose --project-name tutorial service up --create-log-groups --cluster-
config tutorial
```

Step 6: View the Running Containers on a Cluster

After you deploy the compose file, you can view the containers that are running in the service with **ecs-cli compose service ps**.

```
ecs-cli compose --project-name tutorial service ps --cluster-config tutorial
```

Output:

```
WARN[0000] Skipping unsupported YAML option...           option name=networks
WARN[0000] Skipping unsupported YAML option for service...  option name=networks service
 name=wordpress
Name                                          State    Ports TaskDefinition
a06a6642-12c5-4006-b1d1-033994580605/wordpress  RUNNING  54.146.193.73:80->80/tcp
 tutorial:9
```

In the above example, you can see the `wordpress` container from your compose file, and also the IP address and port of the web server. If you point your web browser at that address, you should see the WordPress installation wizard. Also in the output is the `task-id` value for the container. Copy the task ID as you use it in the next step.

Step 7: View the Container Logs

View the logs for the task:

```
ecs-cli logs --task-id a06a6642-12c5-4006-b1d1-033994580605 --follow --cluster-
config tutorial
```

> **Note**
> The `--follow` option tells the Amazon ECS CLI to continuously poll for logs.

Step 8: Scale the Tasks on the Cluster

You can scale up your task count to increase the number of instances of your application with **ecs-cli compose service scale**. In this example, the running count of the application is increased to two.

```
ecs-cli compose --project-name tutorial service scale 2 --cluster-config tutorial
```

Now you should see two more containers in your cluster:

```
ecs-cli compose --project-name tutorial service ps --cluster-config tutorial
```

Output:

```
WARN[0000] Skipping unsupported YAML option...           option name=networks
WARN[0000] Skipping unsupported YAML option for service...  option name=networks service
 name=wordpress
Name                                          State    Ports        TaskDefinition
880f09ed-613d-44bf-99bb-42ca44f82904/wordpress  RUNNING  34.224.60.24:80->80/tcp
 tutorial:9
a06a6642-12c5-4006-b1d1-033994580/wordpress  RUNNING  54.146.193.73:80->80/tcp  tutorial:9
```

Step 9: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

```
ecs-cli compose --project-name tutorial service down --cluster-config tutorial
```

Now, take down your cluster, which cleans up the resources that you created earlier with ecs-cli up.

```
ecs-cli down --force --cluster-config tutorial
```

Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI

This tutorial shows a simple walkthrough of creating an Amazon ECS service that is configured to use service discovery. Many of the service discovery configuration values can be specified with either the ECS parameters file or flags. When flags are used, they take precedence over the ECS parameters file if both are present. When using the Amazon ECS CLI, the compose project name is used as the name for your ECS service.

Prerequisites

It is expected that you have completed the following prerequisites before continuing on:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 179).

Configure the Amazon ECS CLI

Before you can start this tutorial, you must install and configure the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 179).

The Amazon ECS CLI requires credentials in order to make API requests on your behalf. It can pull credentials from environment variables, an AWS profile, or an Amazon ECS profile. For more information, see Configuring the Amazon ECS CLI (p. 184).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

```
ecs-cli configure --cluster ec2-tutorial --region us-east-1 --default-launch-type EC2
 --config-name ec2-tutorial
```

2. Create a profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-
key AWS_SECRET_ACCESS_KEY --profile-name ec2-tutorial
```

> **Note**
> If this is the first time that you are configuring the Amazon ECS CLI, these configurations are marked as default. If this is not your first time configuring the Amazon ECS CLI, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide* to set this as the default configuration and profile.

Create an Amazon ECS Service Configured to Use Service Discovery

Use the following steps to create an Amazon ECS service that is configured to use service discovery with the Amazon ECS CLI.

To create an Amazon ECS service configured to use service discovery

1. Create an Amazon ECS service named `backend` and create a private DNS namespace named `tutorial` within a VPC. In this example, the task is using the `awsvpc` network mode, so the `container_name` and `container_port` values are not required.

```
ecs-cli compose --project-name backend service up --private-dns-namespace tutorial --
vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

Output:

```
INFO[0001] Using ECS task definition                     TaskDefinition="backend:1"
INFO[0002] Waiting for the private DNS namespace to be created...
INFO[0002] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
INFO[0033] Waiting for the Service Discovery Service to be created...
INFO[0034] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0065] Created an ECS service                        service=backend
 taskDefinition="backend:1"
INFO[0066] Updated ECS service successfully              desiredCount=1
 serviceName=backend
INFO[0081] (service backend) has started 1 tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e).  timestamp="2018-09-12 00:00:26 +0000 UTC"
INFO[0157] Service status                                desiredCount=1 runningCount=1
 serviceName=backend
INFO[0157] ECS Service has reached a stable state        desiredCount=1 runningCount=1
 serviceName=backend
```

2. Create another service named `frontend` in the same private DNS namespace. Because the namespace already exists, the Amazon ECS CLI uses it instead of creating a new one.

```
ecs-cli compose --project-name frontend service up --private-dns-namespace tutorial --
vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

Output:

```
INFO[0001] Using ECS task definition                     TaskDefinition="frontend:1"
INFO[0002] Using existing namespace ns-kvhnzhb5vxplfmls
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
INFO[0033] Waiting for the Service Discovery Service to be created...
INFO[0034] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0065] Created an ECS service                        service=frontend
 taskDefinition="frontend:1"
INFO[0066] Updated ECS service successfully              desiredCount=1
 serviceName=frontend
INFO[0081] (service frontend) has started 1 tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e).  timestamp="2018-09-12 00:00:26 +0000 UTC"
INFO[0157] Service status                                desiredCount=1 runningCount=1
 serviceName=frontend
INFO[0157] ECS Service has reached a stable state        desiredCount=1 runningCount=1
 serviceName=frontend
```

3. Verify that the two services are able to discover each other within the VPC using DNS. The DNS hostname uses the following format: `<service_discovery_service_name>.<service_discovery_namespace>`. For this example, the `frontend` service can be discovered at `frontend.tutorial` and the `backend` service can be discovered at `backend.tutorial`. Because these are private DNS namespaces, these DNS names only resolve when within the specified VPC.
4. To update the service discovery settings, update the settings for the `frontend` service. The values that can be updated are the DNS TTL and the value for the health check custom config failure threshold.

```
ecs-cli compose --project-name frontend service up --update-service-discovery --dns-
type SRV --dns-ttl 120 --healthcheck-custom-config-failure-threshold 2
```

Output:

```
INFO[0001] Using ECS task definition                     TaskDefinition="frontend:1"
INFO[0001] Updated ECS service successfully              desiredCount=1
 serviceName=frontend
INFO[0001] Service status                                desiredCount=1 runningCount=1
 serviceName=frontend
INFO[0001] ECS Service has reached a stable state        desiredCount=1 runningCount=1
 serviceName=frontend
INFO[0002] Waiting for your Service Discovery resources to be updated...
INFO[0002] Cloudformation stack status                   stackStatus=UPDATE_IN_PROGRESS
```

5. To clean up, delete the Amazon ECS service and the service discovery resources. When the `frontend` service is deleted, the Amazon ECS CLI automatically removes the associated service discovery service.

```
ecs-cli compose --project-name frontend service rm
```

```
INFO[0000] Updated ECS service successfully              desiredCount=0
 serviceName=frontend
INFO[0001] Service status                                desiredCount=0 runningCount=1
 serviceName=frontend
INFO[0016] Service status                                desiredCount=0 runningCount=0
 serviceName=frontend
INFO[0016] (service frontend) has stopped 1 running tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e).  timestamp="2018-09-12 00:37:25 +0000 UTC"
INFO[0016] ECS Service has reached a stable state        desiredCount=0 runningCount=0
 serviceName=frontend
INFO[0016] Deleted ECS service                           service=frontend
```

```
INFO[0016] ECS Service has reached a stable state        desiredCount=0 runningCount=0
 serviceName=frontend
INFO[0027] Waiting for your Service Discovery Service resource to be deleted...
INFO[0027] Cloudformation stack status                   stackStatus=DELETE_IN_PROGRESS
```

6. To complete the cleanup, delete the `backend` service along with the private DNS namespace that was created with it. The Amazon ECS CLI associates the AWS CloudFormation stack for the private DNS namespace with the Amazon ECS service for which it was created. When the service is deleted, the namespace is also deleted.

```
ecs-cli compose --project-name backend service rm --delete-namespace
```

Using the AWS CLI with Amazon ECS

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/.

The following steps help you set up an Amazon ECS cluster using a Fargate task:

Topics

- Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI (p. 195)

Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI

The following steps 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. Ensure that you are using the latest version of the AWS CLI. For more information on how to upgrade to the latest version, see Installing the AWS Command Line Interface.

Prerequisites (p. 195)

Step 1: (Optional) Create a Cluster (p. 195)

Step 2: Register a Task Definition (p. 196)

Step 3: List Task Definitions (p. 198)

Step 4: Create a Service (p. 198)

Step 5: List Services (p. 199)

Step 6: Describe the Running Service (p. 200)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see Installing the AWS Command Line Interface.
- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 171) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: (Optional) Create a Cluster

By default, your account receives a `default` cluster.

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 subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` *`cluster_name`* for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
    "cluster": {
        "status": "ACTIVE",
        "statistics": [],
        "clusterName": "fargate-cluster",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

Step 2: 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 creates a PHP web app. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 22).

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "httpd:2.4",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' >  /usr/local/apache2/htdocs/index.html && httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
```

```
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}
```

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/fargate-task.json
```

The **register-task-definition** returns a description of the task definition after it completes its registration.

```
{
    "taskDefinition": {
        "status": "ACTIVE",
        "networkMode": "awsvpc",
        "family": "sample-fargate",
        "placementConstraints": [],
        "requiresAttributes": [
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
            },
            {
                "name": "ecs.capability.task-eni"
            }
        ],
        "cpu": "256",
        "compatibilities": [
            "EC2",
            "FARGATE"
        ],
        "volumes": [],
        "memory": "512",
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "taskDefinitionArn": "arn:aws:ecs:region:aws_account_id:task-definition/sample-
fargate:2",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "fargate-app",
                "mountPoints": [],
                "image": "httpd:2.4",
                "cpu": 0,
                "portMappings": [
                    {
                        "protocol": "tcp",
                        "containerPort": 80,
                        "hostPort": 80
                    }
                ],
                "entryPoint": [
                    "sh",
```

```
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' >  /usr/local/apache2/htdocs/index.html && httpd-foreground\""
                ],
                "essential": true,
                "volumesFrom": []
            }
        ],
        "revision": 2
    }
}
```

Step 3: 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
```

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1",
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:2"
    ]
}
```

Step 4: Create a Service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service where at least two instances of the `sample-fargate:1` task definition are kept running in your cluster.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 2 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}"
```

Output:

```
{
    "service": {
        "status": "ACTIVE",
        "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-
fargate:1",
        "pendingCount": 0,
        "launchType": "FARGATE",
        "loadBalancers": [],
        "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS",
        "placementConstraints": [],
```

```
        "createdAt": 1510811361.128,
        "desiredCount": 2,
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-abcd1234"
                ],
                "securityGroups": [
                    "sg-abcd1234"
                ],
                "assignPublicIp": "DISABLED"
            }
        },
        "platformVersion": "LATEST",
        "serviceName": "fargate-service",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
        "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "deployments": [
            {
                "status": "PRIMARY",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "subnets": [
                            "subnet-abcd1234"
                        ],
                        "securityGroups": [
                            "sg-abcd1234"
                        ],
                        "assignPublicIp": "DISABLED"
                    }
                },
                "pendingCount": 0,
                "launchType": "FARGATE",
                "createdAt": 1510811361.128,
                "desiredCount": 2,
                "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1",
                "updatedAt": 1510811361.128,
                "platformVersion": "0.0.1",
                "id": "ecs-svc/9223370526043414679",
                "runningCount": 0
            }
        ],
        "events": [],
        "runningCount": 0,
        "placementStrategy": []
    }
}
```

Step 5: List Services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-service"
    ]
}
```

Step 6: Describe the Running Service

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

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

Output:

```
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-
fargate:1",
            "pendingCount": 2,
            "launchType": "FARGATE",
            "loadBalancers": [],
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "placementConstraints": [],
            "createdAt": 1510811361.128,
            "desiredCount": 2,
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "DISABLED"
                }
            },
            "platformVersion": "LATEST",
            "serviceName": "fargate-service",
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "DISABLED"
                        }
                    },
                    "pendingCount": 2,
```

```
                    "launchType": "FARGATE",
                    "createdAt": 1510811361.128,
                    "desiredCount": 2,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1",
                    "updatedAt": 1510811361.128,
                    "platformVersion": "0.0.1",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "placementStrategy": []
        }
    ],
    "failures": []
}
```

Amazon ECS Task Metadata Endpoint

Amazon ECS provides a method to retrieve various task metadata and Docker stats. This is referred to as the task metadata endpoint. The task metadata endpoint is available for tasks that use the Fargate launch type on platform version v1.1.0 or later.

All containers belonging to tasks that are launched with the `awsvpc` network mode receive a local IPv4 address within a predefined link-local address range. When a container queries the metadata endpoint, the Amazon ECS container agent can determine which task the container belongs to based on its unique IP address, and metadata and stats for that task are returned.

For information about a sample Go application that queries the metadata and stats API endpoints, see https://github.com/aws/amazon-ecs-agent/blob/2bf4348a0ff89e23be4e82a6c5ff28edf777092c/misc/taskmetadata-validator/taskmetadata-validator.go.

Enabling Task Metadata

The task metadata endpoint feature is enabled by default for tasks using the Fargate launch type that use platform version v1.1.0 or later. For more information, see AWS Fargate Platform Versions (p. 19).

Task Metadata Endpoint Paths

The following API endpoints are available to containers:

`169.254.170.2/v2/metadata`

This endpoint returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Task Metadata JSON Response (p. 202).

`169.254.170.2/v2/metadata/<container-id>`

This endpoint returns metadata JSON for the specified Docker container ID.

`169.254.170.2/v2/stats`

This endpoint returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

`169.254.170.2/v2/stats/<container-id>`

This endpoint returns Docker stats JSON for the specified Docker container ID. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

Task Metadata JSON Response

The following information is returned from the task metadata endpoint (`169.254.170.2/v2/metadata`) JSON response.

`Cluster`

The Amazon ECS cluster to which the task belongs.

`TaskARN`

The full Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest for the image.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level (such as CPU and memory). This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`Limits`

The resource limits specified at the task level (such as CPU and memory). This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The time stamp for when the first container image pull began.

`PullStoppedAt`

The time stamp for when the last container image pull finished.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Example Task Metadata Response

The following JSON response is for a single-container task.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
```

```
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
west-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
west-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z"
}
```

Tutorial: Continuous Deployment with AWS CodePipeline

This tutorial helps you to create a complete, end-to-end continuous deployment (CD) pipeline with Amazon ECS with AWS CodePipeline.

Prerequisites

There are a few resources that you must have in place before you can use this tutorial to create your CD pipeline. Here are the things you need to get started:

> **Note**
> All of these resources should be created within the same AWS Region.

- A source control repository (this tutorial uses AWS CodeCommit) with your Dockerfile and application source. For more information, see Create an AWS CodeCommit Repository in the *AWS CodeCommit User Guide*.
- A Docker image repository (this tutorial uses Amazon ECR) that contains an image you have built from your Dockerfile and application source. For more information, see Creating a Repository and Pushing an Image in the *Amazon Elastic Container Registry User Guide*.
- An Amazon ECS task definition that references the Docker image hosted in your image repository. For more information, see Creating a Task Definition in the *Amazon Elastic Container Service Developer Guide*.
- An Amazon ECS cluster that is running a service that uses your previously mentioned task definition. For more information, see Creating a Cluster and Creating a Service in the *Amazon Elastic Container Service Developer Guide*.

After you have satisfied these prerequisites, you can proceed with the tutorial and create your CD pipeline.

Step 1: Add a Build Specification File to Your Source Repository

This tutorial uses AWS CodeBuild to build your Docker image and push the image to Amazon ECR. Add a `buildspec.yml` file to your source code repository to tells AWS CodeBuild how to do that. The example build specification below does the following:

- Pre-build stage:
  - Log in to Amazon ECR.
  - Set the repository URI to your ECR image and add an image tag with the first seven characters of the Git commit ID of the source.
- Build stage:
  - Build the Docker image and tag the image both as `latest` and with the Git commit ID.
- Post-build stage:

- Push the image to your ECR repository with both tags.
- Write a file called `imagedefinitions.json` in the build root that has your Amazon ECS service's container name and the image and tag. The deployment stage of your CD pipeline uses this information to create a new revision of your service's task definition, and then it updates the service to use the new task definition. The `imagedefinitions.json` file is required for the AWS CodeDeploy ECS job worker.

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"hello-world","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG >
 imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

The build specification was written for the following task definition, used by the Amazon ECS service for this tutorial. The `REPOSITORY_URI` value corresponds to the `image` repository (without any image tag), and the *`hello-world`* value near the end of the file corresponds to the container name in the service's task definition.

```
{
  "taskDefinition": {
    "family": "hello-world",
    "containerDefinitions": [
      {
        "name": "hello-world",
        "image": "012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world:6a57b99",
        "cpu": 100,
        "portMappings": [
          {
            "protocol": "tcp",
            "containerPort": 80,
            "hostPort": 80
          }
        ],
        "memory": 128,
        "essential": true
      }
    ]
  }
```

```
}
```

To add a `buildspec.yml` file to your source repository

1. Open a text editor and then copy and paste the build specification above into a new file.
2. Replace the `REPOSITORY_URI` value (*`012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world`*) with your Amazon ECR repository URI (without any image tag) for your Docker image. Replace *`hello-world`* with the container name in your service's task definition that references your Docker image.
3. Commit and push your `buildspec.yml` file to your source repository.

   a. Add the file.

   ```
   git add .
   ```

   b. Commit the change.

   ```
   git commit -m "Adding build specification."
   ```

   c. Push the commit.

   ```
   git push
   ```

Step 2: Creating Your Continuous Deployment Pipeline

Use the AWS CodePipeline wizard to create your pipeline stages and connect your source repository to your ECS service.

To create your pipeline

1. Open the AWS CodePipeline console at https://console.aws.amazon.com/codepipeline/.
2. On the **Welcome** page, choose **Create pipeline**.

   If this is your first time using AWS CodePipeline, an introductory page appears instead of **Welcome**. Choose **Get Started Now**.
3. On the **Step 1: Name** page, for **Pipeline name**, type the name for your pipeline and choose **Next step**. For this tutorial, the pipeline name is **hello-world**.
4. On the **Step 2: Source** page, for **Source provider**, choose **AWS CodeCommit**.

   a. For **Repository name**, choose the name of the AWS CodeCommit repository to use as the source location for your pipeline.

   b. For **Branch name**, choose the branch to use and choose **Next step**.
5. On the **Step 3: Build** page, choose **AWS CodeBuild**, and then choose **Create a new build project**.

   a. For **Project name**, choose a unique name for your build project. For this tutorial, the project name is **hello-world**.

   b. For **Operating system**, choose **Ubuntu**.

   c. For **Runtime**, choose **Docker**.

   d. For **Version**, choose **aws/codebuild/docker:17.09.0** .

   e. Choose **Save build project**.

f. Choose **Next step**.

> **Note**
> The wizard creates an AWS CodeBuild service role for your build project, called **code-build-*build-project-name*-service-role**. Note this role name, as you add Amazon ECR permissions to it later.

6. On the **Step 4: Deploy** page, for **Deployment provider**, choose **Amazon ECS**.

   a. For **Cluster name**, choose the Amazon ECS cluster in which your service is running. For this tutorial, the cluster is **default**.

   b. For **Service name**, choose the service to update and choose **Next step**. For this tutorial, the service name is **hello-world**.

7. On the **Step 5: Service Role** page, choose **Create role**. On the IAM console page that describes the role to be created for you, choose **Allow**.
8. Choose **Next step**.
9. On the **Step 6: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.

   > **Note**
   > Now that the pipeline has been created, it attempts to run through the different pipeline stages. However, the default AWS CodeBuild role created by the wizard does not have permissions to execute all of the commands contained in the `buildspec.yml` file, so the build stage fails. The next section adds the permissions for the build stage.

Step 3: Add Amazon ECR Permissions to the AWS CodeBuild Role

The AWS CodePipeline wizard created an IAM role for the AWS CodeBuild build project, called **code-build-*build-project-name*-service-role**. For this tutorial, the name is **code-build-hello-world-service-role**. Because the `buildspec.yml` file makes calls to Amazon ECR API operations, the role must have a policy that allows permissions to make these Amazon ECR calls. The following procedure helps you attach the proper permissions to the role.

To add Amazon ECR permissions to the AWS CodeBuild role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. In the search box, type **code-build-** and choose the role that was created by the AWS CodePipeline wizard. For this tutorial, the role name is **code-build-hello-world-service-role**.
4. On the **Summary** page, choose **Attach policy**.
5. Select the box to the left of the **AmazonEC2ContainerRegistryPowerUser** policy, and choose **Attach policy**.

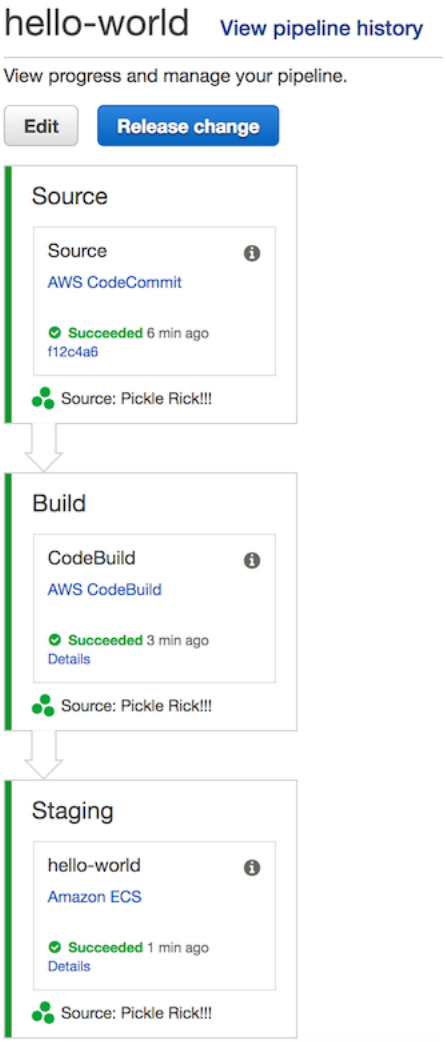
Step 4: Test Your Pipeline

Your pipeline should have everything for running an end-to-end native AWS continuous deployment. Now, test its functionality by pushing a code change to your source repository.

To test your pipeline

1. Make a code change to your configured source repository, commit, and push the change.

2. Open the AWS CodePipeline console at https://console.aws.amazon.com/codepipeline/.
3. Choose your pipeline from the list.
4. Watch the pipeline progress through its stages. Your pipeline should complete and your Amazon ECS service runs the Docker image that was created from your code change.



Amazon ECS Service Limits

The following table provides the default limits for Amazon ECS for an AWS account which can be changed. For more information on the service limits for other AWS services that you can use with Amazon ECS, such as Elastic Load Balancing and Auto Scaling, see AWS Service Limits in the *Amazon Web Services General Reference*.

Resource	Default Limit
Number of clusters per region, per account	1000
Number of container instances per cluster	1000
Number of services per cluster	500
Number of tasks using the EC2 launch type per service (the desired count)	1000
Number of tasks using the Fargate launch type, per region, per account	50
Number of public IP addresses for tasks using the Fargate launch type	50

The following table provides other limitations for Amazon ECS that cannot be changed.

Resource	Limit
Number of load balancers per service	1
Number of tasks launched (`count`) per **run-task**	10
Number of container instances per **start-task**	10
Throttle on container instance registration rate	1 per second / 60 max per minute
Task definition size limit	32 KiB
Task definition max containers	10
Throttle on task definition registration rate	1 per second / 60 max per minute
Number of subnets specified in `awsvpcConfiguration`	16
Number of security groups specified in `awsvpcConfiguration`	5
Maximum layer size of an image used by a task using the Fargate launch type	4 GB
Maximum size of a shared volume used by multiple containers within a task using the Fargate launch type	4 GB

Resource	Limit
Maximum container storage for tasks using the Fargate launch type	10 GB
Maximum number of tags per resource (tasks, services, task definitions, clusters, and container instances)	50

Logging Amazon ECS API Calls with AWS CloudTrail

Amazon ECS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. CloudTrail captures all API calls for Amazon ECS as events, including calls from the Amazon ECS console and from code calls to the Amazon ECS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon ECS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see the AWS CloudTrail User Guide.

Amazon ECS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon ECS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon ECS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon ECS actions are logged by CloudTrail and are documented in the Amazon Elastic Container Service API Reference. For example, calls to the `CreateService`, `RunTask` and `DeleteCluster` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon ECS Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

> **Note**
> These examples have been formatted for improved readability. In a CloudTrail log file, all entries and events are concatenated into a single line. In addition, this example has been limited to a single Amazon ECS entry. In a real CloudTrail log file, you see entries and events from multiple AWS services.

The following example shows a CloudTrail log entry that demonstrates the `CreateCluster` action:

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-06-20T18:32:25Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Mary_Major"
            }
        }
    },
    "eventTime": "2018-06-20T19:04:36Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "CreateCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "clusterName": "default"
    },
    "responseElements": {
        "cluster": {
            "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/default",
            "pendingTasksCount": 0,
            "registeredContainerInstancesCount": 0,
            "status": "ACTIVE",
            "runningTasksCount": 0,
            "statistics": [],
            "clusterName": "default",
            "activeServicesCount": 0
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
```

```
}
```

Amazon ECS Troubleshooting

You may need to troubleshoot issues with your load balancers, tasks, services, or container instances. This chapter helps you find diagnostic information from the Amazon ECS container agent, the Docker daemon on the container instance, and the service event log in the Amazon ECS console.

Topics

- Troubleshooting First-Run Wizard Launch Issues (p. 216)
- Checking Stopped Tasks for Errors (p. 216)
- Service Event Messages (p. 218)
- Invalid CPU or Memory Value Specified (p. 220)
- Cannot Pull Container Image Error (p. 221)
- Troubleshooting Service Load Balancers (p. 222)
- Troubleshooting IAM Roles for Tasks (p. 223)

Troubleshooting First-Run Wizard Launch Issues

The following error can prevent the Amazon ECS first-run wizard from creating your cluster.

VpcLimitExceeded

If you get a `VpcLimitExceeded` error when attempting to complete the Amazon ECS first-run wizard, you have reached the limit on the number of VPCs that you can create in a Region. When you create your AWS account, there are default limits on the number of VPCs you can run in each Region. For more information, see Amazon VPC Limits.

To resolve this issue, you have the following options:

- Request a VPC service limit increase on a per-Region basis. For more information, see Amazon VPC Limits.
- Delete any unused VPCs on your account. For more information, see Working with VPCs and Subnets.

Important
Any Amazon ECS resources that were successfully created during the first-run wizard before receiving this error can be deleted before running the wizard again.

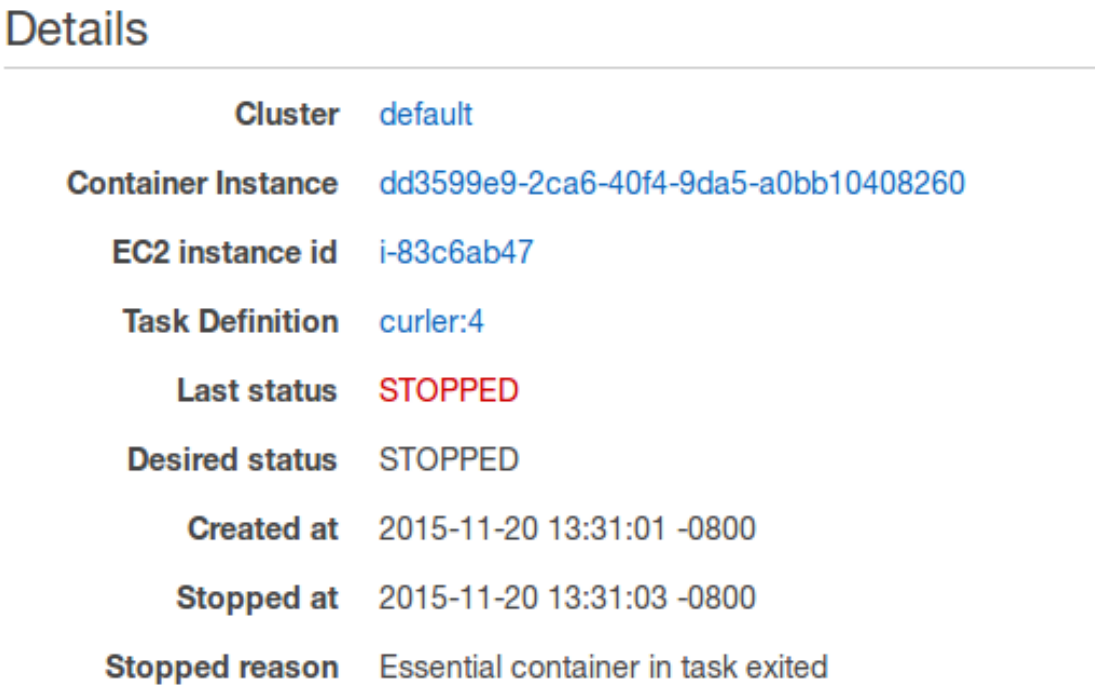
Checking Stopped Tasks for Errors

If you have trouble starting a task, your task might be stopping because of an error. For example, you run the task and the task displays a `PENDING` status and then disappears. 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, select the cluster in which your stopped task resides.

3. On the **Cluster : *clustername*** page, choose **Tasks**.
4. In the **Desired task status** table header, choose **Stopped**, and then select the stopped task to inspect. The most recent stopped tasks are listed first.
5. In the **Details** section, inspect the **Stopped reason** field to see the reason that the task was stopped.



Some possible reasons and their explanations are listed below:

Task failed ELB health checks in (elb elb-name)

The current task failed the Elastic Load Balancing health check for the load balancer that is associated with the task's service. For more information, see Troubleshooting Service Load Balancers (p. 222).

Scaling activity initiated by (deployment deployment-id)

When you reduce the desired count of a stable service, some tasks need to be stopped in order to reach the desired number. Tasks that are stopped by downscaling services have this stopped reason.

Host EC2 (instance *id*) stopped/terminated

If you stop or terminate a container instance with running tasks, then the tasks are given this stopped reason.

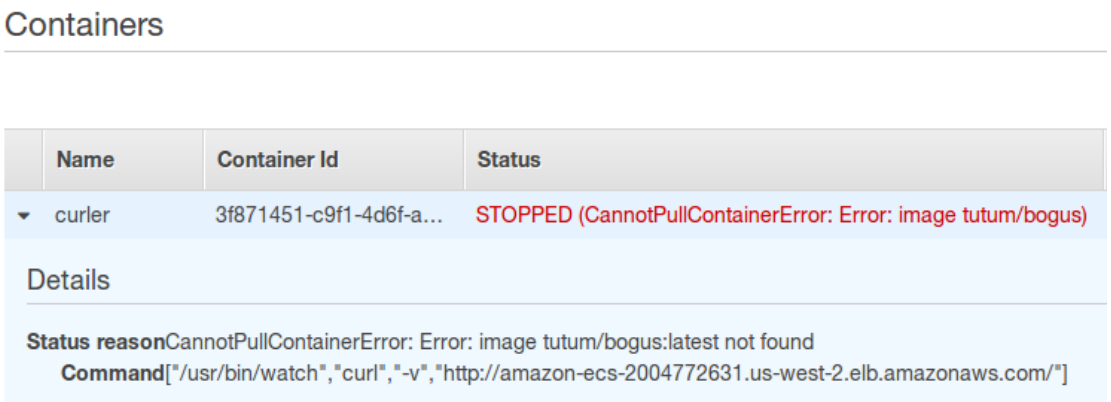
Container instance deregistration forced by user

If you force the deregistration of a container instance with running tasks, then the tasks are given this stopped reason.

Essential container in task exited

Containers marked as `essential` in task definitions cause a task to stop if they exit or die. When an essential container exiting is the cause of a stopped task, the Step 6 (p. 217) can provide more diagnostic information as to why the container stopped.

6. If you have a container that has stopped, expand the container and inspect the **Status reason** row to see what caused the task state to change.



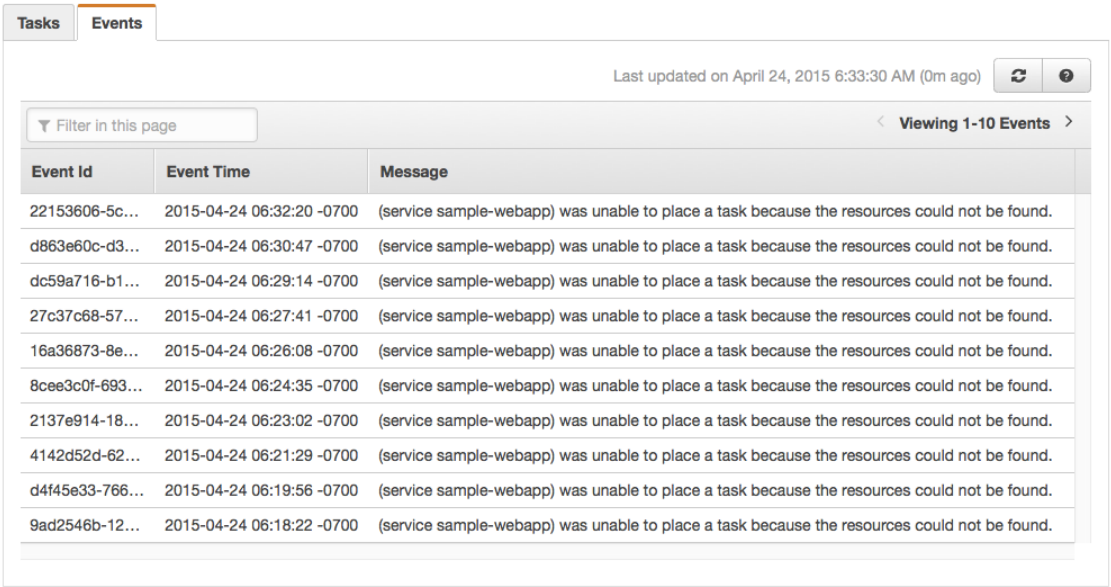
In the previous example, the container image name cannot be found. This can happen if you misspell the image name.

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, select the cluster in which your service resides.
3. On the **Cluster : *clustername*** page, select the service to inspect.
4. On the **Service : *servicename*** page, choose **Events**.



5. Examine the **Message** column for errors or other helpful information.

Service Event Messages

The following are examples of service event messages you may see in the console:

- (service *service-name*) was unable to place a task because the resources could not be found. (p. 219)
- (service *service-name*) (instance *instance-id*) is unhealthy in (elb *elb-name*) due to (reason Instance has failed at least the UnhealthyThreshold number of health checks consecutively.) (p. 220)
- (service *service-name*) is unable to consistently start tasks successfully. (p. 220)

(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 fewer 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.

Not enough available ENI attachment points

Tasks that use the `awsvpc` network mode each receive their own elastic network interface, which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of network interfaces that can be attached to them, and the primary network interface counts as one. For example, a `c4.large` instance may have three network interfaces attached to it. The primary network adapter for the instance counts as one, so you can attach 2 more ENIs to the instance. Because each `awsvpc` task requires a network interface, you can only run two such tasks on this instance type. For more information about how many network interfaces are supported per instance type, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*. You can add container instances to your cluster to provide more available network adapters.

(service *service-name*) (instance *instance-id*) is unhealthy in (elb *elb-name*) due to (reason Instance has failed at least the UnhealthyThreshold number of health checks consecutively.)

This service is registered with a load balancer and the load balancer health checks are failing. For more information, see Troubleshooting Service Load Balancers (p. 222).

(service *service-name*) is unable to consistently start tasks successfully.

This service contains tasks that have failed to start after consecutive attempts. At this point, the service scheduler begins to incrementally increase the time between retries. You should troubleshoot why your tasks are failing to launch. For more information, see Service Throttle Logic (p. 117).

After the service is updated, for example with an updated task definition, the service scheduler resumes normal behavior.

Invalid CPU or Memory Value Specified

When registering a task, if you specify an invalid `cpu` or `memory` value, you receive the following error:

```
An error occurred (ClientException) when calling the RegisterTaskDefinition operation:
 Invalid 'cpu' setting for task. For more information, see the Troubleshooting section of
 the Amazon ECS Developer Guide.
```

To resolve this issue, you must specify a supported value for the task CPU and memory in your task definition.

The `cpu` value can be expressed in CPU units or vCPUs in a task definition but is converted to an integer indicating the CPU units when the task definition is registered. If you are using the EC2 launch type, the supported values are between `128` CPU units (`0.125` vCPUs) and `10240` CPU units (`10` vCPUs). If you are using the Fargate launch type, you must use one of the values in the following table, which determines your range of supported values for the `memory` parameter.

The `memory` value can be expressed in MiB or GB in a task definition but is converted to an integer indicating the MiB when the task definition is registered. If you are using the EC2 launch type, you must specify an integer. If you are using the Fargate launch type, you must use one of the values in the following table, which determines your range of supported values for the `cpu` parameter.

Supported task CPU and memory values for Fargate tasks are as follows.

CPU value	Memory value (MiB)
256 (.25 vCPU)	512 (0.5GB), 1024 (1GB), 2048 (2GB)
512 (.5 vCPU)	1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)
1024 (1 vCPU)	2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)
2048 (2 vCPU)	Between 4096 (4GB) and 16384 (16GB) in increments of 1024 (1GB)

CPU value	Memory value (MiB)
4096 (4 vCPU)	Between 8192 (8GB) and 30720 (30GB) in increments of 1024 (1GB)

Cannot Pull Container Image Error

The following Docker errors indicate that when creating a task, the container image specified could not be retrieved.

Connection timed out

When a Fargate task is launched, its elastic network interface requires a route to the internet to pull container images. If you receive an error similar to the following when launching a task, it is because a route to the internet does not exist:

```
CannotPullContainerError: API error (500): Get https://111122223333.dkr.ecr.us-
east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection"
```

To resolve this issue, you can:

Image not found

When you specify an Amazon ECR image in your container definition, you must use the full ARN or URI of your ECR repository along with the image name in that repository. If the repository or image cannot be found, you receive the following error:

```
CannotPullContainerError: API error (404): repository 111122223333.dkr.ecr.us-
east-1.amazonaws.com/<repo>/<image> not found
```

To resolve this issue, verify the repository ARN or URI and the image name. Also ensure that you have set up the proper access using the task execution IAM role. For more information about the task execution role, see Amazon ECS Task Execution IAM Role (p. 156).

Insufficient disk space

If the root volume of your container instance has insufficient disk space when pulling the container image, you see an error similar to the following:

```
CannotPullContainerError: write /var/lib/docker/tmp/GetImageBlob111111111: no space
 left on device
```

To resolve this issue, you need to free up disk space.

If you are using the Amazon ECS-optimized AMI you can use the following command to retrieve the twenty largest files on your filesystem:

```
du -Sh / | sort -rh | head -20
```

Example output:

```
5.7G    /var/lib/docker/
containers/50501b5f4cbf90b406e0ca60bf4e6d4ec8f773a6c1d2b451ed8e0195418ad0d2
1.2G    /var/log/ecs
```

```
594M    /var/lib/docker/devicemapper/mnt/
c8e3010e36ce4c089bf286a623699f5233097ca126ebd5a700af023a5127633d/rootfs/data/logs
...
```

In some cases, like this example above, the root volume may be filled out by a running container. If the container is using the default `json-file` log driver without a `max-size` limit, it may be that the log file is responsible for most of that space used. You can use the `docker ps` command to verify which container is using the space by mapping the directory name from the output above to the container ID. For example:

```
CONTAINER ID   IMAGE                              COMMAND             CREATED
 STATUS              PORTS                              NAMES
50501b5f4cbf   amazon/amazon-ecs-agent:latest   "/agent"            4 days ago
 Up 4 days                                              ecs-agent
```

By default, when using the `json-file` log driver, Docker captures the standard output (and standard error) of all of your containers and writers them in files using the JSON format. You are able to set the `max-size` as a log driver option, which prevents the log file from taking up too much space. For more information, see Configure logging drivers in the Docker documentation.

The following is a container definition snippet showing how to use this option:

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "256m"
    }
}
```

An alternative if your container logs are taking up too much disk space is to use the `awslogs` log driver. The `awslogs` log driver sends the logs to CloudWatch, which frees up the disk space that would otherwise be used for your container logs on the container instance. For more information, see Using the awslogs Log Driver (p. 55).

Troubleshooting Service Load Balancers

Amazon ECS services can register tasks with an Elastic Load Balancing load balancer. Load balancer configuration errors are common causes for stopped tasks. If your stopped tasks were started by services that use a load balancer, consider the following possible causes.

Important
Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer. The Amazon ECS service scheduler ignores tasks in an `UNHEALTHY` state that are behind a Classic Load Balancer.

Container instance security group

If your container is mapped to port 80 on your container instance, your container instance security group must allow inbound traffic on port 80 for the load balancer health checks to pass.

Elastic Load Balancing load balancer not configured for all Availability Zones

Your load balancer should be configured to use all of the Availability Zones in a region, or at least all of the Availability Zones in which your container instances reside. If a service uses a load balancer and starts a task on a container instance that resides in an Availability Zone that the load balancer is not configured to use, the task never passes the health check and it is killed.

Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that do not exist. If a container instance is determined to be unhealthy, it is removed from the load balancer. Be sure to verify that the following parameters are configured correctly for your service load balancer.

Ping Port

The **Ping Port** value for a load balancer health check is the port on the container instances that the load balancer checks to determine if it is healthy. If this port is misconfigured, the load balancer likely deregisters your container instance from itself. This port should be configured to use the `hostPort` value for the container in your service's task definition that you are using with the health check.

Ping Path

This value is often set to `index.html`, but if your service does not respond to that request, then the health check fails. If your container does not have an `index.html` file, you can set this to `/` to target the base URL for the container instance.

Response Timeout

This is the amount of time that your container has to return a response to the health check ping. If this value is lower than the amount of time required for a response, the health check fails.

Health Check Interval

This is the amount of time between health check pings. The shorter your health check intervals are, the faster your container instance can reach the **Unhealthy Threshold**.

Unhealthy Threshold

This is the number of times your health check can fail before your container instance is considered unhealthy. If you have an unhealthy threshold of 2, and a health check interval of 30 seconds, then your task has 60 seconds to respond to the health check ping before it is assumed unhealthy. You can raise the unhealthy threshold or the health check interval to give your tasks more time to respond.

Unable to update the service *`servicename`*: Load balancer container name or port changed in task definition

If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service.

Troubleshooting IAM Roles for Tasks

If you are having trouble configuring IAM roles for tasks in your cluster, you can try this known good configuration to help debug your own configuration.

The following procedure helps you to:

- Create a CloudWatch Logs log group to store your test logs.
- Create a task IAM role that has full Amazon ECS permissions.
- Register a task definition with a known good AWS CLI configuration that is compatible with IAM roles for tasks.
- Run a task from that task definition to test your container instance support for IAM roles for tasks.

- View the container logs from that task in CloudWatch Logs to verify that it works.

To test IAM roles for tasks with a known good configuration

1. Create a CloudWatch Logs log group called `ecs-tasks`.
   a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
   b. In the left navigation pane, choose **Logs**, **Actions**, **Create log group**.
   c. For **Log Group Name**, enter `ecs-tasks` and choose **Create log group**.
2. Create an IAM role for your task to use.
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the navigation pane, choose **Roles**, **Create role**.
   c. For **Select type of trusted entity**, choose **Elastic Container Service**.
   d. For **Select your use case**, choose **Elastic Container Service Task**, **Next: Permissions**.
   e. On the **Attached permissions policy** page, choose **AmazonEC2ContainerServiceFullAccess**, **Next: Review**.
   f. On the **Review** page, for **Role name**, enter `ECS-task-full-access` and choose **Create role**.
3. Register a task definition that uses your new role.
   a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
   b. In the navigation pane, choose **Task Definitions**.
   c. On the **Task Definitions** page, choose **Create new Task Definition**.
   d. On the **Select launch type compatibility** page, choose **EC2**, **Next step**.
   e. Scroll to the bottom of the page and choose **Configure via JSON**.
   f. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

      Note
      Replace the `awslogs-region` value with the region in which you created your CloudWatch Logs log group.

```
{
    "taskRoleArn": "ECS-task-full-access",
    "containerDefinitions": [
        {
            "memory": 128,
            "essential": true,
            "name": "amazonlinux",
            "image": "amazonlinux",
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": [
                "yum install -y aws-cli; aws ecs list-tasks --region us-west-2"
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "ecs-tasks",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "iam-role-test"
                }
            }
        }
    ],
```

```
    "family": "iam-role-test",
    "requiresCompatibilities": [
        "EC2"
    ],
    "volumes": [],
    "placementConstraints": [],
    "networkMode": null,
    "memory": null,
    "cpu": null
}
```

g. Verify your information and choose **Create**.

4. Run a task from your task definition.

   a. On the **Task Definition: iam-role-test** registration confirmation page, choose **Actions**, **Run Task**.

   b. On the **Run Task** page, choose the **EC2** launch type, a cluster, and then choose **Run Task** to run your task.

5. View the container logs in the CloudWatch Logs console.

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

   b. In the left navigation pane, choose **Logs**.

   c. Select the `ecs-tasks` log group.

   d. Select the most recent log stream.

   e. Scroll down to view the last lines of the log stream. You should see the output of the **aws ecs list-tasks** command.

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/d48feb62-46e2-4cbc-a36b-
e0400b993d1d"
    ]
}
```

Document History

The following table describes the major updates and new features for the *Amazon ECS User Guide for AWS Fargate*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
Resource tagging	Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances. For more information, see Resources and Tags (p. 119).	15 Nov 2018
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions. For more information, see AWS Fargate Platform Versions (p. 19).	07 Nov 2018
Service limits updated	The following service limits were updated: • Number of tasks using the Fargate launch type, per region, per account was raised from `20` to `50`. • Number of public IP addresses for tasks using the Fargate launch type was raised from `20` to `50`. For more information, see Amazon ECS Service Limits (p. 211).	31 Oct 2018
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the EU (London) Region. For more information, see AWS Fargate Platform Versions (p. 19).	26 Oct 2018
Private registry authentication support for Amazon ECS using AWS Fargate tasks	Amazon ECS introduced support for Fargate tasks using private registry authentication using AWS Secrets Manager. This feature enables you to store your credentials securely and then reference them in your container definition, which allows your tasks to use private images. For more information, see Private Registry Authentication for Tasks (p. 61).	10 Sept 2018
Amazon ECS CLI v1.8.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for Docker volumes in Docker compose files. • Added support for task placement constraints and strategies in Docker compose files. • Added support for private registry authentication in Docker compose files.	7 Sept 2018

Change	Description	Date
	• Added support for `--force-update` on `compose up` to force relaunching of tasks. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	
Amazon ECS service discovery region expansion	Amazon ECS service discovery has expanded support to the Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), EU (Frankfurt), and EU (London) regions. For more information, see Service Discovery (p. 97).	30 August 2018
Scheduled tasks with Fargate tasks support	Amazon ECS introduced support for scheduled tasks for the Fargate launch type. For more information, see Scheduled Tasks (`cron`) (p. 70).	28 August 2018
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the EU (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) regions. For more information, see AWS Fargate Platform Versions (p. 19).	19 July 2018
Amazon ECS CLI v1.7.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for container `healthcheck` and `devices` in Docker compose files. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	18 July 2018

Change	Description	Date
Amazon ECS service scheduler strategies added	Amazon ECS introduced the concept of service scheduler strategies. There are two service scheduler strategies available: • `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica (p. 75). • `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 75). **Note** Fargate tasks do not support the `DAEMON` scheduling strategy. For more information, see Service Scheduler Concepts (p. 74).	12 June 2018
Amazon ECS CLI v1.6.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for Docker compose file syntax version 3. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	5 June 2018
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the US East (Ohio), US West (Oregon), and EU West (Ireland) regions. For more information, see AWS Fargate Platform Versions (p. 19).	26 April 2018

Change	Description	Date
Amazon ECS CLI v1.5.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for the ECS CLI to automatically retrieve the latest stable Amazon ECS-optimized AMI by querying the SSM Parameter Store API during the cluster resource creation process. This requires the user account that you are using to have the required SSM permissions. • Added support for the `shm_size` and `tmpfs` parameters in compose files. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	19 April 2018
Amazon ECS CLI download verification	Added new PGP signature method for verifying the Amazon ECS CLI installation file. For more information, see Installing the Amazon ECS CLI (p. 179).	5 April 2018
AWS Fargate Platform Version	New AWS Fargate platform version released, which contains: • Added support for Amazon ECS Task Metadata Endpoint (p. 202). • Added support for Health Check (p. 36). • Added support for Service Discovery (p. 97) For more information, see AWS Fargate Platform Versions (p. 19).	26 March 2018
Amazon ECS Service Discovery	Added integration with Route 53 to support Amazon ECS service discovery. For more information, see Service Discovery (p. 97).	22 March 2018
Amazon ECS CLI v1.4.2	New version of the Amazon ECS CLI released, which added the following functionality: • Updated the AMI to `amzn-ami-2017.09.k-amazon-ecs-optimized`. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	20 March 2018

Change	Description	Date
Amazon ECS CLI v1.4.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for the us-gov-west-1 region. • Added `--force-deployment` flag for the compose service command. • Added support for `aws_session_token` in ECS profiles. • Updated the AMI to `amzn-ami-2017.09.j-amazon-ecs-optimized`. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	09 March 2018
Container Health Checks	Added support for Docker health checks in container definitions. For more information, see Health Check (p. 36).	08 March 2018
Amazon ECS Task Metadata Endpoint	Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and Docker stats are available to tasks that use the `awsvpc` network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Amazon ECS Task Metadata Endpoint (p. 202).	8 February 2018
Amazon ECS Service Auto Scaling using target tracking policies	Added support for ECS Service Auto Scaling using target tracking policies in the Amazon ECS console. For more information, see Target Tracking Scaling Policies (p. 91). Removed the previous tutorial for step scaling in the ECS first run wizard. This was replaced with the new tutorial for target tracking.	8 February 2018
Amazon ECS CLI v1.3.0	New version of the Amazon ECS CLI released, which added the following functionality: • Ability to create empty clusters with the up command. • Added `--health-check-grace-period` flag for the compose service up command. • Updated the AMI to `amzn-ami-2017.09.g-amazon-ecs-optimized`. For more information, see the Amazon ECS Command Line Reference in the *Amazon Elastic Container Service Developer Guide*.	19 January 2018
Elastic Load Balancing health check initialization wait period	Added ability to specify a wait period for health checks. For more information, see (Optional) Health Check Grace Period (p. 110).	27 December 2017

Change	Description	Date
New service scheduler behavior	Updated information about the behavior for service tasks that fail to launch. Documented new service event message that triggers when a service task has consecutive failures. For more information about this updated behavior, see Additional Service Concepts (p. 75).	11 January 2018
Task-level CPU and memory	Added support for specifying CPU and memory at the task-level in task definitions. For more information, see TaskDefinition.	12 December 2017
Amazon ECS console AWS CodePipeline integration	Added Amazon ECS integration with CodePipeline. CodePipeline supports Amazon ECS as a deployment option to help set up deployment pipelines. For more information, see Tutorial: Continuous Deployment with AWS CodePipeline (p. 206).	12 December 2017
Task execution role	The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. The following actions are covered by the task execution role: • Calls to Amazon ECR to pull the container image • Calls to CloudWatch to store container application logs For more information, see Amazon ECS Task Execution IAM Role (p. 156).	7 December 2017
Amazon ECS CLI v1.1.0 with Fargate support	New version of the Amazon ECS CLI released, which added the following features: • Support for task networking • Support for AWS Fargate • Support for viewing CloudWatch Logs data from a task For more information, see ECS CLI changelog.	29 November 2017
AWS Fargate GA	Added support for launching Amazon ECS services using the Fargate launch type. For more information, see Amazon ECS Launch Types (p. 50).	29 November 2017

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.