
Amazon Elastic Container Service

Developer Guide



Amazon Elastic Container Service: Developer Guide

Copyright © 2023 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
Amazon ECS terminology and components	1
Amazon ECS capacity	2
Amazon ECS controller	3
Amazon ECS provisioning	3
Application lifecycle	3
.....	4
Amazon ECS features	4
Pricing	4
Common use cases	5
Additional resources	6
Related services	6
New Amazon ECS console	7
Configuration which is not available in the new console	7
Getting started	10
Set up	10
Sign up for an AWS account	10
Create an administrative user	10
Create the credentials to connect to your EC2 instance	11
Create a virtual private cloud	12
Create a security group	12
Install the AWS CLI	15
Creating a container image	15
Prerequisites	15
Create a Docker image	16
Push your image to Amazon Elastic Container Registry	18
Clean up	19
Next steps	19
Using the console with Linux containers on AWS Fargate	19
Prerequisites	20
Step 1: Create the cluster	20
Step 2: Create a task definition	21
Step 3: Create the service	21
Step 4: View your service	22
Step 5: Clean up	22
Using the console with Windows containers on AWS Fargate	23
Prerequisites	23
Step 1: Create a cluster	23
Step 2: Register a Windows task definition	24
Step 3: Create a service with your task definition	25
Step 4: View your service	25
Step 5: Clean Up	26
Getting started with the console using Windows on Amazon EC2	26
Prerequisites	26
Step 1: Create a cluster	27
Step 2: Register a task definition	28
Step 3: Create a Service	29
Step 4: View your Service	29
Step 5: Clean Up	30
Developer tools overview	31
AWS Management Console	31
AWS Command Line Interface	31
AWS CloudFormation	32
AWS Copilot CLI	32

AWS CDK	32
AWS App2Container	33
Amazon ECS CLI	33
Docker Desktop integration with Amazon ECS	33
AWS SDKs	34
Summary	34
Using the AWS Copilot CLI	35
Installing the AWS Copilot CLI	35
Getting started with AWS Copilot	41
Prerequisites	41
Deploy your application using one command	41
Deploy your application step by step	42
Using the AWS CDK	45
Step 1: Set up your AWS CDK project	46
Step 2: Use the AWS CDK to define a containerized web server on Fargate	48
Step 3: Test the web server	51
Step 4: Clean up	52
Next steps	52
Creating Amazon ECS resources with AWS CloudFormation	52
Amazon ECS and AWS CloudFormation templates	52
Example templates	53
Using the AWS CLI to create resources from templates	58
Learn more about AWS CloudFormation	58
Using the Amazon ECS CLI	58
Installing the Amazon ECS CLI	59
Configuring the Amazon ECS CLI	64
AWS Fargate	67
Task definitions	67
Network mode	68
Task Operating Systems	68
Task CPU architecture	68
Task CPU and memory	69
Task resource limits	69
Logging	70
Amazon ECS task execution IAM role	70
Example Amazon Linux 2 task definition	71
Example Windows task definition	71
Task storage	72
Tasks and services	67
Task networking	73
Fargate Spot	73
Service load balancing	73
Clusters	74
Usage metrics	74
Task maintenance	74
Savings plans	75
Lazy loading SOCI container images	75
Windows containers on AWS Fargate considerations	76
Platform Versions	77
Linux platform versions	77
Windows platform versions	81
Task definitions	83
Task definition states	83
Amazon ECS resources that can block a deletion	84
Architecting your application for Amazon ECS	85
Launch types	85
Container images	87

Task definitions	89
Networking modes	89
Using data volumes in tasks	100
Managing container swap space	123
Amazon EC2 Windows task definition considerations	124
Creating a task definition using the console	125
JSON validation	125
AWS CloudFormation stacks	125
Updating a task definition using the console	139
JSON validation	139
Deregistering a task definition revision using the console	140
AWS CloudFormation stacks	125
Deleting a task definition revision using the console	141
Amazon ECS resources that can block a deletion	84
Task definition use cases	142
Working with GPUs on Amazon ECS	142
Using video transcoding on Amazon ECS	146
Using machine learning on Amazon ECS	153
Working with 64-bit ARM workloads on Amazon ECS	159
Using the awslogs log driver	161
Custom log routing	166
Private registry authentication for tasks	195
Passing environment variables	197
Passing sensitive data to a container	200
Example task definitions	213
Example: Webserver	213
Example: splunk log driver	215
Example: fluentd log driver	215
Example: gelf log driver	216
Example: Amazon ECR image and task definition IAM role	216
Example: Entrypoint with command	217
Example: Container dependency	217
Windows sample task definitions	218
Clusters	220
Capacity providers	220
Capacity provider concepts	220
Capacity provider types	221
Capacity provider considerations	222
AWS Fargate capacity providers	222
Amazon EC2 Auto Scaling group capacity providers	224
Cluster auto scaling	226
How cluster auto scaling works	226
Cluster auto scaling considerations	228
Managed termination protection	229
Managed scale-out behavior	229
Managed scale-in behavior	231
Target tracking considerations	232
Update on the way Amazon ECS creates resources for cluster auto scaling	232
Turn on cluster auto scaling	233
Turn off cluster auto scaling	234
Cluster management	235
Creating a cluster for the Fargate launch type using the console	236
Creating a cluster for the Amazon EC2 launch type using the console	237
Updating a cluster using the console	240
Deleting a cluster using the console	240
Creating a capacity provider for a cluster using the console	241
Updating a capacity provider for a cluster using the console	242

Deleting a capacity provider for a cluster using the console	242
Account settings	244
Amazon Resource Names (ARNs) and IDs	246
ARN and resource ID format timeline	247
AWS Fargate Federal Information Processing Standard (FIPS-140) compliance	247
Tagging authorization	248
Tagging authorization timeline	248
Viewing account settings using the console	249
Modifying account settings	249
Reverting to the default Amazon ECS account settings	249
Account setting management using the AWS CLI	249
Container instances	251
Container instance concepts	251
Container instance lifecycle	252
Choosing the Amazon EC2 instance type	252
Using Amazon EC2 Spot	253
Linux instances	254
Amazon ECS-optimized AMI	255
Bottlerocket	268
Launching a container instance	272
Bootstrap Container Instances	280
Starting a task at container instance launch time	281
Elastic network interface trunking	283
Memory Management	304
Manage container instances remotely	306
Windows instances	307
Amazon ECS-optimized AMI	308
Launching a container instance	326
Bootstrap Container Instances	333
Connect to your container Windows instance	335
Deregister a container instance	335
External instances	336
Supported operating systems and system architectures	337
Considerations	338
IAM permissions	340
Registering an external instance to a cluster	342
Deregistering an external instance	346
Running workloads on external instances	349
Updating the AWS Systems Manager Agent and Amazon ECS container agent	350
.....	354
Container instance draining	354
Draining behavior for services	354
Draining behavior for standalone tasks	354
Draining container instances	355
Container agent	356
Installing the Amazon ECS container agent	356
Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance	357
Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance	357
Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance	358
Running the Amazon ECS agent with host network mode	361
Container agent versions	362
Amazon ECS-optimized AMI	362
Using other Linux Operating Systems	362
Additional information	362
Amazon EC2 Windows containers	363
Windows container caveats	364
Getting started with Windows containers	364

Updating the Amazon ECS container agent	364
Checking the Amazon ECS container agent version	365
Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI	366
Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs)	368
Container agent configuration	370
Available Parameters	370
Storing container instance configuration in Amazon S3	370
Private registry authentication for container instances	371
Authentication formats	372
Turning on private registries	374
Automated task and image cleanup	375
Tunable parameters	375
Cleanup workflow	376
Container metadata file	376
Turning on container metadata	376
Container metadata file locations	377
Container metadata file format	377
Task metadata endpoint	380
Task metadata endpoint version 4	380
Task Metadata Endpoint version 3	397
Task Metadata Endpoint version 2	402
Container agent endpoint	407
Task scale-in protection endpoint	407
Container agent introspection	410
HTTP proxy configuration	412
Amazon Linux container instance configuration	412
Windows container instance configuration	415
Using gMSAs for Windows Containers	416
Considerations	416
Prerequisites	417
Setup	417
Using gMSAs for Linux Containers	421
Considerations	422
Prerequisites	422
Setup	423
CredSpec file	427
Updating the Amazon ECS container agent with the console	428
Scheduling tasks	429
Running a standalone task using the Amazon ECS console	430
Stopping tasks using the console	434
Task placement	434
Task groups	435
Task placement strategies	435
Task placement constraints	437
Cluster query language	442
Scheduled tasks	445
Create a scheduled task in the EventBridge Scheduler console	445
View your EventBridge scheduled tasks in the console	448
Edit an EventBridge scheduled task	449
Task lifecycle	449
Lifecycle states	450
Task maintenance	451
Understanding the task retirement notice	452
Services	453
Service scheduler concepts	453
Daemon	453
Replica	455

Additional service concepts	455
Creating a service using the console	456
Quickly create a service	456
Create a service using defined parameters	457
Updating a service using the console	465
Updating a blue/green deployment configuration using the console	471
Deleting a service using the console	472
Deployment types	472
Rolling update	473
Blue/Green deployment with CodeDeploy	477
External deployment	481
Service load balancing	486
Load balancer types	487
Creating a load balancer	490
Registering multiple target groups with a service	496
Service auto scaling	498
Service auto scaling and deployments	498
IAM permissions required for service auto scaling	498
Considerations	499
AWS CLI and SDK experience	500
Target tracking scaling policies	501
Step scaling policies	501
Interconnecting services	502
Choosing an interconnection method	502
Network mode compatibility table	503
Service Connect	503
Service discovery	522
Task scale-in protection	525
Task scale-in protection mechanisms	526
Task scale-in protection considerations	527
IAM permissions required for task scale-in protection	527
Service throttle logic	528
Resources and tags	529
Tagging your resources	529
How resources are tagged	529
Tagging resources on creation	531
Tag restrictions	531
Amazon ECS-managed tags	532
Tagging your resources for billing	532
Working with tags using the console	533
Working with tags using the CLI or API	535
Service quotas	536
Amazon ECS service quotas	536
AWS Fargate service quotas	538
Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console ..	539
AWS Fargate Regions	540
Supported Regions for Linux containers on AWS Fargate	540
Supported Regions for Windows containers on AWS Fargate	542
Usage Reports	543
Task-level cost and usage	544
Monitoring	545
Monitoring tools	546
Automated Tools	546
Manual Tools	546
CloudWatch metrics	547
Using CloudWatch metrics	547
Available metrics and dimensions	548

Cluster reservation	554
Cluster utilization	555
Service utilization	556
Service RUNNING task count	557
Viewing Amazon ECS metrics	557
Events and EventBridge	558
Amazon ECS events	559
Handling events	571
CloudWatch Container Insights	572
Container Insights considerations	572
Setting up CloudWatch Container Insights for cluster and service level metrics	573
To change the default for Container Insights for all users using the console	573
To change the default for Container Insights for all users using the command line	573
To change the default for Container Insights for a specific user using the command line	574
To turn on Container Insights for a specific cluster using the command line	574
Use CloudWatch Container Insights to view Amazon ECS lifecycle events	574
Container instance health	576
Collecting application trace data	576
Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray	577
Specifying the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration in your task definition	578
Collecting application metrics	579
Exporting application metrics to Amazon CloudWatch	579
Exporting application metrics to Amazon Managed Service for Prometheus	582
Logging Amazon ECS API calls with AWS CloudTrail	584
Amazon ECS information in CloudTrail	584
Understanding Amazon ECS log file entries	585
Compute Optimizer recommendations	586
Task size recommendations for Fargate	586
Security	587
Identity and Access Management	587
Audience	588
Authenticating with identities	588
Managing access using policies	590
How Amazon Elastic Container Service works with IAM	592
Identity-based policy examples	599
AWS managed policies for Amazon ECS	611
Using service-linked roles	624
Task execution IAM role	626
Task IAM role	631
Additional configuration for Windows task role	637
Container instance IAM role	638
ECS Anywhere IAM role	642
CodeDeploy IAM Role	644
CloudWatch Events IAM Role	648
Tag resources during creation	651
Troubleshooting	654
Logging and Monitoring	655
Compliance validation	656
AWS Fargate FIPS-140 compliance	657
Considerations	657
Use FIPS on Fargate	657
Use CloudTrail for auditing	658
Infrastructure Security	659
Interface VPC endpoints (AWS PrivateLink)	659
Working with other services	663
Using Amazon ECR with Amazon ECS	663

Using Amazon ECR Images with Amazon ECS	663
Using Local Zones, Wavelength Zones, and AWS Outposts	664
Local Zones	664
Wavelength Zones	665
AWS Outposts	665
Amazon Elastic Container Service on AWS Outposts	665
Prerequisites	665
Limitations	665
Network Connectivity Considerations	666
Creating an Amazon ECS Cluster on an AWS Outposts	666
Use App Mesh with Amazon ECS	668
AWS Deep Learning Containers on Amazon ECS	668
Deep Learning Containers with Elastic Inference on Amazon ECS	669
Using AWS User Notifications with Amazon ECS	669
Example	669
Tutorials	670
Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI	670
Prerequisites	670
Step 1: Create a Cluster	671
Step 2: Register a Linux Task Definition	671
Step 3: List Task Definitions	672
Step 4: Create a Service	673
Step 5: List Services	673
Step 6: Describe the Running Service	673
Step 7: Test	675
Step 8: Clean Up	678
Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI	678
Prerequisites	678
Step 1: Create a Cluster	679
Step 2: Register a Windows Task Definition	679
Step 3: List task definitions	680
Step 4: Create a service	680
Step 5: List services	681
Step 6: Describe the Running Service	681
Step 7: Clean Up	683
Tutorial: Creating a cluster with an EC2 task using the AWS CLI	683
Prerequisites	683
Step 1: Create a Cluster	684
Step 2: Launch an Instance with the Amazon ECS AMI	684
Step 3: List Container Instances	684
Step 4: Describe your Container Instance	685
Step 5: Register a Task Definition	687
Step 6: List Task Definitions	688
Step 7: Run a Task	688
Step 8: List Tasks	689
Step 9: Describe the Running Task	689
Tutorial: Using cluster auto scaling with the AWS Management Console and the Amazon ECS console ..	690
Prerequisites	690
Step 1: Create an Amazon ECS cluster	690
Step 2: Register a task definition	691
Step 3: Run a task	692
Step 4: Verify	692
Step 5: Clean up	693
Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets	693
Prerequisites	694
Step 1: Create an Secrets Manager Secret	694
Step 2: Update Your Task Execution IAM Role	694

Step 3: Create an Amazon ECS Task Definition	695
Step 4: Create an Amazon ECS Cluster	696
Step 5: Run an Amazon ECS Task	696
Step 6: Verify	696
Step 7: Clean Up	697
Tutorial: Creating a service using Service Discovery	697
Prerequisites	698
Step 1: Create the Service Discovery resources in AWS Cloud Map	698
Step 2: Create the Amazon ECS resources	699
Step 3: Verify Service Discovery in AWS Cloud Map	701
Step 4: Clean up	702
Tutorial: Creating a service using a blue/green deployment	704
Prerequisites	704
Step 1: Create an Application Load Balancer	704
Step 2: Create an Amazon ECS cluster	705
Step 3: Register a task definition	706
Step 4: Create an Amazon ECS service	706
Step 5: Create the AWS CodeDeploy resources	707
Step 6: Create and monitor a CodeDeploy deployment	709
Step 7: Clean up	711
Tutorial: Listening for Amazon ECS CloudWatch Events	713
Prerequisite: Set up a test cluster	713
Step 1: Create the Lambda function	713
Step 2: Register an event rule	713
Step 3: Create a task definition	714
Step 4: Test your rule	715
Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events	715
Prerequisite: Set up a test cluster	715
Prerequisite: Configure permissions for Amazon SNS	715
Step 1: Create and subscribe to an Amazon SNS topic	716
Step 2: Register an event rule	716
Step 3: Test your rule	717
Tutorial: Using Amazon EFS	718
Step 1: Create an Amazon ECS cluster	718
Step 2: Create a security group for the Amazon EFS file system	719
Step 3: Create an Amazon EFS file system	719
Step 4: Add content to the Amazon EFS file system	720
Step 5: Create a task definition	721
Step 6: Run a task and view the results	722
Tutorial: Using FSx for Windows File Server	723
Prerequisites for the tutorial	723
Step 1: Create IAM access roles	724
Step 2: Create Windows Active Directory (AD)	724
Step 3: Verify and update your security group	725
Step 4: Create an FSx for Windows File Server file system	726
Step 5: Create an Amazon ECS cluster	727
Step 6: Create an Amazon ECS instance	727
Step 7: Register a Windows task definition	728
Step 8: Run a task and view the results	730
Step 9: Clean up	730
Tutorial: Deploying Fluent Bit on Amazon ECS for Windows containers	731
Prerequisites	733
Step 1: Create the IAM access roles	733
Step 2: Create an Amazon ECS Windows container instance	734
Step 3: Configure Fluent Bit	735
Step 4: Register a Windows Fluent Bit task definition which routes the logs to CloudWatch	736

Step 5: Run the <code>ecs-windows-fluent-bit</code> task definition as an Amazon ECS service using the daemon scheduling strategy	737
Step 6: Register a Windows task definition which generates the logs	738
Step 7: Run the <code>windows-app-task</code> task definition	739
Step 8: Verify the logs on CloudWatch	739
Step 9: Clean up	740
Fargate AWS CLI capacity provider examples	741
Creating a new cluster that uses Fargate capacity providers	741
Adding Fargate capacity providers to an existing cluster	741
Running tasks using a Fargate capacity provider	742
Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI	743
Prerequisites	743
Step 1: Create and configure the gMSA account on Active Directory Domain Services (AD DS)	744
Step 2: Upload Credentials to Secrets Manager	745
Step 3: Modify your CredSpec JSON to include domainless gMSA information	746
Step 4: Upload CredSpec to Amazon S3	747
Step 5: (Optional) Create an Amazon ECS cluster	747
Step 6: Create an IAM role for container instances	748
Step 7: Create a custom task execution role	748
Step 8: Create a task role for Amazon ECS Exec	749
Step 9: Register a task definition	750
Step 10: Register a Windows container instance	751
Step 11: Verify the container instance	751
Step 12: Run a Windows task	752
Step 13: Verify the container has gMSA credentials	753
Step 14: Clean up	753
Debugging	754
Troubleshooting	755
Using Amazon ECS Exec for debugging	755
Architecture	755
Considerations for using ECS Exec	756
Prerequisites for using ECS Exec	757
Using ECS Exec	757
Logging and Auditing using ECS Exec	759
Using IAM policies to limit access to ECS Exec	762
Troubleshooting issues with ECS Exec	764
Troubleshooting ECS Anywhere issues	764
External instance registration issues	765
External instance network issues	765
Issues running tasks	765
Checking stopped tasks for errors	766
Additional resources	6
CannotPullContainer task errors	767
Service event messages	770
Service event messages	770
Invalid CPU or memory value specified	776
CannotCreateContainerError: API error (500): devmapper	777
Troubleshooting service load balancers	778
Troubleshooting service auto scaling	779
Using Docker debug output	780
Amazon ECS Log File Locations	781
Amazon ECS Container Agent Log	781
Amazon ECS <code>ecs-init</code> Log	783
IAM Roles for Tasks Credential Audit Log	784
Amazon ECS logs collector	784
Agent introspection diagnostics	786
Docker diagnostics	787

List Docker containers	787
View Docker Logs	788
Inspect Docker Containers	789
AWS Fargate throttling quotas	789
Throttling the RunTask API	790
Adjusting rate quotas	790
API failure reasons	790
Troubleshooting IAM Roles for Tasks	796
Parameter references and resource templates	799
Task definition parameters	799
Family	799
Launch types	799
Task role	800
Task execution role	800
Network mode	800
Runtime platform	801
Task size	802
Container definitions	804
Elastic Inference accelerator name	834
Task placement constraints	834
Proxy configuration	835
Volumes	836
Tags	841
Other task definition parameters	841
Task definition template	843
Service definition parameters	847
Launch type	847
Capacity provider strategy	847
Task definition	848
Platform operating system	849
Platform version	849
Cluster	849
Service name	850
Scheduling strategy	850
Desired count	850
Deployment configuration	851
Deployment controller	852
Task placement	853
Tags	854
Network configuration	855
Client token	861
Service definition template	861
Classic Amazon Elastic Container Service console	863
Getting started using the classic console	863
Using the classic console with Linux containers on AWS Fargate	864
Using the classic console with Windows containers on AWS Fargate	867
Using the classic console with Amazon EC2	871
Using the classic console with Windows containers	875
Cluster management in the classic Amazon ECS console	879
Creating a cluster using the classic console	879
Creating an Auto Scaling group capacity provider using the classic console	882
Updating an Auto Scaling group capacity provider using the classic console	883
Creating a cluster with an Auto Scaling group capacity provider	885
Deleting an Auto Scaling group capacity provider using the classic console	885
Deleting a cluster using the classic console	886
Task definition management in the classic Amazon ECS console	887
Creating a task definition using the classic console	887

Updating a task definition using the classic console	894
Deregistering a task definition revision	895
Task management in the classic Amazon ECS console	896
Run a standalone task in the classic Amazon ECS console	896
Service Management	899
Creating a service	899
Updating a service using the classic console	911
Deleting a service using the classic console	913
Tag Management	914
Adding and deleting tags on an individual resource using the classic console	914
Account setting Management	915
Modifying account settings using the classic console	915
Viewing account settings using the classic console	916
Container instance management	916
Connect to your container Windows instance using the classic console	916
Deregister a container instance	917
Registering an external instance to a cluster using the classic console	918
Deregistering an external instance using the classic console	919
Container agent management	921
Updating the Amazon ECS container agent with the classic console	921
Monitoring and troubleshooting	922
Viewing Cluster Metrics	922
Viewing Service Metrics	922
Checking stopped tasks for errors in the Amazon ECS classic console	923
Related information	925
Amazon ECS API reference	926
Document history	927
AWS glossary	949

What is Amazon Elastic Container Service?

Join us for an Amazon ECS and AWS Fargate Container Workshop and Operational Bootcamp. This event is for organizations new to Amazon ECS or wanting to learn more about AWS Fargate. Register now at [Virtual Session August 8-9, 2023](#).

Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. As a fully managed service, Amazon ECS comes with AWS configuration and operational best practices built-in. It's integrated with both AWS and third-party tools, such as Amazon Elastic Container Registry and Docker. This integration makes it easier for teams to focus on building the applications, not the environment. You can run and scale your container workloads across AWS Regions in the cloud, and on-premises, without the complexity of managing a control plane.

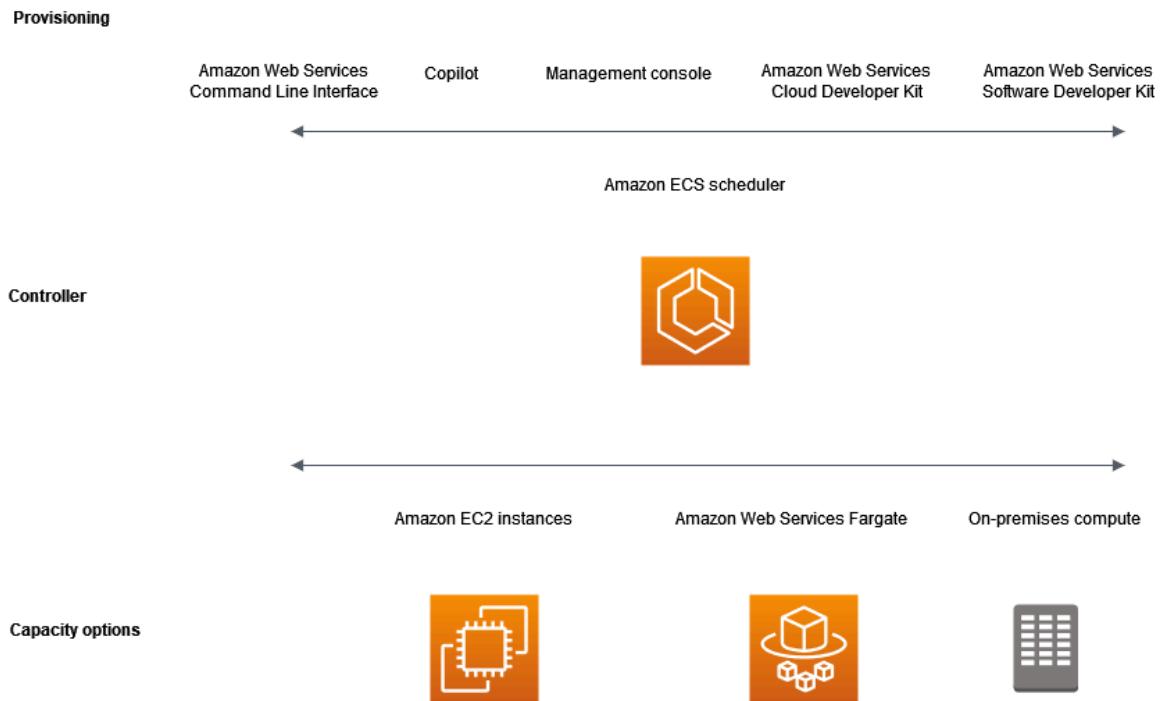
Amazon ECS terminology and components

There are three layers in Amazon ECS:

- Capacity - The infrastructure where your containers run
- Controller - Deploy and manage your applications that run on the containers
- Provisioning - The tools that you can use to interface with the scheduler to deploy and manage your applications and containers

The following diagram shows the Amazon ECS layers.

Amazon Elastic Container Service Layers



Amazon ECS capacity

Amazon ECS capacity is the infrastructure where your containers run. The following is an overview of the capacity options:

- Amazon EC2 instances in the AWS cloud
 - You choose the instance type, the number of instances, and manage the capacity.
- Serverless (AWS Fargate (Fargate)) in the AWS cloud

Fargate is a serverless, pay-as-you-go compute engine. With Fargate you don't need to manage servers, handle capacity planning, or isolate container workloads for security.

- On-premises virtual machines (VM) or servers

Amazon ECS Anywhere provides support for registering an external instance such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster.

The capacity can be located in any of the following AWS resources:

- Availability Zones
- Local Zones
- Wavelength Zones
- AWS Regions
- AWS Outposts

Amazon ECS controller

The Amazon ECS scheduler is the software that manages your applications.

Amazon ECS provisioning

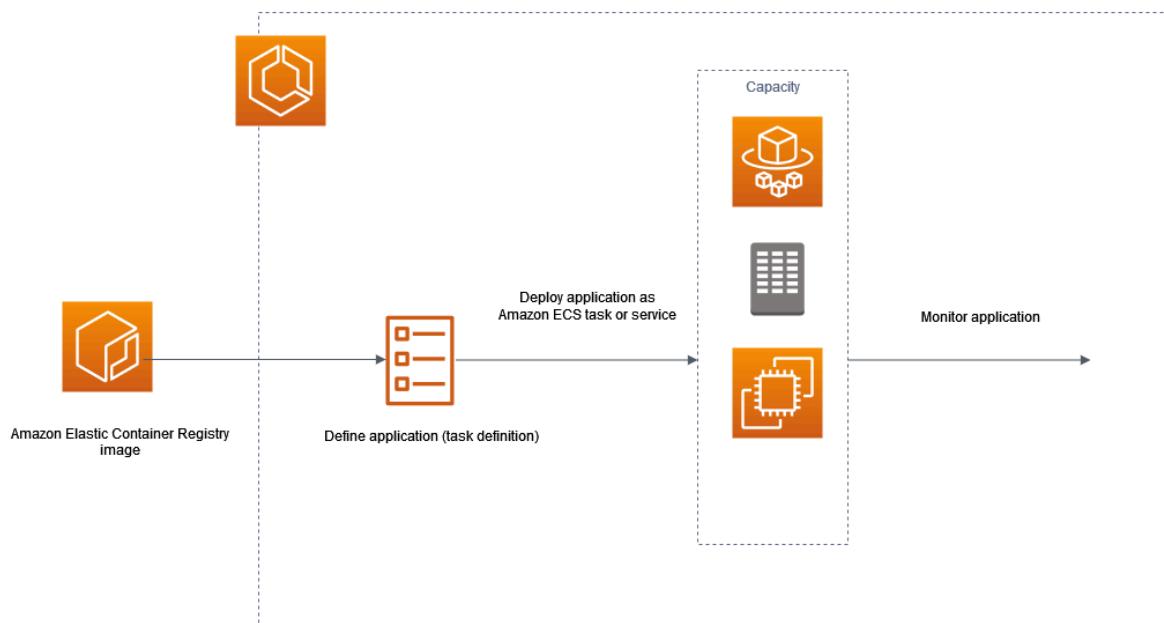
There are multiple options for provisioning Amazon ECS:

- **AWS Management Console** — Provides a web interface that you can use to access your Amazon ECS resources.
- **AWS Command Line Interface (AWS CLI)** — Provides commands for a broad set of AWS services, including Amazon ECS. It's supported on Windows, Mac, and Linux. For more information, see [AWS Command Line Interface](#).
- **AWS SDKs** — Provides language-specific APIs and takes care of many of the connection details. These include calculating signatures, handling request retries, and error handling. For more information, see [AWS SDKs](#).
- **Copilot** — Provides an open-source tool for developers to build, release, and operate production ready containerized applications on Amazon ECS. For more information, see [Copilot](#) on the GitHub website.
- **AWS CDK** — Provides an open-source software development framework that you can use to model and provision your cloud application resources using familiar programming languages. The AWS CDK provisions your resources in a safe, repeatable manner through AWS CloudFormation.

Application lifecycle

The following diagram shows the application lifecycle and how it works with the Amazon ECS components.

Amazon ECS Application Lifecycle



To deploy applications on Amazon ECS, your application components must be configured to run in *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from.

After you create and store your image, you create an Amazon ECS task definition. A *task definition* is a blueprint for your application. It is a text file in JSON format that describes the parameters and one or more containers that form your application. For example, you can use it to specify the image and parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application.

After you define your task definition, you deploy it as either a service or a task on your cluster. A *cluster* is a logical grouping of tasks or services that runs on the capacity infrastructure that is registered to a cluster.

A *task* is the instantiation of a task definition within a cluster. You can run a standalone task, or you can run a task as part of a service. You can use an Amazon ECS *service* to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

The *container agent* runs on each container instance within an Amazon ECS cluster. The agent sends information about the current running tasks and resource utilization of your containers to Amazon ECS. It starts and stops tasks whenever it receives a request from Amazon ECS.

After you deploy the task or service, you can use any of the following tools to monitor your deployment and application:

- CloudWatch

Amazon ECS features

The following are key features of Amazon ECS:

- Options to run your applications on Amazon EC2 instances, a serverless environment, or on-premises VMs.
- Integration with AWS Identity and Access Management (IAM). You can assign granular permissions for each of your containers. This allows for a high level of isolation when building your applications. In other words, you can launch your containers with the security and compliance levels that you've come to expect from AWS.
- AWS managed container orchestration with operational best practices built-in, and no control plane, nodes, or add-ons for you to manage. It natively integrates with both AWS and third-party tools to make it easier for teams to focus on building the applications, not the environment.
- Continuous integration and continuous deployment (CI/CD). This is a common process for microservice architectures that are based on Docker containers. You can create a CI/CD pipeline that takes the following actions:
 - Monitors changes to a source code repository
 - Builds a new Docker image from that source

- Pushes the image to an image repository such as Amazon ECR or Docker Hub
- Updates your Amazon ECS services to use the new image in your application
- Multiple options for a way to interconnect your applications.
 - Service Discovery - Integrates services with AWS Cloud Map namespaces to add entries (specifically, AWS Cloud Map service instances) to the namespace for each task in the Amazon ECS service. To connect, an app resolves these entries as DNS hostname records or uses the AWS Cloud Map API to get the IP address of the tasks.
 - Amazon ECS Service Connect - Define logical names for your service endpoints and use them in your client applications to connect to dependencies.
- Support for service discovery. This is a key component of most distributed systems and service-oriented architectures. With service discovery, your microservice components are automatically discovered as they're created and terminated on a given infrastructure.
- Monitoring and logging
 - Use Amazon CloudWatch to average and aggregate CPU and memory utilization of running tasks. Set alarms to indicate when you need to increase or decrease capacity.
 - Use AWS CloudTrail to record API calls from the management console, AWS SDKs, and AWS Command Line Interface.
 - Use AWS Config to monitor and track how resources were configured, how they relate to one another, and how the configurations and relationships change over time.

Pricing

Amazon ECS pricing depends on whether you use AWS Fargate or Amazon EC2 infrastructure to host your containerized workloads. When using Amazon ECS on AWS Outposts, the pricing follows the same model that's used when you use Amazon EC2 directly. For more information, see [Amazon ECS Pricing](#).

Amazon ECS and Fargate also offer Savings Plans that provide significant savings based on your AWS usage. For more information, see the [Savings Plans User Guide](#).

To view your bill, go to the **Billing and Cost Management Dashboard** in the [AWS Billing and Cost Management console](#). Your bill contains links to usage reports that provide additional details about your bill. To learn more about AWS account billing, see [AWS Account Billing](#).

If you have questions concerning AWS billing, accounts, and events, [contact AWS Support](#).

Trusted Advisor is a service that you can use to help optimize the costs, security, and performance of your AWS environment. For more information about Trusted Advisor, see [AWS Trusted Advisor](#).

Common use cases in Amazon ECS

Fargate is suitable for the following workloads:

- Large workloads that need to be optimized for low overhead
- Small workloads that have occasional burst
- Tiny workloads
- Batch workloads

EC2 is suitable for the following workloads:

- Workloads that require consistently high CPU core and memory usage
- Large workloads that need to be optimized for price
- Your applications need to access persistent storage
- You must directly manage your infrastructure

Additional resources

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

You can view the microservices reference architecture on GitHub. For more information, see [Deploying Microservices with Amazon ECS, AWS CloudFormation, and an Application Load Balancer](#).

The following resources outline how to implement continuous integration and deployment (CI/CD):

- [ECS Reference Architecture: Continuous Deployment](#): This reference architecture demonstrates how to achieve continuous deployment of an application to Amazon ECS using CodePipeline, CodeBuild, and AWS CloudFormation.
- [Continuous Delivery Pipeline for Amazon ECS Using Jenkins, GitHub, and Amazon ECR](#): This AWS labs repository helps you set up and configure a continuous delivery pipeline for Amazon ECS using Jenkins, GitHub, and Amazon ECR.

The [Managing Secrets for Amazon ECS Applications Using Parameter Store and IAM Roles for Tasks](#) post focuses on how to integrate the [IAM roles for tasks \(p. 631\)](#) functionality of Amazon ECS with the AWS Systems Manager Parameter Store. Parameter Store provides a centralized store to manage your configuration data, whether it's plaintext data such as database strings or secrets such as passwords, encrypted through AWS Key Management Service.

The following resources outline how to make your services discoverable:

- [Amazon ECS Service Connect Enabling Easy Communication Between Microservices](#): This post describes how to use the dynamic port mapping and path-based routing features of Elastic Load Balancing Application Load Balancers. This provides service discovery for a microservice architecture.
- [Amazon Elastic Container Service - Reference Architecture: Service Discovery](#): This Amazon ECS reference architecture provides service discovery to containers using CloudWatch Events, Lambda, and Route 53 private hosted zones.
- [Metrics and traces collection from Amazon ECS using AWS Distro for OpenTelemetry with dynamic service discovery](#): This post demonstrates how to employ a single instance of an ADOT Collector to collect X-Ray traces and Prometheus metrics from Amazon ECS services that were dynamically discovered using AWS Cloud Map.

Related services

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

AWS Identity and Access Management

AWS Identity and Access Management (IAM) is an access management service that helps you securely control access to AWS resources. You can use IAM to control who's authenticated (signed in) and authorized (has permissions) to view or perform specific actions on resources. In Amazon ECS,

you can use IAM to control access at the container instance level using IAM roles. You can also use it to control access at the task level using IAM task roles. For more information, see [Identity and Access Management for Amazon Elastic Container Service \(p. 587\)](#).

Amazon EC2 Auto Scaling

Auto Scaling is a service that sets up automatic scaling for your tasks. The scaling is based on user-defined policies, health status checks, and schedules. You can use Auto Scaling alongside a Fargate task within a service to scale in response to a number of metrics. Or, alternatively, you can use it with an EC2 task to scale the container instances within your cluster. For more information, see [Service auto scaling \(p. 498\)](#).

Elastic Load Balancing

The Elastic Load Balancing service automatically distributes incoming application traffic across the tasks in your Amazon ECS service. You can use it to achieve greater levels of fault tolerance in your applications. At the same time, you can use it to also provide the amount of load-balancing capacity that's required 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 \(p. 486\)](#).

Amazon Elastic Container Registry

Amazon ECR is a managed AWS Docker registry service that's 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. More specifically, it makes provisioning and updating resources more predictable. You can define clusters, task definitions, and services as entities in an AWS CloudFormation script. For more information, see [AWS CloudFormation Template Reference](#).

New Amazon ECS console

Welcome to the new and improved console experience—a quicker and easier way to deploy containerized applications, configure load balancing, networking, monitoring, and gives you the new workflows for the effective operations and troubleshooting.

While most of the classic console functionality is available in the new console, it is still a work in progress. We also want to give customers time learn and migrate to the new console. Therefore the classic console will remain available to opt-in and you can toggle back to the classic console for the duration of your login session.

Configuration which is not available in the new console

We're working continuously to improve the experience. The sections below describe when you need to use the classic console. There will be many more functionality added to the new console.

First run wizard

Although the first run wizard is not available, we have provided instructions on how to get started using the new console. For more information, see [Getting started \(p. 10\)](#).

Clusters

You can use the AWS CLI to configure the following cluster parameters:

For information about how to create a cluster using the AWS CLI, see [create-cluster](#) in the *AWS Command Line Interface Reference*.

- **Spot Instances** - For the EC2 launch type, using Spot Instances for your cluster Auto Scaling group
- **Creating a new VPC on cluster creation** - You must use existing VPCs and subnets when you create a cluster

Task definitions

You can use the JSON editor in the new console, or the AWS CLI to configure the following task definition parameters:

For information about how to use the JSON editor, see [Creating a task definition using the console \(p. 125\)](#).

For information about how to register the task definition using the AWS CLI, see [register-task-definition](#) in the *AWS Command Line Interface Reference*.

- **AWS App Mesh integration**
- **EXTERNAL lanuch type** - for Amazon ECS Anywhere
- **FireLens customization**
- **Task placement constraints**
- **Update a service with the new task definition revision**

Tasks

You can use the EventBridge Scheduler console or the AWS CLI to configure scheduled tasks:

For information about how to use the EventBridge Scheduler console, see [Scheduled tasks \(p. 445\)](#).

- **Scheduling tasks** - You cannot create scheduled tasks.
- **Run more like this**

Services

You must use AWS CloudFormation or the AWS Command Line Interface to deploy a service that uses any of the following parameters:

- **Blue/green deployments**
- **Service Discovery** - You can only view your Service Discovery configuration.
- **Tracking policy with a custom metric** - You must use AWS CloudFormation or the AWS Command Line Interface to deploy the service
- **Update Service** - You cannot update the awsvpc network configuration and the health check grace period.

For information about how to create a service using the AWS CLI, see [create-service](#) in the *AWS Command Line Interface Reference*.

For information about how to create a service using AWS CloudFormation, see [AWS::ECS::Service](#) in the *AWS CloudFormation User Guide*.

Container instances

You can use the AWS CLI to view or edit custom metadata for your container instance.

For more information, see [Adding an attribute using the classic console \(p. 439\)](#) and [Filtering by attribute using the console \(p. 440\)](#).

- **Instance attributes** - You cannot view or edit custom metadata for your container instance

Getting started with Amazon ECS

The following guides provide an introduction to the tools available to access Amazon ECS and introductory step by step procedures to run containers. Docker basics takes you through the basic steps to create a Docker container image and upload it to an Amazon ECR private repository. The getting started guides walk you through using the AWS Copilot command line interface and the AWS Management Console to complete the common tasks to run your containers on Amazon ECS and AWS Fargate.

Contents

- [Set up to use Amazon ECS \(p. 10\)](#)
- [Creating a container image for use on Amazon ECS \(p. 15\)](#)
- [Getting started with the console using Linux containers on AWS Fargate \(p. 19\)](#)
- [Getting started with the console using Windows containers on AWS Fargate \(p. 23\)](#)
- [Getting started with the console using Windows on Amazon EC2 \(p. 26\)](#)

Set up to use 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 Amazon ECS cluster.

Complete the following tasks to get set up for Amazon ECS.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

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

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, [assign administrative access to an administrative user](#), and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center (successor to AWS Single Sign-On).

For instructions, see [Getting started](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Create the credentials to connect to your EC2 instance

For Amazon ECS, a key pair is only needed if you intend on using the EC2 launch type.

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

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

To create a key pair

- Use the Amazon EC2 console to create a key pair. For more information about creating a key pair, see [Create a key pair](#) in the *Amazon EC2 User Guide for Linux Instances*.

For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your instance using your key pair

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

To prepare to connect to a Linux instance from Windows using PuTTY

1. Download and install PuTTY from <http://www.chiark.greenend.org.uk/~sgtatham/putty/>. Be sure to install the entire suite.
2. Start PuTTYgen (for example, from the **Start** menu, choose **All Programs > PuTTY > PuTTYgen**).
3. Under **Type of key to generate**, choose **RSA**.
4. Choose **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.
5. Select the private key file that you created in the previous procedure and choose **Open**. Choose **OK** to dismiss the confirmation dialog box.
6. Choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the .ppk file extension.

Create a virtual private cloud

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

If you have a default VPC, you can skip this section and move to the next task, [Create a security group \(p. 12\)](#). 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.

For information about how to create a VPC, see [Create a VPC only](#) in the *Amazon VPC User Guide*, and use the following table to determine what options to select.

Option	Value
Resources to create	VPC only
Name	Optionally provide a name for your VPC.
IPv4 CIDR block	IPv4 CIDR manual input The CIDR block size must have a size between /16 and /28.
IPv6 CIDR block	No IPv6 CIDR block
Tenancy	Default

For more information about Amazon VPC, see [What is Amazon VPC?](#) in the *Amazon VPC User Guide*.

Create a security group

Security groups act as a firewall for associated container instances, controlling both inbound and outbound traffic at the container instance level. You can add rules to a security group that enable you to connect to your container instance from your IP address using SSH. You can also add rules that allow

inbound and outbound HTTP and HTTPS access from anywhere. Add any rules to open ports that are required by your tasks. Container instances require external network access to communicate with the Amazon ECS service endpoint.

Note

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

If you plan to launch container instances in multiple Regions, you need to create a security group in each Region. For more information, see [Regions and Availability Zones](#) in the *Amazon EC2 User Guide for Linux Instances*.

Tip

You need the public IP address of your local computer, which you can get using a service.

For example, we provide the following service: <http://checkip.amazonaws.com/> or <https://checkip.amazonaws.com/>. To locate another service that provides your IP address, use the search phrase "what is my IP address." If you are connecting through an internet service provider (ISP) or from behind a firewall without a static IP address, you must find out the range of IP addresses used by client computers.

For information about how to create a security group, see [Create a security group](#) in the *Amazon EC2 User Guide for Linux Instances* and use the following table to determine what options to select.

Option	Value
Region	The same Region in which you created your key pair.
Name	A name that is easy for you to remember, such as <code>ecs-instances-default-cluster</code> .
VPC	The default VPC (marked with "(default)". Note If your account supports Amazon EC2 Classic, select the VPC that you created in the previous task.

For information about the outbound rules to add for your use cases, see [Security group rules for different use cases](#) in the *Amazon EC2 User Guide for Linux Instances*.

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

Add the following three inbound rules to your security group. For information about how to create a security group, see [Add rules to your security group](#) in the *Amazon EC2 User Guide for Linux Instances*.

Option	Value
HTTP rule	Type: HTTP

Option	Value
	<p>Source: Anywhere (<code>0.0.0.0/0</code>)</p> <p>This option automatically adds the <code>0.0.0.0/0</code> IPv4 CIDR block as the source. This is acceptable for a short time in a test environment, but it's unsafe in production environments. In production, authorize only a specific IP address or range of addresses to access your instance.</p>
HTTPS rule	<p>Type: HTTPS</p> <p>Source: Anywhere (<code>0.0.0.0/0</code>)</p> <p>This is acceptable for a short time in a test environment, but it's unsafe in production environments. In production, authorize only a specific IP address or range of addresses to access your instance.</p>
SSH rule	<p>Type: SSH</p> <p>Source: Custom, specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix <code>/32</code>. For example, if your IP address is <code>203.0.113.25</code>, specify <code>203.0.113.25/32</code>. If your company allocates addresses from a range, specify the entire range, such as <code>203.0.113.0/24</code>.</p> <p>Important For security reasons, we don't recommend that you allow SSH access from all IP addresses (<code>0.0.0.0/0</code>) to your instance, except for testing purposes and only for a short time.</p>

Install the AWS CLI

The AWS Management Console can be used to manage all operations manually with Amazon ECS. However, you can install the AWS CLI on your local desktop or a developer box so that you can 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](#).

Creating a container image for use on Amazon ECS

Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Deploying Docker containers on Amazon ECS](#).

The purpose of the steps outlined here is to walk you through creating your first Docker image and pushing that image to Amazon ECR, which is a container registry, for use in your Amazon ECS task definitions. This walkthrough assumes that you 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](#).

Important

AWS and Docker have collaborated to make a simplified developer experience that allows you to deploy and manage containers on Amazon ECS directly using Docker tools. You can now build and test your containers locally using Docker Desktop and Docker Compose, and then deploy them to Amazon ECS on Fargate. To get started with the Amazon ECS and Docker integration, download Docker Desktop and optionally sign up for a Docker ID. For more information, see [Docker Desktop](#) and [Docker ID signup](#).

Prerequisites

Before you begin, ensure the following prerequisites are met.

- Ensure you have completed the Amazon ECR setup steps. For more information, see [Setting up for Amazon ECR](#) in the [Amazon Elastic Container Registry User Guide](#).
- Your user has the required IAM permissions to access and use the Amazon ECR service. For more information, see [Amazon ECR managed policies](#).
- You have Docker installed. For Docker installation steps for Amazon Linux 2, see [Installing Docker on Amazon Linux 2 \(p. 15\)](#). For all other operating systems, see the Docker documentation at [Docker Desktop overview](#).
- You have the AWS CLI installed and configured. For more information, see [Installing the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

If you don't have or need a local development environment and you prefer to use an Amazon EC2 instance to use Docker, we provide the following steps to launch an Amazon EC2 instance using Amazon Linux 2 and install Docker Engine and the Docker CLI.

Installing Docker on Amazon Linux 2

Docker Desktop is an easy-to-install application for your Mac or Windows environment that you can use to build and share containerized applications and microservices. Docker Desktop includes Docker Engine,

the Docker CLI client, Docker Compose, and other tools that are helpful when using Docker with Amazon ECS. For more information about how to install Docker Desktop on your preferred operating system, see [Docker Desktop overview](#).

To install Docker on an Amazon EC2 instance

1. Launch an instance with the Amazon Linux 2 AMI. For more information, see [Launching an instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
2. Connect to your instance using SSH. For more information, see [Connect to your Linux instance using SSH](#) 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 Engine package.

```
sudo amazon-linux-extras install docker
```

Important

This step assumes you are using the Amazon Linux 2 AMI for your instance. For all other operating systems, see [Docker Desktop overview](#).

5. Start the Docker service.

```
sudo service docker start
```

(Optional) To ensure that the Docker daemon starts after each system reboot, run the following command:

```
sudo systemctl enable docker
```

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 you 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 Amazon EC2 instance, and then push the image to the Amazon ECR container registry so you can use it in an Amazon 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:18.04

# Install dependencies
RUN apt-get update && \
    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 && \
    echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh && \
    echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

This Dockerfile uses the Ubuntu 18.04 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	241MB

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 -t -i -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**.

Push your image to Amazon Elastic Container Registry

Amazon ECR is a managed AWS Docker registry service. You can use the Docker CLI to push, pull, and manage images in your Amazon ECR repositories. For Amazon ECR product details, featured customer case studies, and FAQs, see the [Amazon Elastic Container Registry product detail pages](#).

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.

Substitute `region`, with your AWS Region, for example, `us-east-1`.

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

Output:

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

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

```
docker tag hello-world aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

3. Run the **aws ecr get-login-password** command. Specify the registry URI you want to authenticate to. For more information, see [Registry Authentication](#) in the *Amazon Elastic Container Registry User Guide*.

```
docker login -u AWS -p $(aws ecr get-login-password --region REGION) aws_account_id.dkr.ecr.REGION.amazonaws.com
```

Output:

```
Login Succeeded
```

Important

If you receive an error, install or upgrade to 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*.

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

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

Clean up

To continue on with creating an Amazon ECS task definition and launching a task with your container image, skip to the [Next steps \(p. 19\)](#). When you are done experimenting with your Amazon ECR image, you can delete the repository so you are not charged for image storage.

```
aws ecr delete-repository --repository-name hello-repository --region region --force
```

Next steps

Your task definitions require a task execution role. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

After you have created and pushed your container image to Amazon ECR, you should consider the following next steps.

- [the section called “Using the console with Linux containers on AWS Fargate” \(p. 19\)](#)
- [the section called “Using the console with Windows containers on AWS Fargate” \(p. 23\)](#)
- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI \(p. 670\)](#)

Getting started with the console using Linux containers on AWS Fargate

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For a broad overview on Amazon ECS on Fargate, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks in the Regions where Amazon ECS supports AWS Fargate.

Complete the following steps to get started with Amazon ECS on AWS Fargate.

Prerequisites

Before you begin, complete the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the `AdministratorAccess` or [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.

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

- Your user has administrator access. For more information, see [Set up to use Amazon ECS \(p. 10\)](#).
- 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 \(p. 626\)](#).

Important

The security group you select when creating a service with your task definition must have port 80 open for inbound traffic. Add the following inbound rule to your security group. For information about how to create a security group, see [Add rules to your security group](#) in the *Amazon EC2 User Guide for Linux Instances*.

- Type: HTTP
- Protocol: TCP
- Port range: 80
- Source: Anywhere (`0.0.0.0/0`)

Step 1: Create the cluster

Create a cluster that uses the default VPC.

Before you begin, assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.

6. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

[Add a tag] Choose **Add tag** and do the following:

- For **Key**, enter the key name.
- For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

8. Choose **Create**.

Step 2: 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. In the navigation pane, choose **Task Definitions**.
2. Choose **Create new Task Definition**, **Create new revision with JSON**.
3. Copy and paste the following example task definition into the box and then choose **Save**.

```
{  
    "family": "sample-fargate",  
    "networkMode": "awsvpc",  
    "containerDefinitions": [  
        {  
            "name": "fargate-app",  
            "image": "public.ecr.aws/docker/library/httpd:latest",  
            "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"  
}
```

4. Choose **Create**.

Step 3: Create the service

Create a service using the task definition.

1. In the navigation pane, choose **Clusters**, and then select the cluster you created in [Step 1: Create the cluster \(p. 20\)](#).
2. From the **Services** tab, choose **Create**.
3. Under **Deployment configuration**, specify how your application is deployed.

- a. For **Task definition**, choose the task definition you created in [Step 2: Create a task definition \(p. 21\)](#).
- b. For **Service name**, enter a name for your service.
- c. For **Desired tasks**, enter **1**.
4. Under **Networking**, you can create a new security group or choose an existing security group for your task. Ensure that the security group you use has the inbound rule listed under [Prerequisites \(p. 20\)](#).
5. Choose **Create**.

Step 4: View your service

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.
4. In the **Services** tab, under **Service name**, choose the service you created in [Step 3: Create the service \(p. 21\)](#).
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**. The screenshot below is the expected output.



Step 5: Clean up

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

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

1. In the navigation pane, choose **Clusters**.
2. On the **Clusters** page, select the cluster cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**.

Wait until the service is deleted.

6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete *cluster-name***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Getting started with the console using Windows containers on AWS Fargate

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For a broad overview on Amazon ECS on Fargate, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks in the Regions where Amazon ECS supports AWS Fargate.

Complete the following steps to get started with Amazon ECS on AWS Fargate.

Prerequisites

Before you begin, complete the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.

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

- Your user has administrator access. For more information, see [Set up to use Amazon ECS \(p. 10\)](#).
- 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 \(p. 626\)](#).

Important

The security group you select when creating a service with your task definition must have port 80 open for inbound traffic. Add the following inbound rule to your security group. For information about how to create a security group, see [Add rules to your security group](#) in the [Amazon EC2 User Guide for Linux Instances](#).

- Type: HTTP
- Protocol: TCP
- Port range: 80
- Source: Anywhere ($0\cdot0\cdot0\cdot0/0$)

Step 1: Create a cluster

You can create a new cluster called **windows** that uses the default VPC.

To create a cluster with the AWS Management Console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter **windows**.
6. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

[Add a tag] Choose **Add tag** and do the following:

- For **Key**, enter the key name.
- For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

8. Choose **Create**.

Step 2: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple webpage on port 8080 of a container instance with the `mcr.microsoft.com/windows/servercore/iis` container image.

To register the sample task definition with the AWS Management Console

1. In the navigation pane, choose **Task definitions**.
2. Choose **Create new task definition**, **Create new task definition with JSON**.
3. Copy and paste the following example task definition into the box and then choose **Save**.

```
{  
    "containerDefinitions": [  
        {  
            "command": [  
                "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value  
                '<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>'; C:\\ServiceMonitor.exe  
                w3svc"  
            ],  
            "entryPoint": [  
                "powershell",  
                "-Command"  
            ],  
            "essential": true,  
            "cpu": 2048,  
            "memory": 4096,  
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-  
                ltsc2019",  
            "name": "sample_windows_app",  
            "portMappings": [  
                {  
                    "hostPort": 80,  
                    "containerPort": 80,  
                    "protocol": "tcp"  
                }  
            ]  
        },  
    ]  
}
```

```
"memory": "4096",
"cpu": "2048",
"networkMode": "awsvpc",
"family": "windows-simple-iis-2019-core",
"executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
"runtimePlatform": {
    "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
},
"requiresCompatibilities": [
    "FARGATE"
]
}
```

4. Verify your information and choose **Create**.

Step 3: Create a service with your task definition

After you have registered your task definition, you can place tasks in your cluster with it. The following procedure creates a service with your task definition and places one task in your cluster.

To create a service from your task definition with the console

1. In the navigation pane, choose **Clusters**, and then select the cluster you created in [Step 1: Create a cluster \(p. 23\)](#).
2. From the **Services** tab, choose **Create**.
3. Under **Deployment configuration**, specify how your application is deployed.
 - a. For **Task definition**, choose the task definition you created in [Step 2: Register a Windows task definition \(p. 24\)](#).
 - b. For **Service name**, enter a name for your service.
 - c. For **Desired tasks**, enter 1.
4. Under **Networking**, you can create a security group or choose an existing group. Ensure that the security group you use has the inbound rule listed under [Prerequisites \(p. 23\)](#).
5. Choose **Create**.

Step 4: View your service

After your service has launched a task into your cluster, you can view the service and open the IIS test page in a browser to verify that the container is running.

Note

It can take up to 15 minutes for your container instance to download and extract the Windows container base layers.

To view your service

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.
4. In the **Services** tab, under **Service name**, choose the service you created in [Step 3: Create a service with your task definition \(p. 25\)](#).
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**. The screenshot below is the expected output.



Step 5: Clean Up

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

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

1. In the navigation pane, choose **Clusters**.
2. On the **Clusters** page, select the cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**.
Wait until the service is deleted.
6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete *cluster-name***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Getting started with the console using Windows on Amazon EC2

Amazon Elastic Container Service (Amazon ECS) is a fast and highly scalable container management service that makes it easy to launch and manage your containers. For a broad overview on Amazon ECS, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS using the EC2 launch type by registering a task definition, creating a cluster, and creating a service in the classic console.

Complete the following steps to get started with Amazon ECS using the EC2 launch type.

Prerequisites

Before you begin, complete the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the `AdministratorAccess` or [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.

Step 1: Create a cluster

An Amazon ECS cluster is a logical grouping of tasks, services, and container instances.

The following steps walk you through creating a cluster with one Amazon EC2 instance registered to it which will enable us to run a task on it. If a specific field is not mentioned, leave the default console values.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.

6. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
 - To remove a subnet, under **Subnets**, choose X for each subnet that you want to remove.
 - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
7. To add Amazon EC2 instances to your cluster, expand **Infrastructure**, and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider:
 - a. To use an existing Auto Scaling group, from **Auto Scaling group (ASG)**, select the group.
 - b. To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:
 - For **Operating system/Architecture**, choose the Amazon ECS-optimized AMI for the Auto Scaling group instances.
 - For **EC2 instance type**, choose the instance type for your workloads. For more information about the different instance types, see [Amazon EC2 Instances](#).

Managed scaling works best if your Auto Scaling group uses the same or similar instance types.

 - For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.
 - For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see [Amazon EC2 Pricing](#).
8. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
9. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

[Add a tag] Choose **Add tag** and do the following:

 - For **Key**, enter the key name.
 - For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

10. Choose **Create**.

Step 2: Register a task definition

To register the sample task definition with the AWS Management Console

1. In the navigation pane, choose **Task Definitions**.
2. Choose **Create new task definition**, **Create new task definition with JSON**.
3. Copy and paste the following example task definition into the box, and then choose **Save**.

```
{  
    "containerDefinitions": [  
        {  
            "command": [  
                "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value  
                '<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>'; C:\\ServiceMonitor.exe  
                w3svc"  
            ],  
            "entryPoint": [  
                "powershell",  
                "-Command"  
            ],  
            "essential": true,  
            "cpu": 2048,  
            "memory": 4096,  
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-  
                ltsc2019",  
            "name": "sample_windows_app",  
            "portMappings": [  
                {  
                    "hostPort": 443,  
                    "containerPort": 80,  
                    "protocol": "tcp"  
                }  
            ]  
        },  
        {  
            "memory": "4096",  
            "cpu": "2048",  
            "family": "windows-simple-iis-2019-core",  
            "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",  
            "runtimePlatform": {  
                "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"  
            },  
            "requiresCompatibilities": [  
                "EC2"  
            ]  
        }  
    ]  
}
```

4. Verify your information and choose **Create**.

Step 3: Create a Service

An Amazon ECS service helps you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. 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 in order to maintain the desired number of tasks in the service. For more information on services, see [Amazon ECS services \(p. 453\)](#).

To create a service

1. In the navigation pane, choose **Clusters**.
2. Select the cluster you created in [Step 1: Create a cluster \(p. 27\)](#).
3. On the **Services** tab, choose **Create**.
4. In the **Environment** section, do the following:
 - a. For **Compute options**, choose Launch type.
 - b. For **Launch type**, select **EC2**
5. In the **Deployment configuration** section, do the following:
 - a. For **Family**, choose the task definition you created in [Step 2: Register a task definition \(p. 28\)](#).
 - b. For **Service name**, enter a name for your service.
 - c. For **Desired tasks**, enter 1.
6. Review the options and choose **Create**.
7. Choose **View service** to review your service.

Step 4: View your Service

The service is a web-based application so you can view its containers with a web browser.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.
4. In the **Services** tab, under **Service name**, choose the service you created in [Step 3: Create a Service \(p. 29\)](#).
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**. The screenshot below is the expected output.



Step 5: Clean Up

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

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

1. In the navigation pane, choose **Clusters**.
2. On the **Clusters** page, select the cluster cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**.
Wait until the service is deleted.
6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete *cluster-name***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Amazon ECS developer tools overview

Whether you are part of a large enterprise or a startup, Amazon ECS offers a variety of tools that can help you to get your containers up and running quickly, regardless of your level of expertise. You can work with Amazon ECS in the following ways.

- Learn about, develop, manage and visualize your container applications and services using the [AWS Management Console \(p. 31\)](#).
- Perform specific actions to Amazon ECS resources with automated deployments through programming or scripts using the [AWS Command Line Interface \(p. 31\)](#), [AWS SDKs \(p. 34\)](#) or the ECS API.
- Define and manage all AWS resources in your environment with automated deployment using [AWS CloudFormation \(p. 32\)](#).
- Use the complete [AWS Copilot CLI \(p. 32\)](#) end-to-end developer workflow to create, release, and operate container applications that comply with AWS best practices for infrastructure.
- Using your preferred programming language, define infrastructure or architecture as code with the [AWS CDK \(p. 32\)](#).
- Containerize applications that are hosted on premises or on Amazon EC2 instances or both by using the [AWS App2Container \(p. 33\)](#) integrated portability and tooling ecosystem for containers.
- Deploy an application to Amazon ECS or test local containers with containers running in Amazon ECS using the Docker Compose file format with the [Amazon ECS CLI \(p. 33\)](#).
- Launch containers from [Docker Desktop integration with Amazon ECS \(p. 33\)](#) using Amazon ECS in Docker Desktop.

AWS Management Console

The AWS Management Console is a browser-based interface for managing Amazon ECS resources. The console provides a visual overview of the service, making it easy to explore Amazon ECS features and functions without needing to use additional tools. Many related tutorials and walkthroughs are available that can guide you through use of the console.

For a tutorial that guides you through the console, see [Getting started with Amazon ECS \(p. 10\)](#).

When starting out, many customers prefer using the console because it provides instant visual feedback on whether the actions they take succeed. AWS customers that are familiar with the AWS Management Console, can easily manage related resources such as load balancers and Amazon EC2 instances.

Start with the AWS Management Console.

AWS Command Line Interface

The AWS Command Line Interface (AWS CLI) is a unified tool that you can use to manage your AWS services. With this one tool alone, you can both control multiple AWS services and automate these services through scripts. The Amazon ECS commands in the AWS CLI are a reflection of the Amazon ECS API.

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](#).

The AWS CLI is suitable for customers who prefer and are used to scripting and interfacing with a command line tool and know exactly which actions they want to perform on their Amazon ECS resources. The AWS CLI is also helpful to customers who want to familiarize themselves with the Amazon ECS APIs. Customers can use the AWS CLI to perform a number of operations on Amazon ECS resources, including Create, Read, Update, and Delete operations, directly from the command line interface.

Use the AWS CLI if you are or want to become familiar with the Amazon ECS APIs and corresponding CLI commands and want to write automated scripts and perform specific actions on Amazon ECS resources.

AWS CloudFormation

[AWS CloudFormation](#) and [Terraform](#) for Amazon ECS both provide powerful ways for you to define your infrastructure as code. You can easily track which version of your template or AWS CloudFormation stack is running at any time and rollback to a previous version if needed. You can perform infrastructure and application deployments in the same automated fashion. This flexibility and automation is what makes AWS CloudFormation and Terraform two popular formats for deploying workloads to Amazon ECS from continuous delivery pipelines.

For more information about AWS CloudFormation, see [Creating Amazon ECS resources with AWS CloudFormation \(p. 52\)](#).

Use AWS CloudFormation or Terraform if you want to automate infrastructure deployments and applications on Amazon ECS and explicitly define and manage all of the AWS resources in your environment.

AWS Copilot CLI

The AWS Copilot CLI (command line interface) is a comprehensive tool that allows customers to deploy and operate applications packaged in containers and environments on Amazon ECS directly from their source code. When using AWS Copilot you can perform these operations without understanding AWS and Amazon ECS elements such as Application Load Balancers, public subnets, tasks, services, and clusters. AWS Copilot creates AWS resources on your behalf from opinionated service patterns, such as a load balanced web service or backend service, providing an immediate production environment for containerized applications. You can deploy through an AWS CodePipeline pipeline across multiple environments, accounts, or Regions, all of which can be managed within the CLI. By using AWS Copilot you can also perform operator tasks, such as viewing logs and the health of your service. AWS Copilot is an all-in-one tool that helps you more easily manage your cloud resources so that you can focus on developing and managing your applications.

For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

Use the AWS Copilot complete end-to-end developer workflow to create, release, and operate container applications that comply with AWS best practices for infrastructure.

AWS CDK

The AWS Cloud Development Kit (AWS CDK) is an open source software development framework that you can use to model and provision your cloud application resources using familiar programming languages. AWS CDK provisions your resources in a safe, repeatable manner through AWS

CloudFormation. Using the CDK, customers can generate their environment with fewer lines of code using the same language they used to build their application. Amazon ECS provides a module in the CDK that is named `ecs-patterns`, which creates common architectures. An available pattern is `ApplicationLoadBalancedFargateService()`. This pattern creates a cluster, task definition, and additional resources to run a load balanced Amazon ECS service on AWS Fargate.

For more information, see [Getting started with Amazon ECS using the AWS CDK \(p. 45\)](#).

Use the AWS CDK if you want to define infrastructure or architecture as code in your preferred programming language. For example, you can use the same language that you use to write your applications.

AWS App2Container

Sometimes enterprise customers might already have applications that are hosted on premises or on EC2 instances or both. They are interested in the portability and tooling ecosystem of containers specifically on Amazon ECS, and need to containerize first. AWS App2Container allows you to do just that. App2Container (A2C) is a command line tool for modernizing .NET and Java applications into containerized applications. A2C analyzes and builds an inventory of all applications running in virtual machines, on premises or in the cloud. After you select the application you want to containerize, A2C packages the application artifact and identified dependencies into container images. It then configures the network ports and generates the Amazon ECS task. Last, it creates a CloudFormation template that you can deploy or modify if needed.

For more information, see [Getting started with AWS App2Container](#).

Use App2Container if you have applications that are hosted on premises or on Amazon EC2 instances or both.

Amazon ECS CLI

The Amazon ECS CLI allows you to run your applications on Amazon ECS and AWS Fargate using the Docker Compose file format. You can quickly provision resources, push and pull images using [Amazon ECR](#), and monitor running applications on Amazon ECS or AWS Fargate. You can also test containers running locally along with containers in the cloud within the CLI.

For more information, see [Using the Amazon ECS command line interface \(p. 58\)](#).

Use the ECS CLI if you have a Compose application and want to deploy it to Amazon ECS, or test local containers with containers running in Amazon ECS in the cloud.

Docker Desktop integration with Amazon ECS

AWS and Docker have collaborated to make a simplified developer experience that you can use to deploy and manage containers on Amazon ECS directly using Docker tools. You can now build and test your containers locally using Docker Desktop and Docker Compose, and then deploy them to Amazon ECS on Fargate. To get started with the Amazon ECS and Docker integration, download Docker Desktop and optionally sign up for a Docker ID. For more information, see [Docker Desktop](#) and [Docker ID signup](#).

Beginners to containers often start learning about containers by using Docker tools such as the Docker CLI and Docker Compose. This makes using the Docker Compose CLI plugin for Amazon ECS a natural next step in running containers on AWS after testing locally. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Deploying Docker containers on Amazon ECS](#).

You can take advantage of additional Amazon ECS features, such as service discovery, load balancing and other AWS resources for use with their applications with Docker Desktop.

You can also download the Docker Compose CLI plugin for Amazon ECS directly from GitHub. For more information, see [Docker Compose CLI plugin for Amazon ECS](#) on GitHub.

AWS SDKs

You can also use AWS SDKs to manage Amazon ECS resources and operations from a variety of programming languages. The SDKs provide modules to help take care of tasks, including tasks in the following list.

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

For more information about the available SDKs, see [Tools for Amazon Web Services](#).

Summary

With the many options to choose from, you can choose the options that are best suited to you. Consider the following options.

- If you are visually oriented, you can visually create and operate containers using the AWS Management Console.
- If you prefer CLIs, consider using AWS Copilot or the AWS CLI. Alternatively, if you prefer the Docker ecosystem, you can take advantage of the functionality of ECS from within the Docker CLI to deploy to AWS. After these resources are deployed, you can continue managing them through the CLI or visually through the Console.
- If you are a developer, you can use the AWS CDK to define your infrastructure in the same language as your application. You can use the CDK and AWS Copilot to export to CloudFormation templates where you can change granular settings, add other AWS resources, and automate deployments through scripting or a CI/CD pipeline such as AWS CodePipeline.

The AWS CLI, SDKs, or ECS API are useful tools for automating actions on ECS resources, making them ideal for deployment. To deploy applications using AWS CloudFormation you can use a variety of programming languages or a simple text file to model and provision all the resources needed for your applications. You can then deploy your application across multiple Regions and accounts in an automated and secure manner. For example, you can define your ECS cluster, services, task definitions, or capacity providers, as code in a file and deploy through the AWS CLI CloudFormation commands.

To perform operations tasks, you can view and manage resources programmatically using the AWS CLI, SDK, or ECS API. Commands like `describe-tasks` or `list-services` display the latest metadata or a list of all resources. Similar to deployments, customers can write an automation that includes commands such as `update-service` to provide corrective action upon the detection of a resource that has stopped unexpectedly. You can also operate your services using AWS Copilot. Commands like `copilot svc logs` or `copilot app show` provide details about each of your microservices, or about your application as a whole.

Customers can use any of the available tooling mentioned in this document and use them in variety of combinations. ECS tooling offers various paths to graduate from certain tools to use others that fit your

changing needs. For example, you can opt for more granular control over resources or more automation as needed. ECS also offers a large range of tools for a wide range of needs and levels of expertise.

Using the AWS Copilot command line interface

The AWS Copilot command line interface (CLI) commands simplify building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. The AWS Copilot CLI aligns with developer workflows that support modern application best practices: from using infrastructure as code to creating a CI/CD pipeline provisioned on behalf of a user. Use the AWS Copilot CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

AWS Copilot currently supports Linux, macOS, and Windows systems. For more information about the latest version of the AWS Copilot CLI, see [Releases](#).

Note

The source code for the AWS Copilot CLI is available on [GitHub](#). The latest CLI documentation is available on the AWS Copilot [website](#). We recommend that you submit issues and pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of AWS Copilot code. Report issues with AWS Copilot by connecting with us on [Gitter](#) or [GitHub](#) where you can open issues, provide feedback, and report bugs.

Installing the AWS Copilot CLI

The AWS Copilot CLI can be installed on Linux or macOS systems either by using Homebrew or by manually downloading the binary. Use the following steps with your preferred installation method.

Installing the AWS Copilot CLI using Homebrew

The following command is used to install the AWS Copilot CLI on your macOS or Linux system using Homebrew. Before installation, you should have Homebrew installed. For more information, see [Homebrew](#).

```
brew install aws/tap/copilot-cli
```

Manually installing the AWS Copilot CLI

As an alternative to Homebrew, you can manually install the AWS Copilot CLI on your macOS or Linux system. Use the following command for your operating system to download the binary, apply execute permissions to it, and then verify it works by listing the help menu.

macOS

For macOS:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/latest/download/copilot-darwin \
&& sudo chmod +x /usr/local/bin/copilot \
&& copilot --help
```

For macOS ARM systems:

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-darwin-arm64.asc \
&& sudo chmod +x /usr/local/bin/copilot \
```

```
&& copilot --help
```

Linux

For Linux x86 (64-bit) systems:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux \
&& sudo chmod +x /usr/local/bin/copilot \
&& copilot --help
```

For Linux ARM systems:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux-arm64 \
&& sudo chmod +x /usr/local/bin/copilot \
&& copilot --help
```

Windows

Using Powershell, run the following command:

```
New-Item -Path 'C:\copilot' -ItemType directory; ` 
Invoke-WebRequest -OutFile 'C:\copilot\copilot.exe' https://github.com/aws/copilot-cli/releases/latest/download/copilot-windows.exe
```

(Optional) Verify the manually installed AWS Copilot CLI using PGP signatures

The AWS Copilot CLI executables are cryptographically signed using PGP signatures. The PGP signatures can be used to verify the validity of the AWS Copilot CLI executable. Use the following steps to verify the signatures using the GnuPG tool.

1. Download and install GnuPG. For more information, see the [GnuPG website](#).

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
```

Linux

Install gpg using the package manager on your flavor of Linux.

Windows

Download the Windows simple installer from the GnuPG website and install as an Administrator. After you install GnuPG, close and reopen the Administrator PowerShell.

For more information, see [GnuPG Download](#).

2. Verify the GnuPG path is added to your environment path.

macOS

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
PATH=$PATH:<path to GnuPG executable files>
```

Linux

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
export PATH=$PATH:<path to GnuPG executable files>
```

Windows

```
Write-Output $Env:PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
$Env:PATH += "<path to GnuPG executable files>"
```

3. Create a local plain text file.

macOS

On the terminal, enter:

```
touch <public_key_filename.txt>
```

Open the file withTextEdit.

Linux

Create a text file in a text editor such as gedit. Save as `public_key_filename.txt`

Windows

Create a text file in a text editor such as Notepad. Save as `public_key_filename.txt`

4. Add the following contents of the Amazon ECS PGP public key and save the file.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v2  
  
mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU  
jGtqhCWRDkN+qPpHqdAixRgDZAtn2pXY5fEipHgar4CP8QgRnRM02f1741mavr4Vg  
7K/KH8VH1q2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu  
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANiGgEnpNsB1UwfWluPoGZoTx  
N+6pHBjRkIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeLiCtw4kj0+skizBg0  
2K7oVX80c3j5+ZilhL/qDLXmuCb2az5CM1m0oF8EKX5HaNuq1KfwJxqXE6NNIC0  
1FTrT70wD5fMN1d3FanLgv/ZnIrsSaqJ0L6zRSq804LN10WBVbndExk2Kr+5kFx  
51BPgfPgRj5hQ+KTHMa9Y8Z7yUc64Bj1N6F9N17FJuSsfqbdkvRLsQRbcBG9qxX3  
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrlADLV+hvFJkt0z9Gm6xzBq  
1TnWWCz4xIWTuEBA2qE+M1DheVd78a3gIsEaSTfQq0osYXaQbvlnSW0oc1y/5zb  
zizHTJlhLtUyls9WisPz0emeHZicVMfw61EgPrJAiupgc7kyZvFt4YwfwARAQAB  
tCRBbWF6b24gRUNTIDx1Y3Mtc2VjdXJpdH1AYW1hem9uLmNvbT6JAhwEEAECAYF
```

```

AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHn1VPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkq0lw+0gZ4Q/ue/YbWt0At4qY00cEo0HgcnaX
1sB827QifZIVtGMuh94xzm/SJkvngml6KB3YJNwP61A9qJ37/VbVVLzvcmaZ
McWB4HUMNrh0JgBCo0gIpqCbpJEvUc02Bjn23eEJs9kC70UAHqKvNx4d9UzXF
40oISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJX2Csqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBuva0zy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMzpVkr+<C252IaH1HZFEz+TvBVQM
Y80WxmiJW+J6evjo3Nle019Uh71jvoF8z1jbI4bsL2c+QTJm0v7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNjzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItentAK
1EJQhYtyVXcB1hGYN/wzNg2NW0wb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPkmGopsn/reMu1PdInb249nA0dzoN+nj+tTFOYCiaLaFyjs
Z0r1QA0JAjkEEwECACMFAlq1SasCGwMHcwkIBwMCAYVCAIJCgsEFgIDAQIeAQIX
gAAKRC86dmkLVF4T9iFEACenkmdNxswuX34R3c0vamHrPxvfky11F1EUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEwK4XA9I1JjB8hTZUgvXBL046jhG
eM17+crgUyQeetki0QemLbsbxQ40Bd9V7zf7XJraFd8VrwNuNb+9KFtgAsc9rk+
YIT/PEF+YOPysgcxI4sTwghtyCulvnuGoskgDv4v73PALU0ieUrvvQvQwMRvhVx1
0X90J7cC1K0yh1EQQ1aFTgmQjmXexVtwIBm8LvysFK6YXM41Kj0rlz3+6xBIm/qe
bFyLunf4WoiuOp1AaJhK9pRY+XEnGNxdtn4D26Kd0F+PLkm3Tr3Hy3b10k34F1Gr
KVHUq1TzD7cvMnnKEELTuCKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYRt9X42yF1PIEVrNeBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVs9few2GpI5bCgBKbisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVvreJ2yAiFcUpdR1Vhsu0gnxv1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINTH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWknrfMuNwyDuciSI4D5QHa8T+F8fgN40TpwyJirzel
5yoICMr9hVcbzDnv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXYtVvYInEhLvrJ0aV6zhFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxtiAg7r0vyRN9cAxfeSMf77I+XTifigNna8x
t/M0djXr1fjF4pThEi5u6wsuRdfwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDg1
2iHiOKIpQqlbHEfQmHcD2fix+aAJKMnPgnku9qCFEmbgSRJpxz6BfwN1QuKE+i
R6ja0frUnt2jhiGG/F8RceXzohaaC/Cx7LUUCUFwCn7z32C9/Dtj7I1PM0acdZzz
bjJzRKO/ZDv+UN/c9dwAk1zAyPMwGBkUaY68EBstnI1iW34aWm6iHhxioVPKSp
VJfyiXP00EXqujthLAeChfcjns3I12Ysh1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAOIACQUCWVrVJgwIbDAAKRC86dmkLVF4T-zD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGaeBJiu7wjsNWzMF0v15tLjB7AqeVxZn+WkDD/mI0Q450ZvnYzuy
X7DR0Jszah9wrYTzXLvruAu+t6UL0y/XQ4L1GZ9Q6+r+7t1Mvbfy7B1HbxX/gYt
Rwe/uwdibI0CagEzyX+2D3kT01H05XThbXaNF8AN8za91Jt2Q2UR2X5T6JcwtMz
FBvZn13LsmZyE0E0ehS2iUurU4uW0pGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AFY48s8XQwzmc1owYX9b7T8PZiEi0J40qmH0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qk+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBDhBPVPruruLiAenTtZE0sPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkFH6+6rb1ujxMgR0Sqtqr+RyB+V9A5/0gtNZc811K6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zeNu2ZyyHa0q0uH+TEtCw+jsY9IhbEzqN5yQYGi4pVmDkY5vu
1XbJnbqPKpRxgM9BecV9AmBpGbDq/5LnHJXg+G8YQ0qp4lR/hC1TEfdIp5wM8AK
CWsENyt2o1rjgMXiZ0MF8A5oBLkCDQRatUuSARAAt77kj7j2QR2SZe0S1FBvV7oS
mFeSNnz9xZssqrs6bTwSHM6YLDwc7Sdf2esDdyz0NETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0m0mpt+xLLsKcaP7ogIXsyZniEAEsVw8PnfayoiPCdc3cMCR/1TnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvax17PNe1aHGJQY/xo+m
V0bndxf9IY+4oFJ4b1d32WqvxyxEso7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSSm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmu0mhGyTssG+300cGYH7pWYPhazKHMpm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikA01vE3/wgMqCxscbycbLjLD/bXiufWo3rzoezeXjgi/DJx
jKBAYBTY05nMch109oaFd9d0Hbs0UDkIMnsgGBE766Piro6MHo0T0rX107Tp4pI
rwuSosc6XzCzdImj0Wc6axS/HeUKRXdJxwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67olywiw2qRcfBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1r083m3iRac5a/D0AEQEAAykePgQYAOIACQUCWVrVlkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWVrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX01dQR4tKhk0
P0LRqy6z1BY9ILCLowNdGZdqorogUiUymgn3VhEHvtxtOoHcN7q0uM01PNsRn0eS
EYjf8Xrb1c1zkD6xULwm0c1Tb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTf1oS
0f4inq71kjmGK+T1zQ6mUMQug228NUQC+a84EPqYyAeY1sgvgB7hJbhYL0QAxhcW
6m20Rd8iEc6HzJ3yC0CsKip/nRWAbf00vfHfRBp0+m0ZwnJM8cPRFj0qqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNv7giSRIkk0eDSko+bFy6vbMzKUMkUJK3
D3eHFAMkjmbfJmsMTJ0PGn5SB1HyjCZNx6bhIIbQyEUB9gKCMUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96Vxz0phU17T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmr0qX9zBCVxh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA00QxG

```

```

AHM0dJQQvBsmqMvuAdjkDWPfu5y0My5ddU+hiUzUyQLjL5Hhd5L0UDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5fimShJuCe7vSjkPnCng3EIJSggMOPFjJuLwtZ
vjHeDnbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekc21SsIXk/LqEpLMR/0g50Uif
wcEN1rS9IJXbwIy8Me1N9qr5KcKQlmfdfBNeyyceBhyV10MDyHOKC+7PofMtkGBq
13QieRhV5GJ8LB3fc1qHV8pwTT03Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpEcse0XkiRyKT0Tjw0b73CGkBZZpJyqu/xmCV/fp4ALdSW8zbz
FJV0Raihv0WzjpQKhwcu91ABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fn0RGzyhqUIMii+mWra23EwjChaxpvjjcUH
51Lc5Zq781aCYRygYQw+hu5nfkOH1R+Z50Ubxd/aqUfnGIAX7kPMd3Lof4K1d
Q8ppQriUvxVo+4nPVGripTy/PyqCLWDjkguHjsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCE17nbFrm0vIiA75Sa8KnywTDsyZsu3Xc0cf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWH0is6bq819RE6xr1RBVX6uqqQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMbdA64e9Rq7BFR7Tvt8gasCZY1Nr3lydh+dFHIEkH53HzQe6188HEic
+0jVnLkCDQRa55wJARAAYla2Lx6gyoWoJN1a6740q3o8e9d4Kgg00fGMTcf1meq
ivuzgN+3DZHN+9ty2KxMt0nmhBenzdbNjyjMNT1gAgrhPNB4HtXBxum2wS57WK
DNmade914L7FWTPAWBG2Wn4480EHTqsC1ICXXWy9IIcgc1AEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9x0IzQG5CxSnZfk7zcuL60k14Z3
Cmt49k4T/TZU8goWi8tt+rU78/IL3J/FF9+1civ10wuUiJdgfPCSV0UW1JojsdCQA
L+RZJcoXq71f0Fj/eNje0SstCTDPfTCL+kThE6E5neDtBQHBYkEX1BRItdsV4+M
ucgiTrdQFWKF89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJigjNvJscuUwb
+QEeqJIR1eJRhr0+/CHgMs4fZakWF1VfhKBkcKmEjLn1f7EJJUUW84ZhKXj0/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xu0UMKfeggNAFys1K+tDTm3
Bzo8H5ucjCUEmUm91hkGwqTZg01RX5eqPX+JBoSa0bqhggCa5IPinkRa6MgoFPHK
6sYKqroYwBgZm6Js5chpNchvJMs/3WxNOEVg0J3z3VP0DMhxqWm+r+n9zlw8qsA
EQEEAAyKEPgQYAQgACQUCWuecCQIBAgIpCRC86dmkLVF4T8FdIAQZAQgAbgUCWuec
CQAKCRBQ3szEcQ5hr+ykD+4t0LRHFHXuKUcxgGaubaUcvtsFrwBKma1cyjqapms8u
6Sk0wfGRI32G/Gh0rp0Ts/M0kb0bq6VLTh8N5Yc/53ME18zQFw9Y5AmRoW4PZXER
ujss5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLi0EFHYT8UTxwnGmX0vNkMM7md
xD3CV5q6VAt8WKBo/220II3fcQ1c9r/oWx4kXXkb0v9hoGwKbDj1tzqTPpr/xFt
yohqnvImpn1z+Q9zXmbiWYL9/g8VCmW/MN2gju2G3Lu/T1FUWIT4v/50PK6TdeNb
VKJ04+S8bTayqSG9CML1S57KSgCo5HuHqWeSNHI+fpeoX6FALPT9JLDce80Zz1i
czz0MELP37m00Qu0AlmHm/hVzf0f311PtbzczWaE51tJvgUR/nzFo6Ta305Ezhs
3V1EJNQ1Ijf/6DH87SxvAoRIARCuZ0qxBkD0avpFzUtbJd241RA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfk6E8oLpixegS4fiqC16mFr0dyRk+RJJfIUyZ0WTDVmt
g0U1C01ezokMSqkJ7724pyjr2xf/r9/sC6a0Jwb/1kgZkJfc6NqL71xVA31dUga
LEOvEJTTE4g1+tYtfscDvALCtqL0jduSkUo+RXcBitmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6M1I2IdAT/ximu8gbacI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxz1bNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
R05Nm/ZVS+u2imPCRzNUZEMa+d1E6khx0rS0dPiuj407NtPeYDKkoQtNagspsDvh
cK7CSqa1kMq06UBTxqlTSRkm62e0Ctcs3p30eHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mc7X+451cFmcvT94TFNL5HwEUvJpm0gmzILC18yoDTWz1oo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQGmNj3bCluc/jq7ysGq69xiKmTlUeXFm+aojcR05i
zyShIRJZ0GZfuzDYFDb9wamA/YQGygLw//zP5ju5SW26dNx1f3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRG1u/LpNSefhvDFtEIRcp0Hc
bhayG0bk51Bd4mio0XnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj20+X+j
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==

-----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

You may close the text editor.

5. Import the file with the Amazon ECS PGP public key with the following command in the terminal.

```
gpg --import <public_key_filename.txt>
```

6. Download the AWS Copilot 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.

macOS

For macOS systems, run the following command.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/
download/copilot-darwin.asc
```

Linux

For Linux x86 (64-bit) systems, run the following command.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/
download/copilot-linux.asc
```

For Linux ARM systems, run the following command.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/
download/copilot-linux-arm64.asc
```

Windows

Using Powershell, run the following command.

```
Invoke-WebRequest -OutFile 'C:\copilot\copilot.asc' https://github.com/aws/copilot-
cli/releases/latest/download/copilot-windows.exe.asc
```

7. Verify the signature with the following command.

- For macOS and Linux systems:

```
gpg --verify copilot.asc /usr/local/bin/copilot
```

- For Windows systems:

```
gpg --verify 'C:\copilot\copilot.asc' 'C:\copilot\copilot.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](#).

8. For Windows installations, run the following command on Powershell to add AWS Copilot directory to the path.

```
$Env:PATH += "<path to Copilot executable files>"
```

Next steps

After installation, learn how to deploy an Amazon ECS application using AWS Copilot. For more information, see [Getting started with Amazon ECS using AWS Copilot \(p. 41\)](#).

Getting started with Amazon ECS using AWS Copilot

Get started with Amazon ECS using AWS Copilot by deploying an Amazon ECS application.

Prerequisites

Before you begin, make sure that you meet the following prerequisites:

- Set up an AWS account. For more information see [Set up to use Amazon ECS \(p. 10\)](#).
- Install the AWS Copilot CLI. Releases currently support Linux and macOS systems. For more information, see [Installing the AWS Copilot CLI \(p. 35\)](#).
- Install and configure the AWS CLI. For more information, see [AWS Command Line Interface](#).
- Run aws configure to set up a default profile that the AWS Copilot CLI will use to manage your application and services.
- Install and run Docker. For more information, see [Get started with Docker](#).

Deploy your application using one command

Make sure that you have the AWS command line tool installed and have already run aws configure before you start.

Deploy the application using the following command.

```
git clone https://github.com/aws-samples/amazon-ecs-cli-sample-app.git demo-app && \
cd demo-app && \
copilot init --app demo \
--name api \
--type 'Load Balanced Web Service' \
--dockerfile './Dockerfile' \
--port 80 \
--deploy
```

Deploy your application step by step

Step 1: Configure your credentials

Run `aws configure` to set up a default profile that the AWS Copilot CLI uses to manage your application and services.

```
aws configure
```

Step 2: Clone the demo app

Clone a simple Flask application and Dockerfile.

```
git clone https://github.com/aws-samples/amazon-ecs-cli-sample-app.git demo-app
```

Step 3: Set up your application

1. From within the `demo-app` directory, run the `init` command.

For Windows users, run the `init` command from the folder that contains the downloaded `copilot.exe` file.

```
copilot init
```

AWS Copilot walks you through the setup of your **first application and service** with a series of terminal prompts, starting with **next step**. If you have already used AWS Copilot to deploy applications, you're prompted to choose one from a list of application names.

2. Name your application.

```
What would you like to name your application? [? for help]
```

Enter **demo**.

Step 4: Set up an ECS Service in your "demo" Application

1. You're prompted to choose a service type. You're building a simple Flask application that serves a small API.

```
Which service type best represents your service's architecture? [Use arrows to move, type to filter, ? for more help]
> Load Balanced Web Service
  Backend Service
  Scheduled Job
```

Choose **Load Balanced Web Service**.

2. Provide a name for your service.

```
What do you want to name this Load Balanced Web Service? [? for help]
```

Enter **api** for your service name.

3. Select a Dockerfile.

```
Which Dockerfile would you like to use for api? [Use arrows to move, type to filter, ? for more help]
> ./Dockerfile
Use an existing image instead
```

Choose *Dockerfile*.

For Windows users, enter the path to the Dockerfile in the demo-app folder (*`...\\demo-app\\Dockerfile`*`\\`).

4. Define port.

```
Which port do you want customer traffic sent to? [? for help] (80)
```

Enter **80** or accept default.

5. You will see a log showing the application resources being created.

```
Creating the infrastructure to manage services under application demo.
```

6. After the application resources are created, deploy a test environment.

```
Would you like to deploy a test environment? [? for help] (y/N)
```

Enter **y**.

```
Proposing infrastructure changes for the test environment.
```

7. You will see a log displaying the status of your application deployment.

```
Note: It's best to run this command in the root of your Git repository.
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.
```

```
Use existing application: No
Application name: demo
Workload type: Load Balanced Web Service
Service name: api
Dockerfile: ./Dockerfile
no EXPOSE statements in Dockerfile ./Dockerfile
Port: 80
Ok great, we'll set up a Load Balanced Web Service named api in application demo
listening on port 80.

# Created the infrastructure to manage services under application demo.

# Wrote the manifest for service api at copilot/api/manifest.yml
Your manifest contains configurations like your container size and port (:80).

# Created ECR repositories for service api.

All right, you're all set for local development.
Deploy: Yes

# Created the infrastructure for the test environment.
- Virtual private cloud on 2 availability zones to hold your services [Complete]
- Virtual private cloud on 2 availability zones to hold your services [Complete]
  - Internet gateway to connect the network to the internet [Complete]
  - Public subnets for internet facing services [Complete]
```

```
- Private subnets for services that can't be reached from the internet [Complete]
- Routing tables for services to talk with each other [Complete]
- ECS Cluster to hold your services [Complete]
# Linked account aws_account_id and region region to application demo.  
  
# Created environment test in region region under application demo.  
  
Environment test is already on the latest version v1.0.0, skip upgrade.  
[+] Building 0.8s (7/7) FINISHED  
=> [internal] load .dockignore  
      0.1s  
=> => transferring context: 2B  
      0.0s  
=> [internal] load build definition from Dockerfile  
      0.0s  
=> => transferring dockerfile: 37B  
      0.0s  
=> [internal] load metadata for docker.io/library/nginx:latest  
      0.7s  
=> [internal] load build context  
      0.0s  
=> => transferring context: 32B  
      0.0s  
=> [1/2] FROM docker.io/library/  
nginx@sha256:aeade65e99e5d5e7ce162833636f692354c227ff438556e5f3ed0335b7cc2f1b      0.0s  
=> CACHED [2/2] COPY index.html /usr/share/nginx/html  
      0.0s  
=> exporting to image  
      0.0s  
=> => exporting layers  
      0.0s  
=> => writing image  
sha256:3ee02fd4c0f67d7bd808ed7fc73263880649834cbb05d5ca62380f539f4884c4  
      0.0s  
=> => naming to aws_account_id.dkr.ecr.region.amazonaws.com/demo/api:cee7709  
      0.0s  
WARNING! Your password will be stored unencrypted in /home/user/.docker/config.json.  
Configure a credential helper to remove this warning. See  
https://docs.docker.com/engine/reference/commandline/login/#credentials-store  
  
Login Succeeded  
The push refers to repository [aws_account_id.dkr.ecr.region.amazonaws.com/demo/api]  
592a5c0c47f1: Pushed  
6c7de695ede3: Pushed  
2f4accd375d9: Pushed  
ffc9b21953f4: Pushed  
cee7709: digest: sha_digest  
  
# Deployed api, you can access it at http://demo-  
Publi-10Q8VMS2VC2WG-561733989.region.elb.amazonaws.com.
```

Step 5: Verify your application is running

View the status of your application by using the following commands.

List all of your AWS Copilot applications.

```
copilot app ls
```

Show information about the environments and services in your application.

```
copilot app show
```

Show information about your environments.

```
copilot env ls
```

Show information about the service, including endpoints, capacity and related resources.

```
copilot svc show
```

List of all the services in an application.

```
copilot svc ls
```

Show logs of a deployed service.

```
copilot svc logs
```

Show service status.

```
copilot svc status
```

List available commands and options.

```
copilot --help
```

```
copilot init --help
```

Step 6. Learn to create a CI/CD Pipeline

Instructions can be found in the [ECS Workshop](#) detailing how to fully automate a CI/CD pipeline and git workflow using AWS Copilot.

Step 7: Clean up

Run the following command to delete and clean up all resources.

```
copilot app delete
```

Getting started with Amazon ECS using the AWS CDK

The AWS Cloud Development Kit (AWS CDK) is an Infrastructure-as-Code (IAC) framework that you can use to define AWS cloud infrastructure by using a programming language of your choosing. To define your own cloud infrastructure, you first write an app (in one of the CDK's supported languages) that contains one or more stacks. Then, you synthesize it to an AWS CloudFormation template and deploy

your resources to your AWS account. Follow the steps in this topic to deploy a containerized web server with Amazon Elastic Container Service (Amazon ECS) and the AWS CDK on Fargate.

The AWS Construct Library, included with the CDK, provides modules that you can use to model the resources that AWS services provide. For popular services, the library provides curated constructs with smart defaults and best practices. One of these modules, specifically [aws-ecs-patterns](#), provides high-level abstractions that you can use to define your containerized service and all the necessary supporting resources in a few lines of code.

This topic uses the [ApplicationLoadBalancedFargateService](#) construct. This construct deploys an Amazon ECS service on Fargate behind an application load balancer. The aws-ecs-patterns module also includes constructs that use a network load balancer and run on Amazon EC2.

Before starting this task, set up your AWS CDK development environment, and install the AWS CDK by running the following command. For instructions on how to set up your AWS CDK development environment, see [Getting Started With the AWS CDK - Prerequisites](#).

```
npm install -g aws-cdk
```

Note

These instructions assume you are using AWS CDK v2.

Topics

- [Step 1: Set up your AWS CDK project \(p. 46\)](#)
- [Step 2: Use the AWS CDK to define a containerized web server on Fargate \(p. 48\)](#)
- [Step 3: Test the web server \(p. 51\)](#)
- [Step 4: Clean up \(p. 52\)](#)
- [Next steps \(p. 52\)](#)

Step 1: Set up your AWS CDK project

Create a directory for your new AWS CDK app and initialize the project.

TypeScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language typescript
```

JavaScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language javascript
```

Python

```
mkdir hello-ecs
cd hello-ecs
cdk init --language python
```

After the project is started, activate the project's virtual environment and install the AWS CDK's baseline dependencies.

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-ecs
cd hello-ecs
cdk init --language java
```

Import this Maven project to your Java IDE. For example, in Eclipse, use **File > Import > Maven > Existing Maven Projects**.

C#

```
mkdir hello-ecs
cd hello-ecs
cdk init --language csharp
```

Note

The AWS CDK application template uses the name of the project directory to generate names for source files and classes. In this example, the directory is named `hello-ecs`. If you use a different project directory name, your app won't match these instructions.

AWS CDK v2 includes stable constructs for all AWS services in a single package that's called `aws-cdk-lib`. This package is installed as a dependency when you initialize the project. When working with certain programming languages, the package is installed when you build the project for the first time. This topic covers how to use an Amazon ECS Patterns construct, which provides high-level abstractions for working with Amazon ECS. This module relies on Amazon ECS constructs and other constructs to provision the resources that your Amazon ECS application needs.

The names that you use to import these libraries into your CDK application might differ slightly depending on which programming language you use. For reference, the following are the names that are used in each supported CDK programming language.

TypeScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

JavaScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

Python

```
aws_cdk.aws_ecs
aws_cdk.aws_ecs_patterns
```

Java

```
software.amazon.awscdk.services.ecs
software.amazon.awscdk.services.ecs.patterns
```

C#

```
Amazon.CDK.AWS.ECS
```

Amazon.CDK.AWS.ECS.Patterns

Step 2: Use the AWS CDK to define a containerized web server on Fargate

Use the container image [amazon-ecs-sample](#) from DockerHub. This image contains a PHP web app that runs on Amazon Linux 2.

In the AWS CDK project that you created, edit the file that contains the stack definition to resemble one of the following examples.

Note

A stack is a unit of deployment. All resources must be in a stack, and all the resources that are in a stack are deployed at the same time. If a resource fails to deploy, any other resources that were already deployed are rolled back. An AWS CDK app can contain multiple stacks, and resources in one stack can refer to resources in another stack.

TypeScript

Update lib/hello-ecs-stack.ts so that it resembles the following.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsp from 'aws-cdk-lib/aws-ecs-patterns';

export class HelloEcsStack extends cdk.Stack {
    constructor(scope: Construct, id: string, props?: cdk.StackProps) {
        super(scope, id, props);

        new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
            taskImageOptions: {
                image: ecs.ContainerImage.fromRegistry('amazon/amazon-ecs-sample'),
            },
            publicLoadBalancer: true
        });
    }
}
```

JavaScript

Update lib/hello-ecs-stack.js so that it resembles the following.

```
const cdk = require('aws-cdk-lib');
const { Construct } = require('constructs');
const ecs = require('aws-cdk-lib/aws-ecs');
const ecsp = require('aws-cdk-lib/aws-ecs-patterns');

class HelloEcsStack extends cdk.Stack {
    constructor(scope = Construct, id = string, props = cdk.StackProps) {
        super(scope, id, props);

        new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
            taskImageOptions: {
                image: ecs.ContainerImage.fromRegistry('amazon/amazon-ecs-sample'),
            },
            publicLoadBalancer: true
        });
    }
}
```

```
module.exports = { HelloEcsStack }
```

Python

Update `hello-ecs/hello_ecs_stack.py` so that it resembles the following.

```
import aws_cdk as cdk
from constructs import Construct

import aws_cdk.aws_ecs as ecs
import aws_cdk.aws_ecs_patterns as ecsp

class HelloEcsStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ecsp.ApplicationLoadBalancedFargateService(self, "MyWebServer",
            task_image_options=ecsp.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            public_load_balancer=True
        )
```

Java

Update `src/main/java/com.myorg>HelloEcsStack.java` so that it resembles the following.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

import software.amazon.awscdk.services.ecs.ContainerImage;
import software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedFargateService;
import software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedTaskImageOptions;

public class HelloEcsStack extends Stack {
    public HelloEcsStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloEcsStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);

        ApplicationLoadBalancedFargateService.Builder.create(this, "MyWebServer")
            .taskImageOptions(ApplicationLoadBalancedTaskImageOptions.builder()
                .image(ContainerImage.fromRegistry("amazon/amazon-ecs-sample"))
                .build())
            .publicLoadBalancer(true)
            .build();
    }
}
```

C#

Update `src/HelloEcs/HelloEcsStack.cs` so that it resembles the following.

```
using Amazon.CDK;
```

```

using Constructs;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
namespace HelloEcs
{
    public class HelloEcsStack : Stack
    {
        internal HelloEcsStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props)
        {
            new ApplicationLoadBalancedFargateService(this, "MyWebServer",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-sample")
                    },
                    PublicLoadBalancer = true
                });
        }
    }
}

```

The preceding short snippet includes the following:

- The service's logical name: MyWebServer.
- The container image that was obtained from DockerHub: amazon/amazon-ecs-sample.
- Other relevant information, such as the fact that the load balancer has a public address and is accessible from the Internet.

The AWS CDK will create all the resources that are required to deploy the web server including the following resources. These resources were omitted in this example.

- Amazon ECS cluster
- Amazon VPC and Amazon EC2 instances
- Auto Scaling group
- Application Load Balancer
- IAM roles and policies

Some automatically provisioned resources are shared by all Amazon ECS services defined in the stack.

Save the source file, then run the `cdk synth` command in your application's main directory. The AWS CDK runs the app and synthesizes an AWS CloudFormation template from it, and then displays the template. The template is an approximately 600-line YAML file. The beginning of the file is shown here. Your template might differ from this example.

```

Resources:
  MyWebServerLB3B5FD3AB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      LoadBalancerAttributes:
        - Key: deletion_protection.enabled
          Value: "false"
      Scheme: internet-facing
      SecurityGroups:
        - Fn::GetAtt:

```

```
- MyWebServerLBSecurityGroup01B285AA
- GroupId
Subnets:
- Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1Subnet3C273B99
- Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2Subnet95FF715A
Type: application
DependsOn:
- EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1DefaultRouteFF4E2178
- EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2DefaultRouteB1375520
Metadata:
aws:cdk:path: HelloEcsStack/MyWebServer/LB/Resource
MyWebServerLBSecurityGroup01B285AA:
Type: AWS::EC2::SecurityGroup
Properties:
GroupDescription: Automatically created Security Group for ELB
HelloEcsStackMyWebServerLB06757F57
SecurityGroupIngress:
- CidrIp: 0.0.0.0/0
Description: Allow from anyone on port 80
FromPort: 80
IpProtocol: tcp
ToPort: 80
VpcId:
Ref: EcsDefaultClusterMnL3mNNYNVpc7788A521
Metadata:
aws:cdk:path: HelloEcsStack/MyWebServer/LB/SecurityGroup/Resource
# and so on for another few hundred lines
```

To deploy the service in your AWS account, run the `cdk deploy` command in your application's main directory. You're asked to approve the IAM policies that the AWS CDK generated.

The deployment takes several minutes during which the AWS CDK creates several resources. The last few lines of the output from the deployment include the load balancer's public hostname and your new web server's URL. They are as follows.

```
Outputs:
HelloEcsStack.MyWebServerLoadBalancerDNSXXXXXX = Hello-MyWeb-ZZZZZZZZZZZZ-ZZZZZZZZZZ.us-
west-2.elb.amazonaws.com
HelloEcsStack.MyWebServerServiceURLYYYYYYYY = http://Hello-MyWeb-ZZZZZZZZZZZZ-
ZZZZZZZZZ.us-west-2.elb.amazonaws.com
```

Step 3: Test the web server

Copy the URL from the deployment output and paste it into your web browser. The following welcome message from the web server is displayed.

Simple PHP App

Congratulations

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

The container is running PHP version 5.4.16.

Step 4: Clean up

After you're finished with the web server, end the service using the CDK by running the `cdk destroy` command in your application's main directory. Doing this prevents you from incurring any unintended charges in the future.

Next steps

To learn more about how to develop AWS infrastructure using the AWS CDK, see the [AWS CDK Developer Guide](#).

For information about writing AWS CDK apps in your language of choice, see the following:

TypeScript

[Working with the AWS CDK in TypeScript](#)

JavaScript

[Working with the AWS CDK in JavaScript](#)

Python

[Working with the AWS CDK in Python](#)

Java

[Working with the AWS CDK in Java](#)

C#

[Working with the AWS CDK in C#](#)

For more information about the AWS Construct Library modules used in this topic, see the following AWS CDK API Reference overviews.

- [aws-ecs](#)
- [aws-ecs-patterns](#)

Creating Amazon ECS resources with AWS CloudFormation

Amazon ECS is integrated with AWS CloudFormation, a service that you can use to model and set up AWS resources with templates that you define. This way, you can spend less time creating and managing your resources and infrastructure. Using AWS CloudFormation, you can create a template that describes all the AWS resources that you want, such as specific Amazon ECS clusters. Then, AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon ECS resources in a consistent and repeatable manner. You describe your resources one time, and then provision the same resources again across multiple AWS accounts and AWS Regions.

Amazon ECS and AWS CloudFormation templates

To provision and configure resources for Amazon ECS and related services, make sure that you're familiar with [AWS CloudFormation templates](#). AWS CloudFormation templates are text files in the JSON or YAML format that describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with either the JSON or YAML format, or both, you can use AWS CloudFormation

Designer to get started using AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the [AWS CloudFormation User Guide](#).

Amazon ECS supports creating clusters, task definitions, services, and task sets in AWS CloudFormation. The following examples demonstrate how to create resources with these templates using the AWS CLI. You can also create these resources using the AWS CloudFormation console. For more information about how to create resources using the AWS CloudFormation console, see the [AWS CloudFormation User Guide](#).

Example templates

Creating Amazon ECS resources using separate stacks

The following examples show how to create Amazon ECS resources by using separate stacks for each resource.

Amazon ECS task definitions

You can use the following template to create a Fargate Linux task.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "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"
                            }
                        ]
                    }
                ],
                "Memory": 512,
                "NetworkMode": "awsvpc",
                "TaskRoleArn": "arn:aws:iam::123456789012:task-role/ECS-Task-Role"
            }
        }
    }
}
```

```
        "Cpu": 256,
        "ExecutionRoleArn": "arn:aws:iam::aws_account_id:role/
ecsTaskExecutionRole",
        "Family": "task-definition-cfn",
        "Memory": 512,
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
            "FARGATE"
        ],
        "RuntimePlatform": {
            "OperatingSystemFamily": "LINUX"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  ECSTaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      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::aws\_account\_id:role/ecsTaskExecutionRole'
      Family: task-definition-cfn
      Memory: 512
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX
```

Amazon ECS clusters

You can use the following template to create an empty cluster.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "MyEmptyCluster"
            }
        }
    }
}
```

YAML

```
AWS::TemplateFormatVersion: 2010-09-09
Resources:
  ECSCluster:
    Type: 'AWS::ECS::Cluster'
    Properties:
      ClusterName: MyEmptyCluster
```

Creating multiple Amazon ECS resources in one stack

You can use the following example template to create multiple Amazon ECS resources in one stack. The template creates an Amazon ECS cluster that's named CFNcluster. The cluster contains a Linux Fargate task definition that sets up a web server. The template also creates a service that's named cfn-service that launches and maintains the task defined by the task definition. Before you use this template, make sure that the subnet and security group IDs in the service's NetworkConfiguration all belong to the same VPC and that the security group has the necessary rules. For more information about security group rules, see [Security group rules](#) in the Amazon VPC user guide.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "CFNcluster"
            }
        },
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "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::aws_account_id::role/
    ecsTaskExecutionRole",
        "Family": "task-definition-cfn",
        "Memory": 512,
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
            "FARGATE"
        ],
        "RuntimePlatform": {
            "OperatingSystemFamily": "LINUX"
        }
    },
    "ECSService": {
        "Type": "AWS::ECS::Service",
        "Properties": {
            "ServiceName": "cfn-service",
            "Cluster": {
                "Ref": "ECSCluster"
            },
            "DesiredCount": 1,
            "LaunchType": "FARGATE",
            "NetworkConfiguration": {
                "AwsVpcConfiguration": {
                    "AssignPublicIp": "ENABLED",
                    "SecurityGroups": [
                        "sg-abcdef01234567890"
                    ],
                    "Subnets": [
                        "subnet-abcdef01234567890"
                    ]
                }
            },
            "TaskDefinition": {
                "Ref": "ECSTaskDefinition"
            }
        }
    }
}

```

YAML

```

AWSTemplateFormatVersion: 2010-09-09
Resources:

```

```

ECSCluster:
  Type: 'AWS::ECS::Cluster'
  Properties:
    ClusterName: CFNCluster
ECSTaskDefinition:
  Type: 'AWS::ECS::TaskDefinition'
  Properties:
    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::aws_account_id:role/ecsTaskExecutionRole'
    Family: task-definition-cfn
    Memory: 512
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
    RuntimePlatform:
      OperatingSystemFamily: LINUX
ECSService:
  Type: 'AWS::ECS::Service'
  Properties:
    ServiceName: cfn-service
    Cluster: !Ref ECSCluster
    DesiredCount: 1
    LaunchType: FARGATE
    NetworkConfiguration:
      AwsVpcConfiguration:
        AssignPublicIp: ENABLED
        SecurityGroups:
          - sg-abcdef01234567890
        Subnets:
          - subnet-abcdef01234567890
    TaskDefinition: !Ref ECSTaskDefinition

```

Using the AWS CLI to create resources from templates

The following command creates a stack that's named `ecs-stack` using a template body file that's named `ecs-template-body.json`. Ensure that the template body file is in the JSON or YAML format. The location of the file is specified in the `--template-body` parameter. In this case, the template body file is located in the current directory.

```
aws cloudformation create-stack \
  --stack-name ecs-stack \
  --template-body file://ecs-template-body.json
```

To ensure that resources are created correctly, check the Amazon ECS console or alternatively use the following commands:

- The following command lists all task definitions.

```
aws ecs list-task-definitions
```

- The following command lists all clusters.

```
aws ecs list-clusters
```

- The following command lists all services defined in the cluster `CFNCluster`. Replace `CFNCluster` with the name of the cluster that you want to create the service in.

```
aws ecs list-services \
  --cluster CFNCluster
```

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

Using the Amazon ECS command line interface

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

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, a popular open-source specification for defining and running multi-container applications. Use the ECS 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 only supports the major versions of [Docker Compose file syntax](#) versions 1, 2, and 3. The version specified in the compose file must be the string "1", "1.0", "2", "2.0", "3", or "3.0". Docker Compose minor versions are not supported.

The source code for the Amazon ECS CLI is [available on GitHub](#). This tool is no longer being actively developed. For release notes, see [Changelog](#).

Installing the Amazon ECS CLI

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

The following steps demonstrate how to install the Amazon ECS CLI on your macOS, Linux, or Windows system.

To install the Amazon ECS CLI

1. Download the Amazon ECS CLI binary.

macOS

```
sudo curl -Lo /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest
```

Linux

```
sudo curl -Lo /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest
```

Windows

Open Windows PowerShell and enter the following commands.

Note

If you encounter permission issues, ensure that you have administrator access on Windows and you are running PowerShell as an administrator.

```
New-Item -Path 'C:\Program Files\Amazon\ECSCLI' -ItemType Directory  
Invoke-WebRequest -OutFile 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe' https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-latest.exe
```

2. Verify the Amazon ECS CLI using PGP signatures. The Amazon ECS CLI executables are cryptographically signed using PGP signatures. The PGP signatures can be used to verify the validity of the Amazon ECS CLI executable. Use the following steps to verify the signatures using the GnuPG tool.
 - a. Download and install GnuPG. For more information, see the [GnuPG website](#).

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
```

Linux

Install gpg using the package manager on your flavor of Linux.

Windows

Download the Windows simple installer from the GnuPG website and install as an Administrator. After you install GnuPG, close and reopen the Administrator PowerShell.

For more information, see [GnuPG Download](#).

- b. Verify the GnuPG path is added to your environment path.

macOS

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
PATH=$PATH:<path to GnuPG executable files>
```

Linux

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
export PATH=$PATH:<path to GnuPG executable files>
```

Windows

```
Write-Output $Env:PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
$Env:PATH += "<path to GnuPG executable files>"
```

- c. Create a local plain text file.

macOS

On the terminal, enter:

```
touch <public_key_filename.txt>
```

Open the file with TextEdit.

Linux

Create a text file in a text editor such as gedit. Save as `public_key_filename.txt`

Windows

Create a text file in a text editor such as Notepad. Save as `public_key_filename.txt`

- d. Add the following contents of the Amazon ECS PGP public key and save the file.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v2  
  
mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU  
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRM02f1741mavr4Vg  
7K/KH8VHlq2uRw32/B94XL EgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu  
BoQAhjBQ+bEm0kNy0LjNgjN1nL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx  
N+6pHBjRkIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeLiCrw4kj0+skizBg0  
2K7ovX80c3j5+ZilhL/qDLXmUCb2az5cMM1m0Of8EKX5HaNuq1KfwJxqXE6NNIC0  
1FTrT7QwD5fMNld3FanLgv/SNlrsSaqJ0L6zRSq804LN10WBVbndExk2Kri+5kFxn  
51BPgfPgRj5hQ+KTHMa9Y8Z7yUc64BjIn6F9N17FJuSsfqbdkvRLsQRbcBG9qxX3  
rJAЕhieJzVMEUN1+EgeCkxj5xuSkNu7zw2c3hQzqEcrADLV+hvFJkt0z9Gm6xzbg  
1TnWWCz4xrIWtuEBA2qE+M1Dhevd78a3gIsEaStFQ0osYXaQbvlnSW0oc1y/5zb  
zizHTJihLtUyls9WlsP2s0emeHZicVmFW61EgPrJAIupgc7kyZvFt4YwfARAQAB  
tCRbbWF6b24gRUNTIDx1Y3MtC2VjdXJpdH1AYW1hem9uLmNbT6JAhwEEAACAYF  
AljL0YACgkQHivRXs0TaQrg1g/+JppwPqHn1VPmv7lessB8I5UqZeD6p6uVphd7  
Bs3pcPp8BV7BdRbs3sPlt5bV1+rkq0lw+0gZ4Q/ue/YbWt0At4qY00cEo0HgcnaX  
1sB827QIfZIVtGWMhuh94xz.../SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmaZ  
McWB4HUMNrh0JgBCo0gIpqCbpJEvuC02Bjn23eEjsS9kC70UAHyQkVnx4d9UzXF  
40oISF6hmQKIBoLnRiAlj50vs3GhvHQ0ThYq0Grk/KMJJX2Csqt7tWJ8gk1n3H3Y  
SReRXJRnv7DsDDBwFgT6r5Q2H1TBuaoZy5hF6maD09nHcnvBjqADzeT8Tr/Qu  
bBCLzkNSYqqkpgtw7seoD2P4n1giRvDA0EfMzPvkUr+C252IaH1HZFEz+TvbVQM  
Y80WWxmIJW+J6evjo3N1e019UhV71jvoF8z1jbI4sL2c+QTJm0v7nRqzDQgCWyp  
Id/v2dUVVTk1j9omuLBbWnjZQcb+72LcIzJhYmaP1HC4LcKQG+/f4lexuItenatK  
1EJQhYtyVxcB1h6Yn/wzNg2NWObw3vqY/F7m6u9ixAwgtImgPCDE4aJ86zriXYFz  
N2HqkTSQh77Z8PKmyGopsrn/teMu1lPdINb249nA0dzon+N+j+tTFOYCIaLaFyjs  
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHcwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX  
gAAKCRC86dmkL VF4T9iFEACEEnkm1dNxswUx34R3c0vamHrPxvfkyI1F1EUen8D1h  
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy  
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEwk4XAF9I1jB8hTZUgvXBL046jhG  
eM17+crqUyQeetki0QemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFTgAsc9rk+  
YIT/PEf+YOPsgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1  
0X90J7cC1K0yh1EQQ1aFTgmQjmXexTwIBm8LvsyfK6YXM41KjOrlz3+6xBIm/qe  
bfYLnf4WoiuOp1AaJh9pRY+XEnGndtN4D26Kd0F+PLkm3Tr3H3b10k34F1Gr  
KVHUq1T2D7cvMnnKEELTUCkX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE  
XQ4zuF2IGCpvBFhYAlt5Un5zwkwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I  
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVrNeBRJ3HDezagJrNh0GQWRQkhIx  
gz6/cTR+ekr5TpTvzs9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6  
1bkCDQRatUmzARAAXNPvVwreJ2yAiFcUpdR1Vhsu0gnxvs1QgsIw3H7+Pacr9Hpe  
8uftYZqdC82KeSKhpHq7c8gMTMuCIIntH25x9Bc73E33ejCL9Lqv1TL7+QkgHe  
T+JiH ZwD8Mx2K+LvvVu/aWkNrfMuNwyDuciSI4D5QHa8T+F8fgN40TpwYjirzel  
5yoICMr9hVcbzDnv/ozKcxjx+KKgnFc3wirnDFjntfDAT7ecwbUTL+viQKJ646s+  
psiqXRytVvYInEhLViJ0aV6zhFoige/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu  
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAxfeSMf77I+XTifigNna8x  
t/M0djXr1fjF4pThEi5u6WsruRdfwY2azEv3vedotTi4HoJReH6dFRa6y8c+UDgl  
2iHiOKIpQqlbHEFQmHcDd2fix+AaJKmPGNku9qCFEmbgSRJpXz6BfwnY1QuKE+i  
R6jAofrUnt2jh1GG/F8RceXzohaaC/Cx7LUCUFwC0n7z32C9/Dtj7I1PM0acdZzz  
bjjZrKO/ZDv+UN/c9dwAk11zAyPMwGbKuaY68EBstnIliw34aWm6IiHhxioVPKSp  
VJfyiXP00ExqujthLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAQEAE  
AYkChwQYQAQIAQUCWriVJqwIbDAAKCRC86dmkL VF4T+Zd/D/9x/8APzgNJF3o3StrF  
jvnV1ycyhWYGAeBjiu7wjsNwwzMF0v15tLjB7AqeVxZn+WKDD/mIOQ450ZvnYzuy  
X7DR0JszaH9wrYTzLVRuAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbry7B1HbxV/gYt  
Rwe/uwdibI0CagEzyX+2D3kT01H05XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz  
FBvZn13LSmZyE0EohS2iUurU4uW0pGppuqVnbij0jbCvCHkgDGrqZ0smKNA0ng54  
F365W3g8AfY48s8XQwzmcliowYX9bT8PzIEi0J4QmQh0aXkpqZyFefuWe0L2R94S  
XKzri+grh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe  
TXiKQ8DBWDhBPVPrruLiaenTtZEoSpc4I85yt5U9RoPTStc0r34s3w5yEaJagt6S  
Gc5r9ysjkfH6+6ribi1ujxMgRO5qtqr+RyB+v9A5/0gtNZc811K6u4Uo0Cde8jUUW  
vqWKvjJB/Kz3u4zaeNu2ZyyHa0q0uH+TETcW+jsy9IhbEzqN5yQYGi4pVmDky5v
```

1XbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQ0gp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZ0MF8A5oBLkCDQRatUuSARAAR77kj7j2QR2SZeOS1FBvV7oS
mFeSNz9xZssqrsms6bTwSHM6YLDwc7Sdf2esDdyz0NETwqxVCg+FxgL8hmo9hS4c
rR6tmrP0m0mpt+xLLsKcaP7ogIXsyZnrEAESvW8PnfayoiPCdc3cMCR/1TnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvax17PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4b1D32WqvxyESo7vW6WBh7oqv3Zbm0yQir8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeYHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D55TCxbKdmu0mhGyTssOG+300cGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLzJXikA01vE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAYBTY05nMcth1090aFd9d0Hbs0UDkIMmsgGBE766Piro6MH00T0rXl07Tp4pI
rwuS0sc6XzCzdImj0Wc6axS/HeUKRXdJXjwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmw2qRcUbfgtH3e15XZU28MiCpue8Y8GKJoBAUyvf
KeM1r08Jm3iRAc5a/D0AEQEAAyKEPgQYAQIAQCWrlVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrlVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX01d0R4tKhQ
P0LRqy6z1BY9ILCLowNdGZdqriogUiUymgn3VhEHVtxToohCn7q0uM01PNsRn0eS
EYjf8Xrb1clzkD6xULwm0c1Tb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTf1oS
Of4inq71kjmgK+T1zQ6mUMQUG228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyZJ3yC0CsKip/nRWAbf00vfHFRBp0+m0ZwnJM8cPRFj0qqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZN7g1sRIkk0eDSk0+bFy6VbMzKUMKUJK3
D3eHFAMkujmbfJmSMTJ0PGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAjxLR/shZ5rk96VxZopuH17T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmr0qX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEpsalrCdtanUBHRXA00QxG
AHModJQvBsmqMvuAdjkdWpFu5y0My5ddU+hiUyQljL5Hhd5LOUDdewlZgIw1j
xIEAUzDKetnemM8GkHxDgg8koev5firmShJuce7vSjKpCNg3EIJsgqMOPFjJuLwtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WA1D6Tfekkc21SsIXk/LqEpLMR/0g50Uif
wcEN1rS9IjXBwIy8Me1N9qr5KcKQLmfdfBNEyyceBhyV10MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fc1qHV8pwTT03Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTwpECse0XkiRyKT0Tjw0b73CGkBZpJyqu/rimCV/fp4ALdSw8zbz
FJVORaivh0WwzjpfcQKhwcU91ABxi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUImii+mWa23EwjChaxpvjjcUH
5i1Lc5Zq781aCYRgYQw+hu5nFk0H1R+Z50Ubxd/aqUfnGIAX7kPMD3Lof4K1dD
Q8ppQriUvxVo+4nPv6rpTy/PyqCLWDjkguHpsEfMsMkwajrAz0QNSAU5CJ0G2Zu4
yxvY1umHCE17nbFim0vIia75Sa8KnywTDsyZsu3Xc0cf3g+g1xWTpjJqy2bYX1qz
9uDOWtArWHOis6bq819RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEmahbc
e71BfeB1CMBdA64e9Rq7bFR7Tvt8gasCZY1Nr3lydh+dFHIEKH53HzQe6188HEic
+0jVnLkCDQRa55wJARAAYLy2Lx6gyoWoJN1a6740q3o8e9d4KggQ0fGMTCf1meq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNjyjMNT1gAgrhPNB4HtXBxum2wS57WK
DNmade914L7FWTPAWBG2Wn4480EHTqsC1ICXXW9yIIICgc1AEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFxwQmDmkAsCygQMvhAlmu9x0IzQG5CxSnZFK7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1ciw10wuUiDgfpCSvOUW1jojsdCQA
L+RZjcoXq71f0fj/eNje0SstCTDPfTCL+kTheE5neDtbQHBykEX1BRiTedsV4+M
ucgiTrdQFWf89G72xdv8ut9AYYQ2BbEYU+JAYHUH8rYYui2dHKJigjNvJscuUWb
+QEJqIRleJRhro+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXj0/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIx9nxUc0UMKfeggNAFys1K+TDTm3
BzoH5ucjCUEmUm91hkGwqTZg01RX5eqPX+jBoSa0bqhqaCa5IPinKRa6MgoFPHK
6sYKqroYwBggZm6j5s5chpNchvJMs/3WXNOEVg0j3z3vP0DMhxqWm+r+n9z1W8qsA
E0EAAyKEPgQYAQgACQUCWuecCQIBagIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4t0LRHFHuKUcxgGaubaUcVtsF1wBKma1cYjqaPms8u
6sk0wfGRI32G/Gh0rp0Ts/M0kb0bq6VLT8N5Yc/53ME18zQFw9Y5AmRoW4PZXR
ujs5s7p4oR7xHMiMjCCBn1bvrR+34YPfgzTcgLi0EFHYT8UTxwnGmX0vNkMM7md
xD3CV5q6VAt8WKBo/220II3fcQ1c9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPip/xFt
yoHqnvImpn1z+Q9zXmbwYLY9/g8VcmW/NN2gju2G3Lu/T1FWIT4v/50PK6TdeNb
VKJ04+S8bTayqSG9CM157KSgCo5HuQWeSNHI+fpe5oX6FALPT9JLDce80Zz1i
cZZ0MELP37m00Qun0AlmHm/hVzf0f311PtbczqWaE51tJvgUR/nZFo6Ta305Ezhs
3V1EJNQ1Ijf/6DH87SvxAoRIARCuZd0qxBcDK0avpFzUtbJd241RA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpxiegS4fiqC16mFr0dyRk+RJJfIUyz0WTDVm
g0U1CO1ezokMSqkj7724pyjr2xf/r9/sC6a0JwB/lKgZkJfC6NqL7T1xVA31dUga
LE0vEJTTE4gl+tYtfsCdVAlCtqL0jduSkUo+RxxCBiTMxhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9yd11yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6M1I2IdAT/xImu8gbacI8tmyfpIiLnPKiR9D
VFYFGBXuAX7+HgPPSFtrHQONCALxxz1bNpS+zxt9r0MiLgcLyspWxSdmoYgZ6nQP
R05Nm/ZVS+u2imPCRzNUZEMa+d1E6kHx0rS0dPiJ407NtPeYDKkoQtNagspsDvh
ck7CsqAiKMq06UBTxqlTSRkm62e0Ctcs3p30eHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+451cCfmcVt94TFNL5HwEUVJpm0gmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNj3bCLuc/jq7ysGq69xiKmT1UeXFm+aojcR05i
zyShIRJZ0GfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNx1f3MdFQE5JJ86rn9

```
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRG1U/LpNSefnvDFTtEIRcp0Hc  
bhayG0bk51Bd4mio0XnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj20+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
```

You may close the text editor.

- e. Import the file with the Amazon ECS PGP public key with the following command in the terminal.

```
gpg --import <public_key_filename.txt>
```

- f. 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.

macOS

```
curl -Lo ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-  
amd64-latest.asc
```

Linux

```
curl -Lo ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-  
amd64-latest.asc
```

Windows

```
Invoke-WebRequest -OutFile ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/  
ecs-cli-windows-amd64-latest.exe.asc
```

- g. Verify the signature.

macOS and Linux

```
gpg --verify ecs-cli.asc /usr/local/bin/ecs-cli
```

Windows

```
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](#).

3. Apply execute permissions to the binary.

macOS and Linux

```
sudo chmod +x /usr/local/bin/ecs-cli
```

Windows

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:

```
setx path "%path%;C:\Program Files\Amazon\ECSCLI"
```

Restart PowerShell so the changes go into effect.

Note

After the PATH variable is set, the Amazon ECS CLI can be used from either Windows PowerShell or the command prompt.

4. Verify that the CLI is working properly.

```
ecs-cli --version
```

Proceed to [Configuring the Amazon ECS CLI \(p. 64\)](#).

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. For more information, see [Configuring the Amazon ECS CLI \(p. 64\)](#).

Configuring the Amazon ECS CLI

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

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
```

Using 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.

Using 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.

Understanding the 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. Amazon 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)

Amazon ECS on AWS Fargate

AWS Fargate is a technology that you can use with Amazon ECS to run [containers](#) without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task. You configure your task definitions for Fargate by setting the `requiresCompatibilities` task definition parameter to `FARGATE`. For more information, see [see Launch types \(p. 799\)](#).

Fargate offers platform versions for Amazon Linux 2 and Microsoft Windows 2019 Server Full and Core editions. Unless otherwise specified, the information on this page applies to all Fargate platforms.

This topic describes the different components of Fargate tasks and services, and calls out special considerations for using Fargate with Amazon ECS.

For information about the Regions that support Linux containers on Fargate, see [the section called "Supported Regions for Linux containers on AWS Fargate" \(p. 540\)](#).

For information about the Regions that support Windows containers on Fargate, see [the section called "Supported Regions for Windows containers on AWS Fargate" \(p. 542\)](#).

Task definitions

Amazon ECS tasks on AWS Fargate do not support all of the 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:

- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `extraHosts`
- `gpu`
- `ipcMode`
- `links`
- `placementConstraints`
- `privileged`
- `maxSwap`
- `swappiness`

The following task definition parameters are valid in Fargate tasks, but have limitations that should be noted:

- `linuxParameters` – When specifying Linux-specific options that are applied to the container, for capabilities the only capability you can add is `CAP_SYS_PTRACE`. The devices,

`sharedMemorySize`, and `tmpfs` parameters are not supported. For more information, see [Linux parameters \(p. 827\)](#).

- `volumes` – Fargate tasks only support bind mount host volumes, so the `dockerVolumeConfiguration` parameter is not supported. For more information, see [Volumes \(p. 836\)](#).
- `cpu` - For Windows containers on AWS Fargate, the value cannot be less than 1 vCPU.

To ensure that your task definition validates for use with Fargate, 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, specify the `--requires-compatibilities` option.
- In the Amazon ECS API, specify the `requiresCompatibilities` flag.

Network mode

Amazon ECS task definitions for AWS Fargate require that the network mode is set to `awsvpc`. The `awsvpc` network mode provides each task with its own elastic network interface. For more information, see [AWS Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

A network configuration is also required when creating a service or manually running tasks. For more information, see [AWS Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Task Operating Systems

When you configure a task and container definition for AWS Fargate, you must specify the Operating System that the container runs. The following Operating Systems are supported for AWS Fargate:

- Amazon Linux 2
 - Note**
Note that Linux containers use only the kernel and kernel configuration from the host Operating System. For example, the kernel configuration includes the `sysctl` system controls. A Linux container image can be made from a base image that contains the files and programs from any Linux distribution. If the CPU architecture matches, you can run containers from any Linux container image on any Operating System.
 - Amazon Linux 2023
 - Windows Server 2019 Full
 - Windows Server 2019 Core
 - Windows Server 2022 Full
 - Windows Server 2022 Core

Task CPU architecture

There are 2 architectures available for the Amazon ECS task definition, `ARM` and `X86_64`.

When you run Windows containers on AWS Fargate, you must have the `X86_64` CPU architecture.

When you run Linux containers on AWS Fargate, you can use the `X86_64` CPU architecture, or the `ARM64` architecture for your ARM-based applications. For more information, see [the section called “Working with 64-bit ARM workloads on Amazon ECS” \(p. 159\)](#).

Task CPU and memory

Amazon ECS task definitions for AWS Fargate 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	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

Task resource limits

Amazon ECS task definitions for Linux containers on AWS Fargate support the `ulimits` parameter to define the resource limits to set for a container.

Amazon ECS task definitions for Windows on AWS Fargate do not support the `ulimits` parameter to define the resource limits to set for a container.

Amazon ECS tasks hosted on Fargate use the default resource limit values set by the operating system with the exception of the `nofile` resource limit parameter. The `nofile` resource limit sets a restriction on the number of open files that a container can use. On Fargate, the default `nofile` soft limit is 1024 and hard limit is 4096. You can set the values of both limits up to 1048576.

The following is an example task definition snippet that shows how to define a custom `nofile` limit that has been doubled:

```
"ulimits": [
  {
    "name": "nofile",
    "softLimit": 2048,
```

```
        "hardLimit": 8192
    }
]
```

For more information on the other resource limits that can be adjusted, see [Resource limits \(p. 825\)](#).

Logging

Event logging

Amazon ECS logs the actions that it takes to EventBridge. You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters, services, and tasks. Additionally, you can automate actions to respond to these events. For more information, see [Amazon ECS events and EventBridge \(p. 558\)](#).

Task lifecycle logging

Tasks that run on Fargate publish timestamps to track the task through the states of the task lifecycle. You can see the timestamps in the task details in the AWS Management Console and by describing the task in the AWS CLI and SDKs. For example, you can use the timestamps to evaluate how much time the task spent downloading the container images and decide if you should optimize the container image size, or use Seekable OCI indexes. For more information about container image practices, see [Container images \(p. 87\)](#).

Application logging

Amazon ECS task definitions for AWS Fargate support the awslogs, splunk, and awsfirelens log drivers for the log configuration.

The awslogs log driver 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 a task definition to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 161\)](#).

For more information about the awsfirelens log driver in a task definition, see [Custom log routing \(p. 166\)](#).

For more information about using the splunk log driver in a task definition, see [Example: splunk log driver \(p. 215\)](#).

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 calling CloudWatch

to store container application logs. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Example Amazon Linux 2 task definition

The following is an example task definition that sets up a web server using the Fargate launch type with an Amazon Linux 2 operating system:

```
{  
    "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",  
        "runtimePlatform": {  
            "operatingSystemFamily": "LINUX"  
        },  
        "memory": "512",  
        "networkMode": "awsvpc",  
        "requiresCompatibilities": [  
            "FARGATE"  
        ]  
    }  
}
```

Example Windows task definition

The following is an example task definition that sets up a web server using the Fargate launch type with a Windows 2019 Server operating system.

```
{
```

```

"containerDefinitions": [
    {
        "command": [
            "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<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>'; C:\\ServiceMonitor.exe w3svc"
        ],
        "entryPoint": [
            "powershell",
            "-Command"
        ],
        "essential": true,
        "cpu": 2048,
        "memory": 4096,
        "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/fargate-windows-task-definition",
                "awslogs-region": "us-east-1",
                "awslogs-stream-prefix": "ecs"
            }
        },
        "name": "sample_windows_app",
        "portMappings": [
            {
                "hostPort": 80,
                "containerPort": 80,
                "protocol": "tcp"
            }
        ]
    },
    {
        "memory": "4096",
        "cpu": "2048",
        "networkMode": "awsvpc",
        "family": "windows-simple-iis-2019-core",
        "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
        "runtimePlatform": {
            "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
        },
        "requiresCompatibilities": [
            "FARGATE"
        ]
    }
]
}

```

Task storage

For Amazon ECS tasks hosted on Fargate, the following storage types are supported:

- Amazon EFS volumes for persistent storage. For more information, see [Amazon EFS volumes \(p. 102\)](#).
- Bind mounts for ephemeral storage. For more information, see [Bind mounts \(p. 114\)](#).

Tasks and services

After you have your Amazon ECS task definitions for AWS Fargate prepared, there are some decisions to make when creating your service.

Task networking

Amazon ECS tasks for AWS Fargate require the awsvpc network mode, which provides each task with an elastic network interface. When you run a task or create a service with this network mode, you must specify one or more subnets to attach the network interface and one or more security groups to apply to the network interface.

If you are using public subnets, decide whether to provide a public IP address for the network interface. For a Fargate task in a public subnet to pull container images, 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. For a Fargate task in a private subnet to pull container images, you need a NAT gateway in the subnet to route requests to the internet. When you host your container images in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint. In this case, the task's private IPv4 address is used for the image pull. For more information about Amazon ECR interface endpoints, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

The following is an example of the networkConfiguration section for a Fargate service:

```
"networkConfiguration": {  
    "awsvpcConfiguration": {  
        "assignPublicIp": "ENABLED",  
        "securityGroups": [ "sg-12345678" ],  
        "subnets": [ "subnet-12345678" ]  
    }  
}
```

Fargate Spot

Amazon ECS capacity providers enable you to use both AWS Fargate and Fargate Spot capacity with your Amazon ECS tasks.

Windows containers on AWS Fargate cannot use the Fargate Spot capacity provider.

With Fargate Spot you can run interruption tolerant Amazon ECS tasks at a discounted rate compared to the AWS Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks will be interrupted with a two-minute warning. For more information, see [AWS Fargate capacity providers \(p. 222\)](#).

Service load balancing

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

Amazon ECS services on AWS Fargate support the Application Load Balancer and Network Load Balancer load balancer types. Application Load Balancers are used to route HTTP/HTTPS (or layer 7) traffic. Network Load Balancers are used to route TCP or UDP (or layer 4) traffic. For more information, see [Load balancer types \(p. 487\)](#).

When you create a target group 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 \(p. 486\)](#).

Using a Network Load Balancer to route UDP traffic to your Amazon ECS on AWS Fargate tasks is only supported when using platform version 1.4 or later.

Cluster differences with AWS Fargate

Clusters may contain tasks using both the Fargate and EC2 launch types. When viewing your clusters in the AWS Management Console, Fargate and EC2 task counts are displayed separately.

For more information about Amazon ECS clusters, including a walkthrough for creating a cluster, see [Amazon ECS clusters \(p. 220\)](#).

Usage metrics

You can use CloudWatch usage metrics to provide visibility into your accounts usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about AWS Fargate service quotas, see [AWS Fargate service quotas \(p. 538\)](#).

For more information about AWS Fargate usage metrics, see [AWS Fargate usage metrics](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Task maintenance

When AWS determines that a security or infrastructure update is needed for an Amazon ECS task hosted on AWS Fargate, the tasks need to be stopped and new tasks launched to replace them. For more information, see [Task maintenance](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The following table describes these scenarios.

Task type	Issue	Action
Standalone task	Host issue	A task retirement notice is sent using your AWS Health Dashboard and email. If no action is taken by the task retirement date, AWS stops the task.
	Security vulnerability	A task retirement notice is sent using your AWS Health Dashboard and email. If no action is taken by the task retirement date, AWS stops the task.
Service task	Host issue	The task is stopped by AWS and the service scheduler will launch a new task in an attempt to maintain the service's desired count. No notification is sent.
	Security vulnerability	A task retirement notice is sent using your AWS Health Dashboard and email. If no

Task type	Issue	Action
		action is taken by the task retirement date, AWS stops the task and the service scheduler will launch a new task in an attempt to maintain the service's desired count.

Savings plans

Savings Plans are a pricing model that offer significant savings on AWS usage. You commit to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years, and receive a lower price for that usage. For more information, see the [Savings Plans User Guide](#).

To create a Savings Plan for your AWS Fargate usage, use the **Compute Savings Plans** type. To get started, see [Getting started with Savings Plans](#) in the *Savings Plans User Guide*.

Lazy loading container images using Seekable OCI (SOCI)

Amazon ECS tasks on Fargate that use Linux platform version 1.4.0 can use Seekable OCI (SOCI) to help start tasks faster. With SOCI, containers only spend a few seconds on the image pull before they can start, providing time for environment setup and application instantiation while the image is downloaded in the background. This is called *lazy loading*. When Fargate starts an Amazon ECS task, Fargate automatically detects if a SOCI index exists for an image in the task and starts the container without waiting for the entire image to be downloaded.

For containers that run without SOCI indexes, container images are downloaded completely before the container is started. This behavior is the same on all other platform versions of Fargate and on the Amazon ECS-optimized AMI on Amazon EC2 instances.

Seekable OCI indexes

Seekable OCI (SOCI) is an open source technology developed by AWS that can launch containers faster by lazily loading the container image. SOCI works by creating an index (SOCI Index) of the files within an existing container image. This index helps to launch containers faster, providing the capability to extract an individual file from a container image before downloading the entire image. For more information, see [Introducing Seekable OCI for lazy loading container images](#).

Considerations

If you want Fargate to use a SOCI index to lazily load container images in a task, consider the following:

- Only tasks that run on Linux platform version 1.4.0 can use SOCI indexes. Tasks that run Windows containers on Fargate aren't supported.
- Tasks that run on X86_64 and ARM64 CPU architecture are supported.
- All container images in the task definition must have SOCI indexes in the same container registry as each image.
- All container images in the task definition must be stored in a compatible image registry. The following lists the compatible registries:
 - Amazon ECR private registries.

- Only container images that use gzip compression or are not compressed are supported. Container images that use zstd compression aren't supported.
- We recommend that you try lazy loading with container images greater than 250 MiB in size. You are less likely to see a reduction in the time to load smaller images.
- Because lazy loading can change how long your tasks take to start, you might need to change various timeouts like the health check grace period for Elastic Load Balancing.
- Services that use Service Connect are not supported because all container images in the task definition must have SOCI indexes in the same container registry as each image.
- All container images in a task must qualify by the other considerations to be lazily loaded. If you want to prevent a container image from being lazy loaded, delete the SOCI index from the container registry. If any container image in the task doesn't meet one of the considerations, the container images in the task are downloaded by the default method.

Creating a Seekable OCI index

To create SOCI indexes, first you create the index of the files in the container image. To do this, you can use the open source CLI tool [awslabs/soci-snapshotter](#) on GitHub. Second, you push the artifact and the image to a compatible image registry, like Amazon ECR. Or, you can deploy the following solution with AWS CloudFormation to automatically create and push the index when images are pushed to Amazon ECR. For more information about the solution and the installation steps, see [CFN AWS SOCI Index Builder on AWS](#) on GitHub. The AWS CloudFormation template is an easier way to automate getting started with SOCI, while the open source build tool has more flexibility around index generation and the ability to integrate index generation in your continuous integration and continuous delivery (CI/CD) pipelines.

Note

For the SOCI index to be created for an image, the image must exist in the containerd image store on the computer running `soci-snapshotter`. If the image is in the Docker image store, the image can't be found.

Verifying that a task used lazy loading

To verify that a task was lazily loaded using SOCI, check the task metadata endpoint from inside the task. When you query the task metadata endpoint version 4, there is a `Snapshotter` field in the default path for the container that you are querying from. Additionally, there are `Snapshotter` fields for each container in the `/task` path. The default value for this field is `overlayfs`, and this field is set to `soci` if SOCI is used. For more information, see [Task metadata endpoint version 4](#) in the Amazon Elastic Container Service User Guide for AWS Fargate.

Windows containers on AWS Fargate considerations

Windows containers on AWS Fargate supports the following operating systems:

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

AWS handles the operating system license management, so you do not need any additional Microsoft Windows Server licenses.

Windows containers on AWS Fargate supports the awslogs driver. For more information, see [the section called "Using the awslogs log driver" \(p. 161\)](#).

Your tasks can run either Linux containers or Windows containers. If you need run both container types, you must create separate tasks.

The following features are not supported on Windows containers on Fargate:

- Group managed service accounts (gMSA)
- Amazon FSx
- ENI trunking
- App Mesh service and proxy integration for tasks
- Firelens log router integration for tasks
- EFS volumes
- The following task definition parameters:
 - maxSwap
 - swappiness
- The Fargate Spot capacity provider
- Image volumes

The Dockerfile volume option is ignored. Instead, use bind mounts in your task definition. For more information, see [Bind mounts \(p. 114\)](#).

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. You select a platform version when you run a task or when you create a service to maintain a number of identical tasks.

New revisions of 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. A Fargate platform version is updated by making a new platform version revision. Each task runs on one platform version revision during its lifecycle. If you want to use the latest platform version revision, then you must start a new task. A new task that runs on Fargate always runs on the latest revision of a platform version, ensuring that tasks are always started on secure and patched infrastructure.

If a security issue is found that affects an existing platform version, AWS creates a new patched revision of the platform version and retires tasks running on the vulnerable revision. In some cases, you may be notified that your tasks on Fargate have been scheduled for retirement. For more information, see [AWS Fargate task maintenance \(p. 451\)](#).

Topics

- [Linux platform versions \(p. 77\)](#)
- [Windows platform versions \(p. 81\)](#)

Linux platform versions

Platform version considerations

Consider the following when specifying a platform version:

- When specifying a platform version, you can use either a specific version number, for example 1.4.0, or LATEST.

When the **LATEST** platform version is selected, 1.4.0 platform version is used.

- If you want to update the platform version for a service, create a deployment. For example, assume that you have a service that runs tasks on the Linux platform version 1.3.0. To change the service to run tasks on the Linux platform version 1.4.0, you can update your service and specify a new platform version. Your tasks are redeployed with the latest platform version and the latest platform version revision. For more information about deployments, see [Amazon ECS Deployment types \(p. 472\)](#).
- 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. For example, assume that you have a service that runs tasks on the Linux platform version 1.3.0. If you increase the desired count of the service, the service scheduler starts the new tasks using the latest platform version revision of platform version 1.3.0.
- New tasks always run on the latest revision of a platform version, ensuring that tasks are always started on secured and patched infrastructure.
- The platform version numbers for Linux containers and Windows containers on Fargate are independent. For example, the behavior, features, and software used in platform version 1.0.0 for Windows containers on Fargate aren't comparable to those of platform version 1.0.0 for Linux containers on Fargate.

The following are the available Linux platform versions. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 81\)](#).

1.4.0

The following is the changelog for platform version 1.4.0.

- Beginning on November 5, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will be able to use the following features:
 - When using Secrets Manager to store sensitive data, you can inject a specific JSON key or a specific version of a secret as an environment variable or in a log configuration. For more information, see [Passing sensitive data to a container \(p. 200\)](#).
 - Specify environment variables in bulk using the `environmentFiles` container definition parameter. For more information, see [Passing environment variables to a container \(p. 197\)](#).
 - Tasks run in a VPC and subnet enabled for IPv6 will be assigned both a private IPv4 address and an IPv6 address. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
 - The task metadata endpoint version 4 provides additional metadata about your task and container including the task launch type, the Amazon Resource Name (ARN) of the container, and the log driver and log driver options used. When querying the `/stats` endpoint you also receive network rate stats for your containers. For more information, see [Task metadata endpoint version 4](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Beginning on July 30, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will be able to route UDP traffic using a Network Load Balancer to their Amazon ECS on Fargate tasks. For more information, see [Service load balancing \(p. 486\)](#).
- Beginning on May 28, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will have its ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS owned encryption key. For more information, see [Fargate task storage \(p. 101\)](#).
- Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see [Amazon EFS volumes \(p. 102\)](#).
- The ephemeral task storage has been increased to a minimum of 20 GB for each task. For more information, see [Fargate task storage \(p. 101\)](#).
- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs.

For more information, see [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

- Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC.
- CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
- Added support for the task metadata endpoint version 4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see [Task metadata endpoint version 4 \(p. 380\)](#).
- Added support for the SYS_PTRACE Linux parameter in container definitions. For more information, see [Linux parameters \(p. 827\)](#).
- The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. Usually, this change does not have an effect on how your tasks run.
- The container runtime is now using Containerd instead of Docker. Most likely, this change does not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime changes from mentioning Docker to more general errors. For more information, see [Stopped tasks error codes](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Based on Amazon Linux 2.

1.3.0

The following is the changelog for platform version 1.3.0.

- Beginning on Sept 30, 2019, any new Fargate task that is launched supports the awsfirelens log driver. Configure the FireLens for Amazon ECS to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. For more information, see [Custom log routing \(p. 166\)](#).
- Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. For more information, [Task maintenance](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Beginning on March 27, 2019, any new Fargate task that is launched can use additional task definition parameters that you use to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see [Proxy configuration \(p. 835\)](#), [Container dependency \(p. 830\)](#), and [Container timeouts \(p. 831\)](#).
- Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see [Passing sensitive data to a container \(p. 200\)](#).
- Beginning on May 1, 2019, any new Fargate task that is launched supports referencing sensitive data in the log configuration of a container using the secretOptions container definition parameter. For more information, see [Passing sensitive data to a container \(p. 200\)](#).
- Beginning on May 1, 2019, any new Fargate task that is launched supports the splunk log driver in addition to the awslogs log driver. For more information, see [Storage and logging \(p. 819\)](#).
- Beginning on July 9, 2019, any new Fargate tasks that is launched supports CloudWatch Container Insights. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
- Beginning on December 3, 2019, the Fargate Spot capacity provider is supported. For more information, see [AWS Fargate capacity providers \(p. 222\)](#).

- Based on Amazon Linux 2.

1.2.0

The following is the changelog for platform version 1.2.0.

Note

Platform version 1.2.0 is no longer available. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 81\)](#).

- Added support for private registry authentication using AWS Secrets Manager. For more information, see [Private registry authentication for tasks \(p. 195\)](#).

1.1.0

The following is the changelog for platform version 1.1.0.

Note

Platform version 1.1.0 is no longer available. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 81\)](#).

- Added support for the Amazon ECS task metadata endpoint. For more information, see [Amazon ECS task metadata endpoint \(p. 380\)](#).
- Added support for Docker health checks in container definitions. For more information, see [Health check \(p. 811\)](#).
- Added support for Amazon ECS service discovery. For more information, see [Service discovery \(p. 522\)](#).

1.0.0

The following is the changelog for platform version 1.0.0.

Note

Platform version 1.0.0 is no longer available. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 81\)](#).

- Based on Amazon Linux 2017.09.
- Initial release.

Migrating to platform version 1.4.0

Consider the following when migrating your Amazon ECS on Fargate tasks from platform version 1.0.0, 1.1.0, 1.2.0, or 1.3.0 to platform version 1.4.0. It is considered best practice to confirm your task works properly on platform version 1.4.0 prior to migrating your tasks.

- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Amazon ECS on Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- If you are using interface VPC endpoints, consider the following.
 - When using container images hosted with Amazon ECR, both the `com.amazonaws.region.ecr.dkr` and `com.amazonaws.region.ecr.api` Amazon ECR VPC endpoints as well as the Amazon S3 gateway

endpoint are required. For more information, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

- When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- When using a task definition that references Systems Manager Parameter Store parameters to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Systems Manager. For more information, see [Using Systems Manager with VPC endpoints](#) in the *AWS Systems Manager User Guide*.
- Ensure that the security group in the Elastic Network Interface (ENI) associated with your task has the security group rules created to allow traffic between the task and the VPC endpoints you are using.

AWS Fargate platform version deprecation

This page lists platform versions that AWS Fargate has deprecated or have been scheduled for deprecation. These platform versions remain available until the published deprecation date.

A *force update date* is provided for each platform version scheduled for deprecation. On the force update date, any service using the LATEST platform version that is pointed to a platform version that is scheduled for deprecation will be updated using the force new deployment option. When the service is updated using the force new deployment option, all tasks running on a platform version scheduled for deprecation are stopped and new tasks are launched using the platform version that the LATEST tag points to at that time. Standalone tasks or services with an explicit platform version set are not affected by the force update date.

We recommend updating your services standalone tasks to use the most recent platform version. For more information on migrating to the most recent platform version, see [Migrating to platform version 1.4.0 \(p. 80\)](#).

Once a platform version reaches the *deprecation date*, the platform version will no longer be available for new tasks or services. Any standalone tasks or services which explicitly use a deprecated platform version will continue using that platform version until the tasks are stopped. After the deprecation date, a deprecated platform version will no longer receive any security updates or bug fixes.

Platform version	Force update date	Deprecation date
1.0.0	October 26, 2020	December 14, 2020
1.1.0	October 26, 2020	December 14, 2020
1.2.0	October 26, 2020	December 14, 2020

For information about current platform versions, see [AWS Fargate platform versions \(p. 77\)](#).

Windows platform versions

Platform version considerations

Consider the following when specifying a platform version:

- When specifying a platform version, you can use either a specific version number, for example 1.0.0, or LATEST.

When the **LATEST** platform version is selected the **1.0.0** platform is used.

- New tasks always run on the latest revision of a platform version, ensuring that tasks are always started on secured and patched infrastructure.
- Microsoft Windows Server container images must be created from a specific version of Windows Server. You must select the same version of Windows Server in the `platformFamily` when you run a task or create a service that matches the Windows Server container image. Additionally, you can provide a matching `operatingSystemFamily` in the task definition to prevent tasks from being run on the wrong Windows version. For more information, see [Matching container host version with container image versions](#) on the Microsoft Learn website.
- The platform version numbers for Linux containers and Windows containers on Fargate are independent. For example, the behavior, features, and software used in platform version **1.0.0** for Windows containers on Fargate aren't comparable to those of platform version **1.0.0** for Linux containers on Fargate.

The following are the available platform versions for Windows containers.

1.0.0

The following is the changelog for platform version **1.0.0**.

- Initial release for support on the following Microsoft Windows Server operating systems:
 - Windows Server 2019 Full
 - Windows Server 2019 Core
 - Windows Server 2022 Full
 - Windows Server 2022 Core

Amazon ECS task definitions

A *task definition* is a blueprint for your application. It is a text file in JSON format that describes the parameters and one or more containers that form your application.

The following are some of the parameters that you can specify in a task definition:

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

For a complete list of task definition parameters, see [Task definition parameters \(p. 799\)](#).

After you create a task definition, you can run the task definition as a task or a service.

- A *task* is the instantiation of a task definition within a cluster. After you create a task definition for your application within Amazon ECS, you can specify the number of tasks to run on your cluster.
- An Amazon ECS *service* runs and maintains your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

Topics

- [Task definition states \(p. 83\)](#)
- [Architecting your application for Amazon ECS \(p. 85\)](#)
- [Creating a task definition using the console \(p. 125\)](#)
- [Updating a task definition using the console \(p. 139\)](#)
- [Deregistering a task definition revision using the console \(p. 140\)](#)
- [Deleting a task definition revision using the console \(p. 141\)](#)
- [Task definition use cases \(p. 142\)](#)
- [Example task definitions \(p. 213\)](#)

Task definition states

The following are the possible states for a task definition:

ACTIVE

A task definition is ACTIVE after it is registered with Amazon ECS. You can use task definitions in the ACTIVE state to run tasks, or create services.

INACTIVE

A task definition transitions from the ACTIVE state to the INACTIVE state when you deregister a task definition. You can retrieve an INACTIVE task definition by calling `DescribeTaskDefinition`. You cannot run new tasks or create new services with a task definition in the INACTIVE state. There is no impact on existing services or tasks.

DELETE_IN_PROGRESS

A task definition transitions from the INACTIVE state to the DELETE_IN_PROGRESS state after you submitted the task definition for deletion. After the task definition is in the DELETE_IN_PROGRESS state, Amazon ECS periodically verifies that the target task definition is not being referenced by any active tasks or deployments, and then deletes the task definition permanently. You cannot run new tasks or create new services with a task definition in the DELETE_IN_PROGRESS state. A task definition can be submitted for deletion at any moment without causing impacts to existing tasks and services.

Task definitions that are in the DELETE_IN_PROGRESS state can be viewed in the console and you can retrieve the task definition by calling `DescribeTaskDefinition`.

When you delete all INACTIVE task definition revisions, the task definition name is not displayed in the console and not returned in the API. If a task definition revisions is in the DELETE_IN_PROGRESS state, the task definition name is displayed in the console and returned in the API. The task definition name is retained by Amazon ECS and the revision is incremented the next time you create a task definition with that name.

If you use AWS Config to manage your task definitions, AWS Config charges you for all task definition registrations. You are only charged for deregistering the latest ACTIVE task definition. There is no charge for deleting a task definition. For more information about pricing, see [AWS Config Pricing](#).

Amazon ECS resources that can block a deletion

A task definition deletion request will not complete if there are any Amazon ECS resources that depend on the task definition revision. The following resources might prevent a task definition from being deleted:

- Amazon ECS tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS deployments and task sets - The task definition is required when a scaling event is initiated for an Amazon ECS deployment or task set.

If your task definition remains in the DELETE_IN_PROGRESS state, you can use the console, or the AWS CLI to identify, and then stop the resources which block the task definition deletion.

Task definition deletion after the blocked resource is removed

The following rules apply after you remove the resources that block the task definition deletion:

- Amazon ECS tasks - The task definition deletion can take up to 1 hour to complete after the task is stopped.
- Amazon ECS deployments and task sets - The task definition deletion can take up to 24 hours to complete after the deployment or task set is deleted.

Architecting your application for Amazon ECS

Architecting your application for Amazon ECS includes deciding on the launch type (which is determined by your capacity), building your images, and configuring your task definitions (including networking and data storage).

Topics

- [Amazon ECS launch types \(p. 85\)](#)
- [Container images \(p. 87\)](#)
- [Task definitions \(p. 89\)](#)
- [Networking modes \(p. 89\)](#)
- [Using data volumes in tasks \(p. 100\)](#)
- [Managing container swap space \(p. 123\)](#)
- [Amazon EC2 Windows task definition considerations \(p. 124\)](#)

Amazon ECS launch types

Fargate launch type

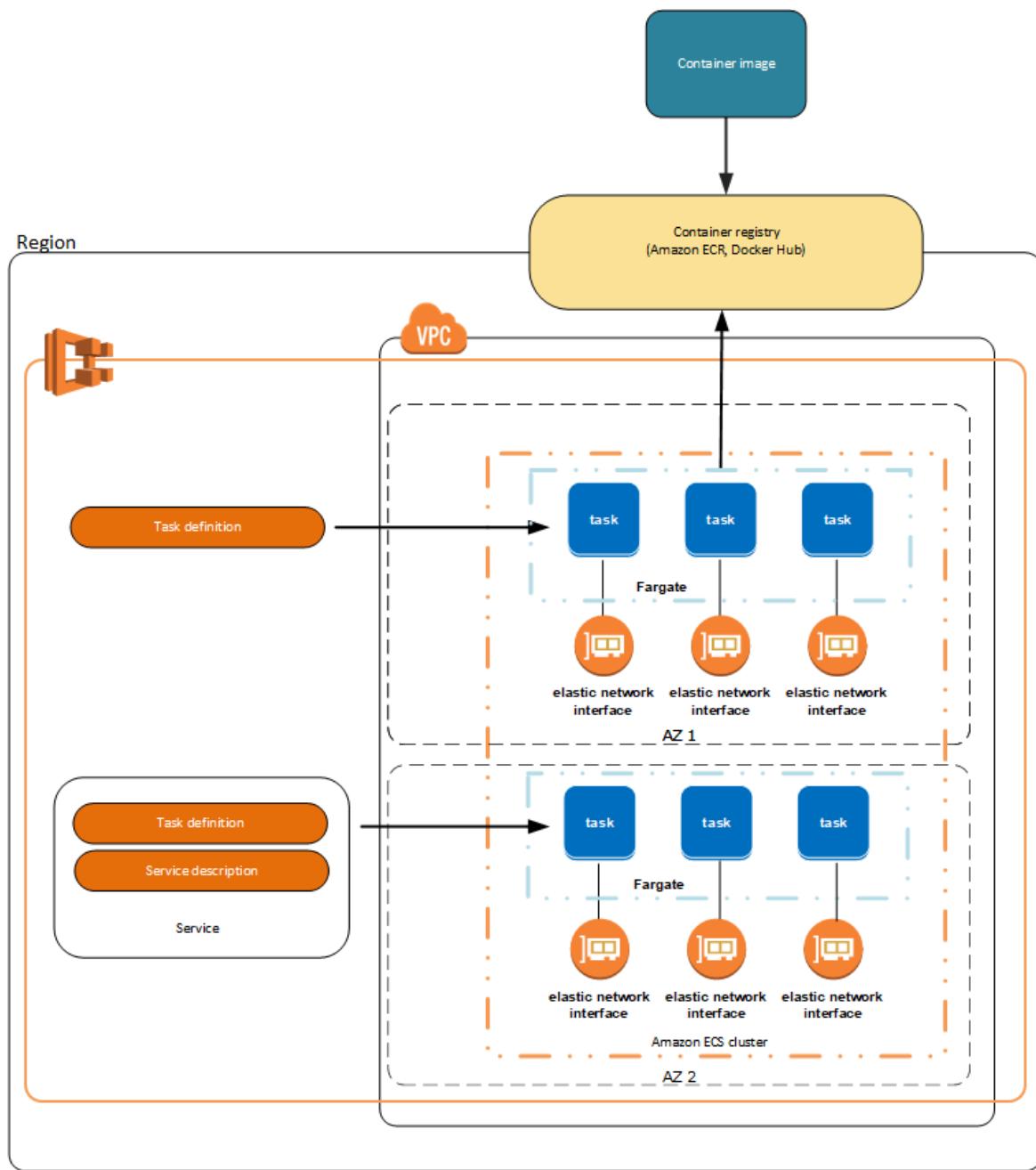
You can use the Fargate launch type to run your containerized applications without the need of provisioning and managing the underlying infrastructure. AWS Fargate is the serverless way to host your Amazon ECS workloads.

The Fargate launch type is suitable for the following workloads:

- Large workloads that require low operational overhead
- Small workloads that have occasional burst
- Tiny workloads
- Batch workloads

For information about the Regions that support Fargate, see [the section called "AWS Fargate Regions" \(p. 540\)](#).

The following diagram shows the general architecture.



For more information about Amazon ECS on Fargate, see [Amazon ECS on AWS Fargate \(p. 67\)](#).

EC2 launch type

The EC2 launch type is suitable for large workloads that must be price optimized.

When considering how to model task definitions and services using the EC2 launch type, we recommend that you consider what processes must run together and how you might go about scaling each component.

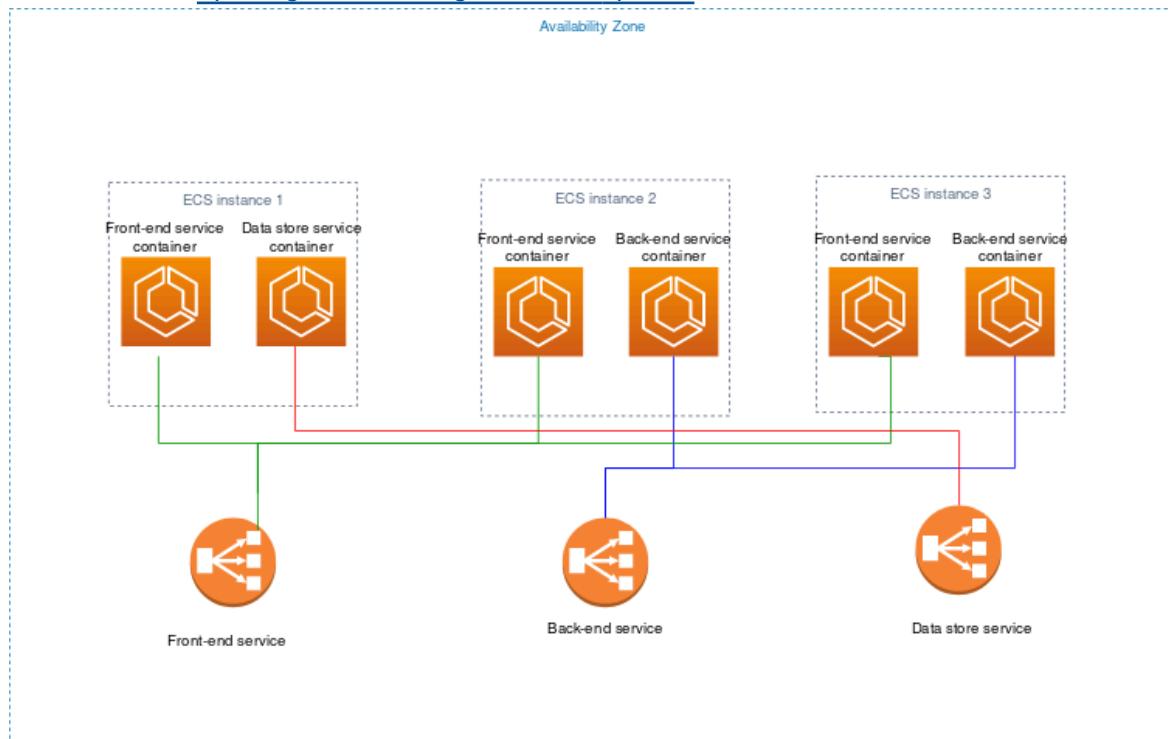
As an example, suppose that an application consists of the following components:

- A frontend service that displays information on a webpage
- A backend service that provides APIs for the frontend service
- A data store

For this example, create task definitions that group the containers that are used for a common purpose together. Separate the different components into multiple and separate task definitions. The following example cluster has three container instances that are running three front-end service containers, two backend service containers, and one data store service container.

You can group related containers in a task definition, such as linked containers that must be run together. For example, add a log streaming container to your front-end service and include it in the same task definition.

After you have your task definitions, you can create services from them to maintain the availability of your desired tasks. For more information, see [Creating an Amazon ECS service in the classic console \(p. 899\)](#). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see [Service load balancing \(p. 486\)](#). When your application requirements change, you can update your services to scale the number of desired tasks up or down. Or, you can update your services to deploy newer versions of the containers in your tasks. For more information, see [Updating a service using the console \(p. 465\)](#).



External launch type

The External launch type is used to run your containerized applications on your on-premise server or virtual machine (VM) that you register to your Amazon ECS cluster and manage remotely. For more information, see [External instances \(Amazon ECS Anywhere\) \(p. 336\)](#).

Container images

A container image is a set of instructions on how to build the container. A container image holds your application code and all the dependencies that your application code requires to run. Application

dependencies include the source code packages that your application code relies on, a language runtime for interpreted languages, and binary packages that your dynamically linked code relies on.

Use the following guidelines when you design and build your container images:

- Make your container images complete by storing all application dependencies as static files inside the container image.

If you want to change something in the container image, build a new container image with the changes applied to it.

- Run a single application process within a container.

The container lifetime is as long as the application process runs.

Crashed processes are replaced through a centralized decision making process in the Amazon ECS control plane. The control plane makes smarter choices about where to launch the replacement process. This makes the overall deployment more resilient than if each individual compute instance attempts to relaunch its own processes locally.

- Handle SIGTERM within the application

When Amazon ECS stops a task, it first sends a SIGTERM signal to the task to notify the application needs to finish and shut down, and then Amazon ECS sends a SIGKILL message. When applications ignore the SIGTERM, the Amazon ECS service must wait to send the SIGKILL signal to terminate the process..

To prepare your application, you need to identify how long it takes your application to complete its work, and ensure that your applications handles the SIGTERM signal. Within the application's signal handling, you need to stop the application from taking new work and complete the work that is in-progress, or save unfinished work to storage outside of the task if it would take too long to complete.

- Configure containerized applications to write logs to `stdout` and `stderr`.

When you decouple log handling from your application code, it gives you greater flexibility to adjust log handling at the infrastructure level. Assume that you want to switch from one logging system to another. You can do so by adjusting a few settings at the container orchestrator level, rather than having to change code in all your services, build a new container image, and deploy it.

- Use tags to version your container images.

Container images are stored in a container registry. Each image in a registry is identified by a tag. There's a tag called `latest`. This tag functions as a pointer to the latest version of the application container image, similar to the `HEAD` in a git repository. We recommend that you use the `latest` tag only for testing purposes. As a best practice, tag container images with a unique tag for each build. We recommend that you tag your images using the git SHA for the git commit that was used to build the image.

You don't need to build a container image for every commit. However, we recommend that you build a new container image each time you release a particular code commit to the production environment. We also recommend that you tag the image with a tag that corresponds to the git commit of the code that's inside the image. If you tagged the image with the git commit, you can more quickly find which version of the code the image is running.

We also recommend that you turn on immutable image tags in Amazon Elastic Container Registry. With this setting, you can't change the container image that a tag points at. Instead Amazon ECR enforces that a new image must be uploaded to a new tag, rather than overwriting a pre-existing tag. For more information, see [Image tag mutability](#) in the *Amazon ECR User Guide*.

In addition to the above list, when you architect your application to run on Amazon ECS using AWS Fargate, you must decide between deploying multiple containers into the same task definition and deploying containers separately in multiple task definitions.

If the following conditions are required, we recommend deploying multiple containers into the same task definition:

- Your containers share a common lifecycle (that is, they're launched and terminated together).
- Your containers must run on the same underlying host (that is, one container references the other on a localhost port).
- You containers share resources.
- Your containers share data volumes.

If these conditions aren't required, we recommend deploying containers separately in multiple task definitions. This is because, by doing so, you can scale, provision, and deprovision them separately.

Task definitions

A task definition is a blueprint for your application. Use the following guidelines when you create your task definitions:

- Use each task definition family for only one business purpose.

If you group multiple types of application container together in the same task definition, you can't independently scale those containers. For example, it's unlikely that both a website and an API require scaling out at the same rate. As traffic increases, there will be a different number of web containers required than API containers. If these two containers are being deployed in the same task definition, every task runs the same number of web containers and API containers.

- Match each application version with a task definition revision within a task definition family.

Within a task definition family, consider each task definition revision as a point in time snapshot of the settings for a particular container image. This is similar to how the container is a snapshot of all the things that are needed to run a particular version of your application code.

Make sure that there's a one-to-one mapping between a version of application code, a container image tag, and a task definition revision. A typical release process involves a git commit that gets turned into a container image that's tagged with the git commit SHA. Then, that container image tag gets its own Amazon ECS task definition revision. Last, the Amazon ECS service is updated to tell it to deploy the new task definition revision.

- Use different IAM roles for each task definition family.

Define each task definition with its own IAM role. This recommendation should be done in tandem with our recommendation for providing each business component its own task definition family. By implementing both of these best practices, you can limit how much access each service has to resources in your AWS account. For example, you can give your authentication service access to connect to your passwords database. At the same time, you can also ensure that only your order service has access to the credit card payment information.

Networking modes

The following sections describe the networking modes and their behavior for tasks hosted on Amazon EC2 instances and Fargate.

Topics

- [Task networking for tasks that are hosted on Amazon EC2 instances \(p. 90\)](#)

- [Task networking for tasks hosted on Fargate \(p. 97\)](#)

Task networking for tasks that are hosted on Amazon EC2 instances

The networking behavior of Amazon ECS tasks that are hosted on Amazon EC2 instances is dependent on the *network mode* that's defined in the task definition. We recommend that you use the `awsvpc` network mode unless you have a specific need to use a different network mode.

The following are the available network modes.

Network mode	Linux containers on EC2	Windows containers on EC2	Description
<code>awsvpc</code>	Yes	Yes	The task is allocated its own elastic network interface (ENI) and a primary private IPv4 address. This gives the task the same networking properties as Amazon EC2 instances.
<code>bridge</code>	Yes	No	The task uses Docker's built-in virtual network on Linux, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Linux uses the bridge Docker network driver. This is the default network mode on Linux if a network mode isn't specified in the task definition.
<code>host</code>	Yes	No	The task uses the host's network which bypasses Docker's built-in virtual network by mapping container ports directly to the ENI of the Amazon EC2 instance that hosts the task. Dynamic port mappings can't be used in this network mode. A container in a task definition that uses this mode must specify a specific <code>hostPort</code> number. A port number on a host can't be used by multiple tasks. As a result, you can't run multiple tasks of the same task definition on a single Amazon EC2 instance.
<code>none</code>	Yes	No	The task has no external network connectivity.
<code>default</code>	No	Yes	The task uses Docker's built-in virtual network on Windows, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Windows uses the nat Docker network driver. This is the default network mode on Windows if a network mode isn't specified in the task definition.

For more information about Docker networking on Linux, see [Networking overview](#) in the *Docker Documentation*.

For more information about Docker networking on Windows, see [Windows container networking](#) in the *Microsoft Containers on Windows Documentation*.

Topics

- [awsVpc network mode \(p. 91\)](#)
- [Host mode \(p. 95\)](#)
- [Bridge mode \(p. 96\)](#)

awsVpc network mode

The task networking features that are provided by the awsVpc network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. Using the awsVpc network mode simplifies container networking, you have more control over how containerized applications communicate with each other and other services within your VPCs. The awsVpc network mode also provides greater security for your containers by enabling you to use security groups and network monitoring tools at a more granular level within your tasks. Because each task gets its own elastic network interface (ENI), you can also use other Amazon EC2 networking features such as VPC Flow Logs to monitor traffic to and from your tasks. Additionally, containers that belong to the same task can communicate over the localhost interface.

The task ENI is a fully managed feature of Amazon ECS. Amazon ECS creates the ENI and attaches it to the host Amazon EC2 instance with the specified security group. The task sends and receives network traffic over the ENI in the same way that Amazon EC2 instances do with their primary network interfaces. Each task ENI is assigned a private IPv4 address by default. If your VPC is enabled for dual-stack mode and you use a subnet with an IPv6 CIDR block, the task ENI will also receive an IPv6 address. Each task can only have one ENI.

These ENIs are visible in the Amazon EC2 console for your account, but they can't be detached manually or modified by your account. This is to prevent accidental deletion of an ENI that is associated with a running task. You can view the ENI 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 task ENI is detached and deleted.

After you enable the awsVpcTrunking account setting and you have launched a container instance with the increased ENI density, Amazon ECS also creates and attaches a "trunk" network interface for your container instance. The trunk network is fully managed by Amazon ECS. The trunk ENI is deleted when you either terminate or deregister your container instance from the Amazon ECS cluster. For more information on opting in to the awsVpcTrunking account setting, see [Working with container instances with increased ENI limits \(p. 285\)](#).

Considerations

There are several things to consider when using the awsVpc network mode.

Linux considerations

Consider the following when using the Linux operating system.

- 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 \(p. 624\)](#). 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
```

- Your Amazon EC2 Linux instance requires version 1.15.0 or later of the container agent to run tasks that use the awsVpc network mode. If you're using an Amazon ECS-optimized AMI, your instance needs at least version 1.15.0-4 of the ecs-init package as well.

- Amazon ECS populates the hostname of the task with an Amazon-provided (internal) DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is set to a random hostname. For more information about the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 instance that hosts it. There's a default quota for the number of network interfaces that can be attached to an Amazon EC2 Linux instance. The primary network interface counts as one toward that quota. For example, by default, a `c5.1large` instance might have only up to three ENIs that can be attached to it. The primary network interface for the instance counts as one. You can attach an additional two ENIs to the instance. Because each task that uses the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information about the default ENI limits for each instance type, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Linux Instances*.
- Amazon ECS supports the launch of Amazon EC2 Linux instances that use supported instance types with increased ENI density. When you opt in to the `awsvpcTrunking` account setting and register Amazon EC2 Linux instances that use these instance types to your cluster, these instances have higher ENI quota. Using these instances with this higher quota means that you can place more tasks on each Amazon EC2 Linux instance. To use the increased ENI density with the trunking feature, your Amazon EC2 instance must use version `1.28.1` or later of the container agent. If you're using an Amazon ECS-optimized AMI, your instance also requires at least version `1.28.1-2` of the `ecs-init` package. For more information about opting in to the `awsvpcTrunking` account setting, see [Account settings \(p. 244\)](#). For more information about ENI trunking, see [Elastic network interface trunking \(p. 283\)](#).
- When hosting tasks that use the `awsvpc` network mode on Amazon EC2 Linux instances, your task ENIs aren't given public IP addresses. To access the internet, tasks must be launched in a private subnet that's configured to use a NAT gateway. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within a VPC that uses the private IP address or routed through a load balancer from within the VPC. Tasks that are launched within public subnets do not have access to the internet.
- Amazon ECS recognizes only the ENIs that it attaches to your Amazon EC2 Linux instances. If you manually attached ENIs to your instances, Amazon ECS might attempt to add a task to an instance that doesn't have enough network adapters. This can result in the task timing out and moving to a deprovisioning status and then a stopped status. We recommend that you don't attach ENIs to your instances manually.
- Amazon EC2 Linux instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the `awsvpc` network mode. Instances running version `1.15.0-4` or later of `ecs-init` are registered with this attribute automatically.
- The ENIs that are created and attached to your Amazon EC2 Linux instances cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an ENI that is associated with a running task. To release the ENIs for a task, stop the task.
- There is a limit of 16 subnets and 5 security groups that are able to be specified in the `awsVpcConfiguration` when running a task or creating a service that uses the `awsvpc` network mode. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- 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 running 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 ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode only support Application Load Balancer and Network Load Balancer. When you create any target groups for these services, you must choose `ip` as the target type. Do not use `instance`. This is because tasks that use the `awsvpc` network mode are

associated with an ENI, not with an Amazon EC2 Linux instance. For more information, see [Service load balancing \(p. 486\)](#).

- If your VPC is updated to change the DHCP options set it uses, you can't apply these changes to existing tasks. Start new tasks with these changes applied to them, verify that they are working correctly, and then stop the existing tasks in order to safely change these network configurations.

Windows considerations

The following are considerations when you use the Windows operating system:

- Container instances using the Amazon ECS optimized Windows Server 2016 AMI can't host tasks that use the awsvpc network mode. If you have a cluster that contains Amazon ECS optimized Windows Server 2016 AMIs and Windows AMIs that support awsvpc network mode, tasks that use awsvpc network mode aren't launched on the Windows 2016 Server instances. Rather, they're launched on instances that support awsvpc network mode.
- Your Amazon EC2 Windows instance requires version 1.57.1 or later of the container agent to use CloudWatch metrics for Windows containers that use the awsvpc network mode.
- 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 \(p. 624\)](#). 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
```

- Your Amazon EC2 Windows instance requires version 1.54.0 or later of the container agent to run tasks that use the awsvpc network mode. When you bootstrap the instance, you must configure the options that are required for awsvpc network mode. For more information, see [the section called "Bootstrap Container Instances" \(p. 333\)](#).
- Amazon ECS populates the hostname of the task with an Amazon provided (internal) DNS hostname when both the enableDnsHostnames and enableDnsSupport options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is a random hostname. For more information about the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the awsvpc network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 Windows instance that hosts it. There is a default quota for the number of network interfaces that can be attached to an Amazon EC2 Windows instance. The primary network interface counts as one toward this quota. For example, by default a c5.large instance might have only up to three ENIs attached to it. The primary network interface for the instance counts as one of those. You can attach an additional two ENIs to the instance. Because each task using the awsvpc network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information about the default ENI limits for each instance type, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Windows Instances*.
- When hosting tasks that use the awsvpc network mode on Amazon EC2 Windows instances, your task ENIs aren't given public IP addresses. To access the internet, launch tasks in a private subnet that's configured to use a NAT gateway. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC that is using the private IP address or routed through a load balancer from within the VPC. Tasks that are launched within public subnets don't have access to the internet.
- Amazon ECS recognizes only the ENIs that it has attached to your Amazon EC2 Windows instance. If you manually attached ENIs to your instances, Amazon ECS might attempt to add a task to an instance that doesn't have enough network adapters. This can result in the task timing out and moving to a deprovisioning status and then a stopped status. We recommend that you don't attach ENIs to your instances manually.

- Amazon EC2 Windows instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the `awsvpc` network mode.
- You can't manually modify or detach ENIs that are created and attached to your Amazon EC2 Windows instances. This is to prevent you from accidentally deleting an ENI that's associated with a running task. To release the ENIs for a task, stop the task.
- You can only specify up to 16 subnets and 5 security groups in `awsVpcConfiguration` when you run a task or create a service that uses the `awsvpc` network mode. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- 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 running 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 ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode only support Application Load Balancer and Network Load Balancer. 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 ENI, not with an Amazon EC2 Windows instance. For more information, see [Service load balancing \(p. 486\)](#).
- If your VPC is updated to change the DHCP options set it uses, you can't apply these changes to existing tasks. Start new tasks with these changes applied to them, verify that they are working correctly, and then stop the existing tasks in order to safely change these network configurations.
- The following are not supported when you use `awsvpc` network mode in an EC2 Windows configuration:
 - Dual-stack configuration
 - IPv6
 - ENI trunking

Enabling task networking

For tasks to use the `awsvpc` network mode, it must be specified in the task definition. For more information, see [Network mode \(p. 800\)](#). Then, when you run a task or create a service, specify a network configuration that includes one or more subnets to place your tasks in. Also specify one or more security groups to attach to an ENI. The tasks are placed on compatible Amazon EC2 instances in the same Availability Zones as those subnets, and the specified security groups are associated with the ENI that's provisioned for the task.

Using a VPC in dual-stack mode

When using a VPC in dual-stack mode, your tasks can communicate over IPv4 or IPv6, or both. IPv4 and IPv6 addresses are independent of each other and you must configure routing and security in your VPC separately for IPv4 and IPv6. For more information about how to configure your VPC for dual-stack mode, see [Migrating to IPv6](#) in the *Amazon VPC User Guide*.

If you configured your VPC with an internet gateway or an outbound-only internet gateway, you can use your VPC in dual-stack mode. By doing this, tasks that are assigned an IPv6 address can access the internet through an internet gateway or an egress-only internet gateway. NAT gateways are optional. For more information, see [Internet gateways](#) and [Egress-only internet gateways](#) in the *Amazon VPC User Guide*.

Amazon ECS tasks are assigned an IPv6 address if the following conditions are met:

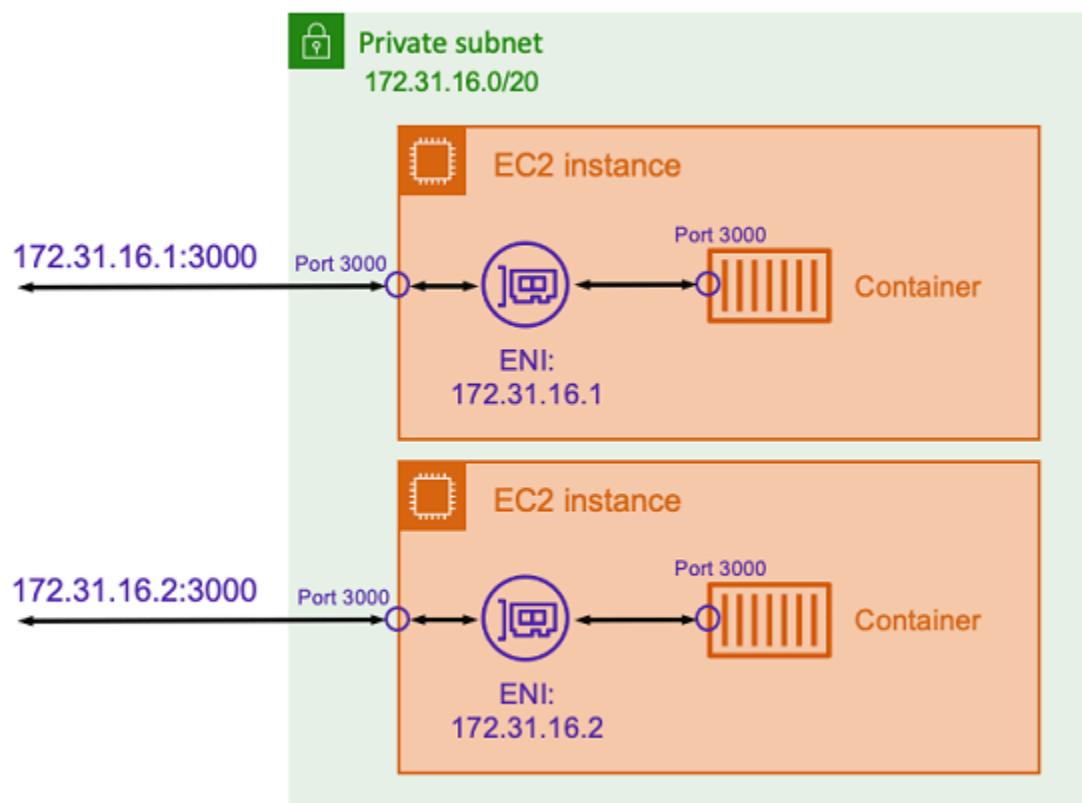
- The Amazon EC2 Linux instance that hosts the task is using version `1.45.0` or later of the container agent. For information about how to check the agent version your instance is using, and updating it if needed, see [Updating the Amazon ECS container agent \(p. 364\)](#).

- The dualStackIPv6 account setting is enabled. For more information, see [Account settings \(p. 244\)](#).
- Your task is using the awsvpc network mode.
- Your VPC and subnet are configured for IPv6. The configuration includes the network interfaces that are created in the specified subnet. For more information about how to configure your VPC for dual-stack mode, see [Migrating to IPv6](#) and [Modify the IPv6 addressing attribute for your subnet](#) in the [Amazon VPC User Guide](#).

Host mode

The host network mode is only supported for Amazon ECS tasks hosted on Amazon EC2 instances. It's not supported when using Amazon ECS on Fargate.

The host network mode is the most basic network mode that's supported in Amazon ECS. Using host mode, the networking of the container is tied directly to the underlying host that's running the container.



Assume that you're running a Node.js container with an Express application that listens on port 3000 similar to the one illustrated in the preceding diagram. When the host network mode is used, the container receives traffic on port 3000 using the IP address of the underlying host Amazon EC2 instance. We do not recommend using this mode.

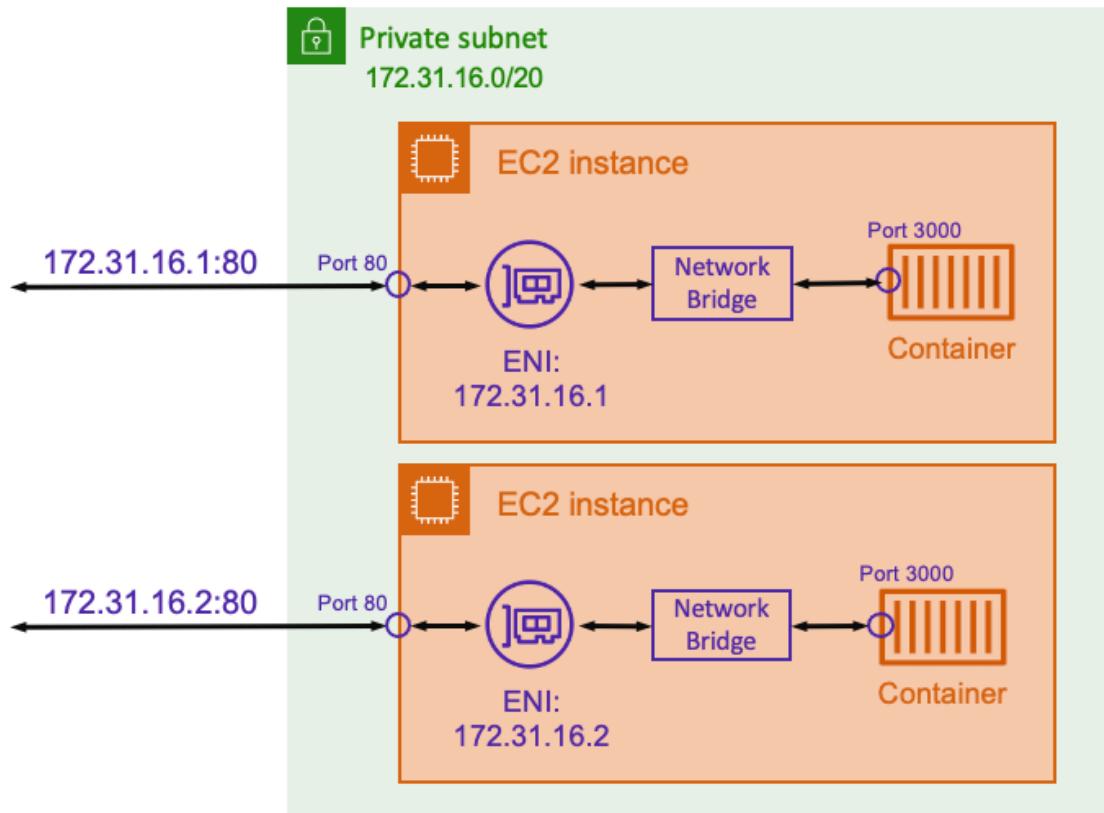
There are significant drawbacks to using this network mode. You can't run more than a single instantiation of a task on each host. This is because only the first task can bind to its required port on the Amazon EC2 instance. There's also no way to remap a container port when it's using host network mode. For example, if an application needs to listen on a particular port number, you can't remap the port number directly. Instead, you must manage any port conflicts through changing the application configuration.

There are also security implications when using the host network mode. This mode allows containers to impersonate the host, and it allows containers to connect to private loopback network services on the host.

Bridge mode

The bridge network mode is only supported for Amazon ECS tasks hosted on Amazon EC2 instances. It is not supported when using Amazon ECS on Fargate.

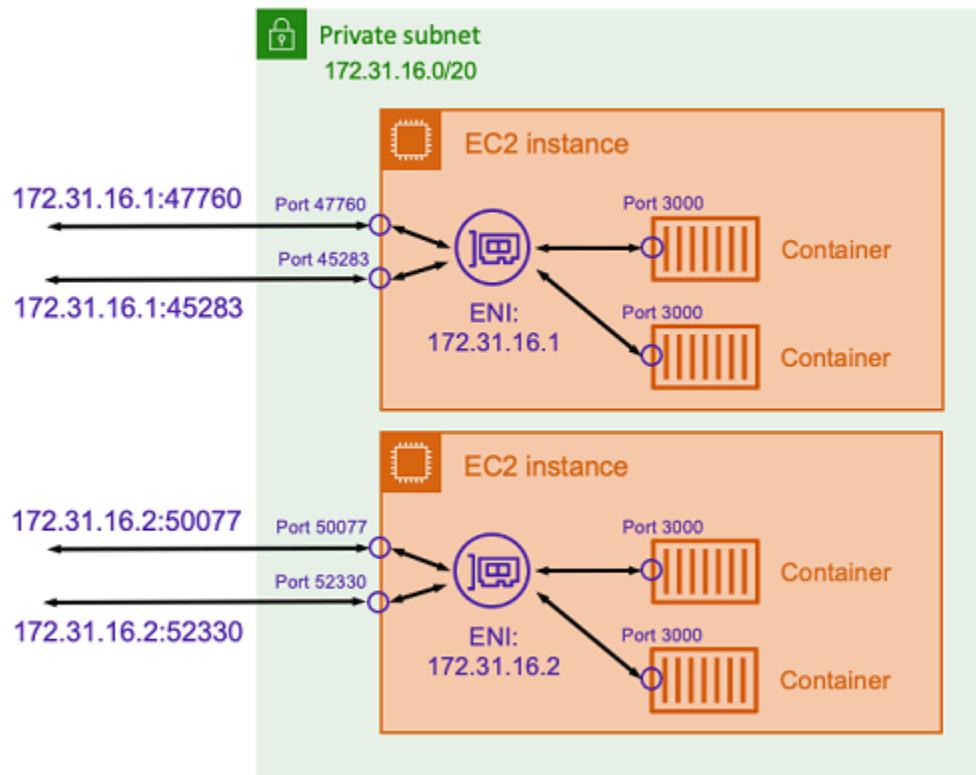
With bridge mode, you're using a virtual network bridge to create a layer between the host and the networking of the container. This way, you can create port mappings that remap a host port to a container port. The mappings can be either static or dynamic.



With a static port mapping, you can explicitly define which host port you want to map to a container port. Using the example above, port 80 on the host is being mapped to port 3000 on the container. To communicate to the containerized application, you send traffic to port 80 to the Amazon EC2 instance's IP address. From the containerized application's perspective it sees that inbound traffic on port 3000.

If you only want to change the traffic port, then static port mappings is suitable. However, this still has the same disadvantage as using the host network mode. You can't run more than a single instantiation of a task on each host. This is because a static port mapping only allows a single container to be mapped to port 80.

To solve this problem, consider using the bridge network mode with a dynamic port mapping as shown in the following diagram.



By not specifying a host port in the port mapping, you can have Docker choose a random, unused port from the ephemeral port range and assign it as the public host port for the container. For example, the Node.js application listening on port 3000 on the container might be assigned a random high number port such as 47760 on the Amazon EC2 host. Doing this means that you can run multiple copies of that container on the host. Moreover, each container can be assigned its own port on the host. Each copy of the container receives traffic on port 3000. However, clients that send traffic to these containers use the randomly assigned host ports.

Amazon ECS helps you to keep track of the randomly assigned ports for each task. It does this by automatically updating load balancer target groups and AWS Cloud Map service discovery to have the list of task IP addresses and ports. This makes it easier to use services operating using bridge mode with dynamic ports.

However, one disadvantage of using the bridge network mode is that it's difficult to lock down service to service communications. Because services might be assigned to any random, unused port, it's necessary to open broad port ranges between hosts. However, it's not easy to create specific rules so that a particular service can only communicate to one other specific service. The services have no specific ports to use for security group networking rules.

Task networking for tasks hosted on Fargate

By default, every Amazon ECS task on Fargate is provided an elastic network interface (ENI) with a primary private IP address. When using a public subnet, you can optionally assign a public IP address to the task's ENI. If your VPC is enabled for dual-stack mode and you use a subnet with an IPv6 CIDR block, your task's ENI also receives an IPv6 address. A task can only have one ENI that's associated with it at a time. Containers that belong to the same task can also communicate over the localhost interface. For more information about VPCs and subnets, see [VPCs and subnets](#) in the *Amazon VPC User Guide*.

For a task on Fargate to pull a container image, the task must have a route to the internet. The following describes how you can verify that your task has a route to the internet.

- When using a public subnet, you can assign a public IP address to the task ENI.
- When using a private subnet, the subnet can have a NAT gateway attached.
- When using container images that are hosted in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint and the image pull occurs over the task's private IPv4 address. For more information, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

Because each task gets its own ENI, you can use networking features such as VPC Flow Logs, which you can use to monitor traffic to and from your tasks. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You can also take advantage of AWS PrivateLink. You can configure a VPC interface endpoint so that you can access Amazon ECS APIs through private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway. For more information, see [AWS PrivateLink](#) in the *Amazon ECS Best Practices Guide*.

For examples of how to use the NetworkConfiguration resource with AWS CloudFormation, see [the section called "Creating Amazon ECS resources using separate stacks" \(p. 53\)](#).

The ENIs that are created are fully managed by AWS Fargate. Moreover, there's an associated IAM policy that's used to grant permissions for Fargate. For tasks using Fargate platform version 1.4.0 or later, the task receives a single ENI (referred to as the task ENI) and all network traffic flows through that ENI within your VPC. This traffic is recorded in your VPC flow logs. For tasks that use Fargate platform version 1.3.0 and earlier, in addition to the task ENI, the task also receives a separate Fargate owned ENI, which is used for some network traffic that isn't visible in the VPC flow logs. The following table describes the network traffic behavior and the required IAM policy for each platform version.

Action	Traffic flow with Linux platform version 1.3.0 and earlier	Traffic flow with Linux platform version 1.4.0	Traffic flow with Windows platform version 1.0.0	IAM permission
Retrieving Amazon ECR login credentials	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role
Image pull	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through a log driver	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through FireLens for Amazon ECS	Task ENI	Task ENI	Task ENI	Task IAM role
Retrieving secrets from Secrets Manager or Systems Manager	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role

Action	Traffic flow with Linux platform version 1.3.0 and earlier	Traffic flow with Linux platform version 1.4.0	Traffic flow with Windows platform version 1.0.0	IAM permission
Amazon EFS file system traffic	Not available	Task ENI	Task ENI	Task IAM role
Application traffic	Task ENI	Task ENI	Task ENI	Task IAM role

Fargate task networking considerations

Consider the following when using task networking.

- The Amazon ECS service-linked role is required to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you 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 \(p. 624\)](#). You can also create the service-linked role using the following AWS CLI command.

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

- Amazon ECS populates the hostname of the task with an Amazon provided DNS hostname when both the enableDnsHostnames and enableDnsSupport options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is set to a random hostname. For more information about the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.
- You can only specify up to 16 subnets and 5 security groups for awsVpcConfiguration. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- You can't manually detach or modify the ENIs that are created and attached by Fargate. This is to prevent the accidental deletion of an ENI that's associated with a running task. To release the ENIs for a task, stop the task.
- If a VPC subnet is updated to change the DHCP options set it uses, you can't also apply these changes to existing tasks that use the VPC. Start new tasks, which will receive the new setting to smoothly migrate while testing the new change and then stop the old ones, if no rollback is required.
- Tasks that are launched in subnets with IPv6 CIDR blocks only receive an IPv6 address when using Fargate platform version 1.4.0 or later for Linux or 1.0.0 for Windows.
- For tasks that use platform version 1.4.0 or later for Linux or 1.0.0 for Windows, the task ENIs support jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames reduces overhead when the network path between your task and the destination supports jumbo frames.
- Services with tasks that use the Fargate launch type only support Application Load Balancer and Network Load Balancer. Classic Load Balancer isn't supported. When you create any target groups, you must choose ip as the target type, not instance. For more information, see [Service load balancing \(p. 486\)](#).

Using a VPC in dual-stack mode

When using a VPC in dual-stack mode, your tasks can communicate over IPv4 or IPv6, or both. IPv4 and IPv6 addresses are independent of each other and you must configure routing and security in your VPC separately for IPv4 and IPv6. For more information about configuring your VPC for dual-stack mode, see [Migrating to IPv6](#) in the *Amazon VPC User Guide*.

If the following conditions are met, Amazon ECS tasks on Fargate are assigned an IPv6 address:

- Your VPC and subnet are enabled for IPv6. For more information about how to configure your VPC for dual-stack mode, see [Migrating to IPv6](#) in the *Amazon VPC User Guide*.
- Your subnet is enabled for auto-assigning IPv6 addresses. For more information about how to configure your subnet, see [Modify the IPv6 addressing attribute for your subnet](#) in the *Amazon VPC User Guide*.
- The task or service uses Fargate platform version 1.4.0 or later for Linux.
- The dualStackIPv6 account setting is enabled. For more information, see [Account settings \(p. 244\)](#).

If you configure your VPC with an internet gateway or an outbound-only internet gateway, Amazon ECS tasks on Fargate that are assigned an IPv6 address can access the internet. NAT gateways aren't needed. For more information, see [Internet gateways](#) and [Egress-only internet gateways](#) in the *Amazon VPC User Guide*.

Using data volumes in tasks

Amazon ECS supports the following data volume options for containers.

- Amazon EFS volumes — Provides simple, scalable, and persistent file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage that they need and when they need it. Amazon EFS volumes are supported for tasks that are hosted on Fargate or Amazon EC2 instances. For more information, see [Amazon EFS volumes \(p. 102\)](#).
- FSx for Windows File Server volumes — Provides fully managed Microsoft Windows file servers. These file servers are backed by a Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, and static file storage. For more information, see [FSx for Windows File Server volumes \(p. 105\)](#).

Windows containers on Fargate do not support this option.

- Docker volumes — A Docker-managed volume that's created under `/var/lib/docker/volumes` on the host Amazon EC2 instance. Docker volume drivers (also referred to as plugins) are used to integrate the volumes with external storage systems, such as Amazon EBS. The built-in local volume driver or a third-party volume driver can be used. Docker volumes are only supported when running tasks on Amazon EC2 instances. Windows containers only support the use of the local driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Docker volumes \(p. 109\)](#).
- Bind mounts — A file or directory on the host, such as an Amazon EC2 instance or AWS Fargate, is mounted into a container. Bind mount host volumes are supported for tasks that are hosted on Fargate or Amazon EC2 instances. For more information, see [Bind mounts \(p. 114\)](#).

Topics

- [Fargate task storage \(p. 101\)](#)
- [Amazon EFS volumes \(p. 102\)](#)
- [FSx for Windows File Server volumes \(p. 105\)](#)
- [Docker volumes \(p. 109\)](#)
- [Bind mounts \(p. 114\)](#)

Fargate task storage

When provisioned, each Amazon ECS task that are hosted on AWS Fargate receives the following ephemeral storage for bind mounts. This can be mounted and shared among containers that use the `volumes`, `mountPoints`, and `volumesFrom` parameters in the task definition.

Fargate tasks using Windows platform version 1.0.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version 1.0.0 or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB. You can do this by specifying the `ephemeralStorage` parameter in your task definition.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage that your task has to use, you must subtract the amount of storage that your container image uses from the total amount of ephemeral storage your task is allocated.

For more information, see [Bind mounts \(p. 114\)](#).

Fargate tasks using Linux platform version 1.4.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version 1.4.0 or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB. You can do this by specifying the `ephemeralStorage` parameter in your task definition.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage that your task has to use, you must subtract the amount of storage that your container image uses from the total amount of ephemeral storage your task is allocated.

For tasks that use platform version 1.4.0 or later that are launched on May 28, 2020 or later, the ephemeral storage is encrypted with an AES-256 encryption algorithm. The algorithm uses an AWS owned encryption key.

For tasks that use platform version 1.4.0 or later that are launched on November 18, 2022 or later, the ephemeral storage usage is reported through the task metadata endpoint. Your applications in your tasks can query the task metadata endpoint version 4 to get their ephemeral storage reserved size and the amount used. Each task can only query the usage of that task.

Additionally, the ephemeral storage reserved size and the amount used are sent to Amazon CloudWatch Container Insights if you turn on Container Insights.

Note

Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as `df`.

Fargate tasks using Linux platform version 1.3.0 or earlier

For Amazon ECS on Fargate tasks using platform version 1.3.0 or earlier, each task receives the following ephemeral storage.

- 10 GB of Docker layer storage

Note

This amount includes both compressed and uncompressed container image artifacts.

- 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.

Amazon EFS volumes

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage they need and when they need it.

You can use Amazon EFS file systems with Amazon ECS to export file system data across your fleet of container instances. That way, your tasks have access to the same persistent storage, no matter the instance on which they land. Your task definitions must reference volume mounts on the container instance to use the file system. The following sections describe how to get started using Amazon EFS with Amazon ECS.

For a tutorial, see [Tutorial: Using Amazon EFS file systems with Amazon ECS using the console \(p. 718\)](#).

Amazon EFS volume considerations

Consider the following when using Amazon EFS volumes:

- For tasks that use the EC2 launch type, Amazon EFS file system support was added as a public preview with Amazon ECS-optimized AMI version 20191212 with container agent version 1.35.0. However, Amazon EFS file system support entered general availability with Amazon ECS-optimized AMI version 20200319 with container agent version 1.38.0, which contained the Amazon EFS access point and IAM authorization features. We recommend that you use Amazon ECS-optimized AMI version 20200319 or later to use these features. For more information, see [Amazon ECS-optimized AMI changelog \(p. 258\)](#).

Note

If you create your own AMI, you must use container agent 1.38.0 or later, `ecs-init` version 1.38.0-1 or later, and run the following commands on your Amazon EC2 instance to enable the Amazon ECS volume plugin. The commands are dependent on whether you're using Amazon Linux 2 or Amazon Linux as your base image.

Amazon Linux 2

```
yum install amazon-efs-utils
systemctl enable --now amazon-ecs-volume-plugin
```

Amazon Linux

```
yum install amazon-efs-utils
sudo shutdown -r now
```

- For tasks that are hosted on Fargate, Amazon EFS file systems are supported on platform version 1.4.0 or later (Linux). For more information, see [AWS Fargate platform versions \(p. 77\)](#).
- When using Amazon EFS volumes for tasks that are hosted on Fargate, Fargate creates a supervisor container that's responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the task's memory. The supervisor container is visible when querying the task metadata version 4 endpoint. Additionally, it is visible in CloudWatch Container Insights as the container name `aws-fargate-supervisor`. For more information, see [Task metadata endpoint version 4 \(p. 380\)](#).
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported on external instances.
- We recommend that you set the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` parameter in the agent configuration file to a value that is less than the default (about 1 hour). This change helps

prevent EFS mount credential expiration and allows for cleanup of mounts that are not in use. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

Using Amazon EFS access points

Amazon EFS access points are application-specific entry points into an EFS file system for managing application access to shared datasets. For more information about Amazon EFS access points and how to control access to them, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

Access points can enforce a user identity, including the user's POSIX groups, for all file system requests that are made through the access point. Access points can also enforce a different root directory for the file system. This is so that clients can only access data in the specified directory or its subdirectories.

Note

When creating an EFS access point, specify a path on the file system to serve as the root directory. When referencing the EFS file system with an access point ID in your Amazon ECS task definition, the root directory must either be omitted or set to /, which enforces the path set on the EFS access point.

You can use an Amazon ECS task IAM role to enforce that specific applications use a specific access point. By combining IAM policies with access points, you can provide secure access to specific datasets for your applications. For more information about how to use task IAM roles, see [Task IAM role \(p. 631\)](#).

Specifying an Amazon EFS file system in your task definition

To use Amazon EFS file system volumes for your containers, you must specify the volume and mount point configurations in your task definition. The following task definition JSON snippet shows the syntax for the volumes and mountPoints objects for a container.

```
{  
    "containerDefinitions": [  
        {  
            "name": "container-using-efs",  
            "image": "amazonlinux:2",  
            "entryPoint": [  
                "sh",  
                "-c"  
            ],  
            "command": [  
                "ls -la /mount/efs"  
            ],  
            "mountPoints": [  
                {  
                    "sourceVolume": "myEfsVolume",  
                    "containerPath": "/mount/efs",  
                    "readOnly": true  
                }  
            ]  
        }  
    ],  
    "volumes": [  
        {  
            "name": "myEfsVolume",  
            "efsVolumeConfiguration": {  
                "fileSystemId": "fs-1234",  
                "rootDirectory": "/path/to/my/data",  
                "transitEncryption": "ENABLED",  
                "transitEncryptionPort": integer,  
                "authorizationConfig": {  
                    "accessPointId": "fsap-1234",  
                }  
            }  
        }  
    ]  
}
```

```
        "iam": "ENABLED"
    }
}
]
```

efsVolumeConfiguration

Type: Object

Required: No

This parameter is specified when using Amazon EFS volumes.

fileSystemId

Type: String

Required: Yes

The Amazon EFS file system ID to use.

rootDirectory

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume is used. Specifying / has the same effect as omitting this parameter.

Important

If an EFS access point is specified in the authorizationConfig, the root directory parameter must either be omitted or set to /, which enforces the path set on the EFS access point.

transitEncryption

Type: String

Valid values: ENABLED | DISABLED

Required: No

Specifies whether to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. If Amazon EFS IAM authorization is used, transit encryption must be enabled. If this parameter is omitted, the default value of DISABLED is used. For more information, see [Encrypting Data in Transit](#) in the *Amazon Elastic File System User Guide*.

transitEncryptionPort

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, it uses the port selection strategy that the Amazon EFS mount helper uses. For more information, see [EFS Mount Helper](#) in the *Amazon Elastic File System User Guide*.

authorizationConfig

Type: Object

Required: No

The authorization configuration details for the Amazon EFS file system.
`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the `efsVolumeConfiguration` must either be omitted or set to `/`, which enforces the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: ENABLED | DISABLED

Required: No

Specifies whether to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of DISABLED is used. For more information, see [IAM Roles for Tasks](#).

FSx for Windows File Server volumes

FSx for Windows File Server provides fully managed Microsoft Windows file servers, that are backed by a Windows file system. When using FSx for Windows File Server together with ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#).

Note

EC2 instances that use the Amazon ECS-Optimized Windows Server 2016 Full AMI do not support FSx for Windows File Server ECS task volumes.

You cannot use FSx for Windows File Server volumes in a Windows containers on Fargate configuration.

You can use FSx for Windows File Server to deploy Windows workloads that require access to shared external storage, highly-available Regional storage, or high-throughput storage. You can mount one or more FSx for Windows File Server file system volumes to an ECS container that runs on an ECS Windows instance. You can share FSx for Windows File Server file system volumes between multiple ECS containers within a single ECS task.

To enable the use of FSx for Windows File Server with ECS, include the FSx for Windows File Server file system ID and the related information in a task definition. This is in the following example task definition JSON snippet. Before you create and run a task definition, you need the following.

- An ECS Windows EC2 instance that's joined to a valid domain. It can be hosted by an [AWS Directory Service for Microsoft Active Directory](#), On-premises Active Directory or self-hosted Active Directory on Amazon EC2.
- An AWS Secrets Manager secret or Systems Manager parameter that contains the credentials that are used to domain join the Active Directory and attach the FSx for Windows File Server file system. The credential values are the name and password credentials that you entered when creating the Active Directory.

The following sections describe how to get started using FSx for Windows File Server with Amazon ECS.

For a related tutorial, see [Tutorial: Using FSx for Windows File Server file systems with Amazon ECS \(p. 723\)](#).

FSx for Windows File Server volume considerations

Consider the following when using FSx for Windows File Server volumes:

- FSx for Windows File Server with Amazon ECS only supports Windows Amazon EC2 instances. Linux Amazon EC2 instances aren't supported.
- FSx for Windows File Server with Amazon ECS doesn't support AWS Fargate.
- FSx for Windows File Server with Amazon ECS with awsvpc network mode requires version 1.54.0 or later of the container agent.
- The maximum number of drive letters that can be used for an Amazon ECS task is 23. Each task with an FSx for Windows File Server volume gets a drive letter assigned to it.
- By default, task resource cleanup time is three hours after the task ended. Even if no tasks use it, a file mapping that's created by a task persists for three hours. The default cleanup time can be configured by using the Amazon ECS environment variable ECS_ENGIN_TASK_CLEANUP_WAIT_DURATION. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).
- Tasks typically only run in the same VPC as the FSx for Windows File Server file system. However, it's possible to have cross-VPC support if there's an established network connectivity between the Amazon ECS cluster VPC and the FSx for Windows File Server file-system through VPC peering.
- You control access to an FSx for Windows File Server file system at the network level by configuring the VPC security groups. Only tasks that are hosted on EC2 instances joined to the AD domain with correctly configured AD security groups can access the FSx for Windows File Server file-share. If the security groups are misconfigured, ECS fails to launch the task with the following error message: `unable to mount file system fs-id`.
- FSx for Windows File Server is integrated with AWS Identity and Access Management (IAM) to control the actions that your IAM users and groups can take on specific FSx for Windows File Server resources. With client authorization, customers can define IAM roles that allow or deny access to specific FSx for Windows File Server file systems, optionally require read-only access, and optionally allow or disallow root access to the file system from the client. For more information, see [Security](#) in the Amazon FSx Windows User Guide.

Specifying an FSx for Windows File Server file system in your task definition

To use FSx for Windows File Server file system volumes for your containers, specify the volume and mount point configurations in your task definition. The following task definition JSON snippet shows the syntax for the volumes and mountPoints objects for a container.

```
{  
    "containerDefinitions": [  
        {  
            "entryPoint": [  
                "powershell",  
                "-Command"  
            ],  
            "portMappings": [],  
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType file -Value '<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>It Works!</h2> <p>You are using Amazon FSx for Windows File Server file system for persistent container storage.</p>' -Force"],  
            "cpu": 512,  
            "memory": 256,  
            "mountPoints": [  
                {  
                    "sourceVolume": "AmazonFSxWindowsFileServer",  
                    "containerPath": "C:\\fsx-windows-dir",  
                    "readOnly": false  
                }  
            ]  
        }  
    ]  
}
```

```

    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
    "essential": false,
    "name": "container1",
    "mountPoints": [
        {
            "sourceVolume": "fsx-windows-dir",
            "containerPath": "C:\\\\fsx-windows-dir",
            "readOnly": false
        }
    ],
    {
        "entryPoint": [
            "powershell",
            "-Command"
        ],
        "portMappings": [
            {
                "hostPort": 443,
                "protocol": "tcp",
                "containerPort": 80
            }
        ],
        "command": ["Remove-Item -Recurse C:\\\\inetpub\\\\wwwroot\\\\* -Force; Start-Sleep -Seconds 120; Move-Item -Path C:\\\\fsx-windows-dir\\\\index.html -Destination C:\\\\inetpub\\\\wwwroot\\\\index.html -Force; C:\\\\ServiceMonitor.exe w3svc"],
        "mountPoints": [
            {
                "sourceVolume": "fsx-windows-dir",
                "containerPath": "C:\\\\fsx-windows-dir",
                "readOnly": false
            }
        ],
        "cpu": 512,
        "memory": 256,
        "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
        "essential": true,
        "name": "container2"
    }
],
"family": "fsx-windows",
"executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
"volumes": [
    {
        "name": "fsx-windows-dir",
        "fsxWindowsFileServerVolumeConfiguration": {
            "fileSystemId": "fs-0eeb5730b2EXAMPLE",
            "authorizationConfig": {
                "domain": "example.com",
                "credentialsParameter": "arn:arn-1234"
            },
            "rootDirectory": "share"
        }
    }
]
}

```

FSxWindowsFileServerVolumeConfiguration

Type: Object

Required: No

This parameter is specified when you're using [FSx for Windows File Server](#) file system for task storage.

`fileSystemId`

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

`rootDirectory`

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

`authorizationConfig`

`credentialsParameter`

Type: String

Required: Yes

The authorization credential options:

- Amazon Resource Name (ARN) of an [Secrets Manager](#) secret.
- Amazon Resource Name (ARN) of an [Systems Manager](#) parameter.

`domain`

Type: String

Required: Yes

A fully qualified domain name that's hosted by an [AWS Directory Service](#) Managed Microsoft AD (Active Directory) or self-hosted EC2 AD.

Credential storage methods

There are two different methods of storing credentials for use with the `credentialsParameter` parameter.

- **AWS Secrets Manager secret**

This credential can be created in the AWS Secrets Manager console by using the *Other type of secret* category. You add a row for each key/value pair, `username/admin` and `password/password`.

- **Systems Manager parameter**

This credential can be created in the Systems Manager parameter console by entering text in the form that's in the following example code snippet.

```
{  
  "username": "admin",  
  "password": "password"  
}
```

The `credentialsParameter` in the task definition `FSxWindowsFileServerVolumeConfiguration` parameter holds either the secret ARN or the Systems Manager parameter ARN. For more information, see [What is AWS Secrets Manager](#) in the *Secrets Manager User Guide* and [Systems Manager Parameter Store](#) from the *Systems Manager User Guide*.

Docker volumes

When using Docker volumes, the built-in local driver or a third-party volume driver can be used. Docker volumes are managed by Docker and a directory is created in `/var/lib/docker/volumes` on the container instance that contains the volume data.

To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Using Volumes](#).

Some common use cases for Docker volumes are the following:

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

Docker volume considerations

Consider the following when using Docker volumes:

- Docker volumes are only supported when using the EC2 launch type or external instances.
- Windows containers only support the use of the local driver.
- If a third-party driver is used, make sure it's installed and active on the container instance before the container agent is started. If the third-party driver isn't active before the agent is started, you can restart the container agent using one of the following commands:

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

Specifying a Docker volume in your task definition

Before your containers can use data volumes, you must specify the volume and mount point configurations in your task definition. This section describes the volume configuration for a container. For tasks that use a Docker volume, specify a `dockerVolumeConfiguration`. For tasks that use a bind mount host volume, specify a host and optional `sourcePath`.

The following task definition JSON shows the syntax for the `volumes` and `mountPoints` objects for a container.

```
{  
    "containerDefinitions": [  
        {  
            "mountPoints": [  
                {  
                    "sourceVolume": "string",  
                    "containerPath": "/path/to/mount_volume",  
                    "readOnly": boolean  
                }  
            ]  
        }  
    ]  
}
```

```
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autoprovision": boolean,
                "driver": "string",
                "driverOpts": {
                    "key": "value"
                },
                "labels": {
                    "key": "value"
                }
            }
        }
    ]
}
```

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.

dockerVolumeConfiguration

Type: Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when using the EC2 launch type. Windows containers only support the use of the local driver. To use bind mounts, specify a host instead.

scope

Type: String

Valid Values: task | shared

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a task are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as shared persist after the task stops.

autoprovision

Type: Boolean

Default value: false

Required: No

If this value is true, the Docker volume is created if it does not already exist. This field is only used if the scope is shared. If the scope is task then this parameter must either be omitted or set to false.

driver

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see [Docker plugin discovery](#). This parameter maps to `Driver` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--driver` option to [docker volume create](#).

driverOpts

Type: String

Required: No

A map of Docker driver specific options to pass through. This parameter maps to `DriverOpts` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--opt` option to [docker volume create](#).

labels

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to `Labels` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--label` option to [docker volume create](#).

mountPoints

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to `Volumes` in the [Create a container](#) section of the [Docker Remote API](#) and the `--volume` option to [docker run](#).

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`.

Examples

The following are examples that show the use of Docker volumes.

To provide nonpersistent storage for a container using a Docker volume

In this example, a container uses an empty data volume that is disposed of after the task is finished. One example use case is that you might have a container that needs to access some scratch file storage location during a task. This task can be achieved using a Docker volume.

1. In the task definition `volumes` section, define a data volume with name and `DockerVolumeConfiguration` values. In this example, we specify the scope as `task` so the volume is deleted after the task stops and use the built-in `local` driver.

```
"volumes": [  
    {  
        "name": "scratch",  
        "dockerVolumeConfiguration" : {  
            "scope": "task",  
            "driver": "local",  
            "labels": {  
                "scratch": "space"  
            }  
        }  
    }  
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume at on the container.

```
"containerDefinitions": [  
    {  
        "name": "container-1",  
        "mountPoints": [  
            {  
                "sourceVolume": "scratch",  
                "containerPath": "/var/scratch"  
            }  
        ]  
    }  
]
```

To provide persistent storage for a container using a Docker volume

In this example, you want a shared volume for multiple containers to use and you want it to persist after any single task that use it stopped. The built-in `local` driver is being used. This is so the volume is still tied to the lifecycle of the container instance.

1. In the task definition `volumes` section, define a data volume with name and `DockerVolumeConfiguration` values. In this example, specify a `shared` scope so the volume persists, set `autoProvision` to `true`. This is so that the volume is created for use. Then, also use the built-in `local` driver.

```
"volumes": [
    {
        "name": "database",
        "dockerVolumeConfiguration" : {
            "scope": "shared",
            "autoprovision": true,
            "driver": "local",
            "labels": {
                "database": "database_name"
            }
        }
    }
]
```

2. In the containerDefinitions section, define a container with mountPoints values that reference the name of the defined volume and the containerPath value to mount the volume at on the container.

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
            {
                "sourceVolume": "database",
                "containerPath": "/var/database"
            }
        ]
    },
    {
        "name": "container-2",
        "mountPoints": [
            {
                "sourceVolume": "database",
                "containerPath": "/var/database"
            }
        ]
    }
]
```

To provide NFS persistent storage for a container using a Docker volume

In this example, a container uses an NFS data volume that is automatically mounted when the task starts and unmounted when the task stops. This uses the Docker built-in local driver. One example use case is that you might have a local NFS storage and need to access it from an ECS Anywhere task. This can be achieved using a Docker volume with NFS driver option.

1. In the task definition volumes section, define a data volume with name and DockerVolumeConfiguration values. In this example, specify a task scope so the volume is unmounted after the task stops. Use the local driver and configure the driverOpts with the type, device, and o options accordingly. Replace NFS_SERVER with the NFS server endpoint.

```
"volumes": [
    {
        "name": "NFS",
        "dockerVolumeConfiguration" : {
            "scope": "task",
            "driver": "local",
            "driverOpts": {
                "type": "nfs",
                "device": "$NFS_SERVER:/mnt/nfs",
                "o": "rw,nosuid,nodev"
            }
        }
    }
]
```

```
        "o": "addr=$NFS_SERVER"
    }
}
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume on the container.

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
            {
                "sourceVolume": "NFS",
                "containerPath": "/var/nfsmount"
            }
        ]
    }
]
```

Bind mounts

With bind mounts, a file or directory on a host, such as an Amazon EC2 instance or AWS Fargate, is mounted into a container. Bind mounts are supported for tasks that are hosted on both Fargate and Amazon EC2 instances. By default, bind mounts are tied to the lifecycle of the container that uses them. After all of the containers that use a bind mount are stopped, such as when a task is stopped, the data is removed. For tasks that are hosted on Amazon EC2 instances, the data can be tied to the lifecycle of the host Amazon EC2 instance by specifying a `hostPath` and optional `sourcePath` value in your task definition. For more information, see [Using bind mounts](#) in the Docker documentation.

The following are common use cases for bind mounts.

- To provide an empty data volume to mount in one or more containers.
- To mount a host data volume in one or more containers.
- To share a data volume from a source container with other containers in the same task.
- To expose a path and its contents from a Dockerfile to one or more containers.

Considerations when using bind mounts

When using bind mounts, consider the following.

- For tasks that are hosted on AWS Fargate using platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows), by default they receive a minimum of 20 GiB of ephemeral storage for bind mounts. For Linux tasks, the total amount of ephemeral storage can be increased to a maximum of 200 GiB by specifying the `ephemeralStorage` parameter in your task definition. By default, Amazon ECS Windows tasks that are hosted on Fargate using platform version `1.0.0` or later receive a minimum of 20 GiB of ephemeral storage. You can increase the total amount of ephemeral storage, up to a maximum of 200 GiB by specifying the `ephemeralStorage` parameter in your task definition.
- To expose files from a Dockerfile to a data volume when a task is run, the Amazon ECS data plane looks for a `VOLUME` directive. If the absolute path that's specified in the `VOLUME` directive is the same as the `containerPath` that's specified in the task definition, the data in the `VOLUME` directive path is copied to the data volume. In the following Dockerfile example, a file that's named `examplefile` in the `/var/log/exported` directory is written to the host and then mounted inside the container.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

By default, the volume permissions are set to 0755 and the owner as root. You can customize these permissions in the Dockerfile. The following example defines the owner of the directory as node.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

- For tasks that are hosted on Amazon EC2 instances, when a host and sourcePath value aren't specified, the Docker daemon manages the bind mount for you. When no containers reference this bind mount, the Amazon ECS container agent task cleanup service eventually deletes it. By default, this happens three hours after the container exits. However, you can configure this duration with the [ECS_ENGIN_TASK_CLEANUP_WAIT_DURATION](#) agent variable. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#). If you need this data to persist beyond the lifecycle of the container, specify a sourcePath value for the bind mount.

Specifying a bind mount in your task definition

For Amazon ECS tasks that are hosted on either Fargate or Amazon EC2 instances, the following task definition JSON snippet shows the syntax for the volumes, mountPoints, and ephemeralStorage objects for a task definition.

```
{
    "family": "",
    ...
    "containerDefinitions" : [
        {
            "mountPoints" : [
                {
                    "containerPath" : "/path/to/mount_volume",
                    "sourceVolume" : "string"
                }
            ],
            "name" : "string"
        }
    ],
    ...
    "volumes" : [
        {
            "name" : "string"
        }
    ],
    "ephemeralStorage": {
        "sizeInGiB": integer
    }
}
```

For Amazon ECS tasks that are hosted on Amazon EC2 instances, you can use the optional host parameter and a sourcePath when specifying the task volume details. When it's specified, it ties the bind mount to the lifecycle of the task rather than the container.

```
"volumes" : [
  {
    "host" : {
      "sourcePath" : "string"
    },
    "name" : "string"
  }
]
```

The following describes each task definition parameter in more detail.

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.

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. This behavior is not supported for Windows containers. For information about the behavior on Windows containers, see [docker bind mount on Windows does not create a new directory on host, if same was missing #44799](#) on the moby Github website.

mountPoints

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to Volumes in the [Create a container](#) section of the [Docker Remote API](#) and the --volume option to [docker run](#).

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.

`ephemeralStorage`

Type: Object

Required: No

The amount of ephemeral storage to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks hosted on AWS Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).

You can use the Copilot CLI, CloudFormation, the AWS SDK or the CLI to specify ephemeral storage for a bind mount.

Bind mount examples

The following examples cover the most common use cases for using a bind mount for your containers.

To allocate an increased amount of ephemeral storage space for a Fargate task

For Amazon ECS tasks that are hosted on Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 (Windows), you can allocate more than the default amount of ephemeral storage for the containers in your task to use. This example can be incorporated into the other examples to allocate more ephemeral storage for your Fargate tasks.

- In the task definition, define an `ephemeralStorage` object. The `sizeInGiB` must be an integer between the values of 21 and 200 and is expressed in GiB.

```
"ephemeralStorage": {  
    "sizeInGiB": integer
```

```
}
```

To provide an empty data volume for one or more containers

In some cases, you want to provide the containers in a task some scratch space. For example, you might have two database containers that need to access the same scratch file storage location during a task. This can be achieved using a bind mount.

1. In the task definition volumes section, define a bind mount with the name `database_scratch`.

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

2. In the containerDefinitions section, create the database container definitions. This is so that they mount the volume.

```
"containerDefinitions": [  
  {  
    "name": "database1",  
    "image": "my-repo/database",  
    "cpu": 100,  
    "memory": 100,  
    "essential": true,  
    "mountPoints": [  
      {  
        "sourceVolume": "database_scratch",  
        "containerPath": "/var/scratch"  
      }  
    ]  
  },  
  {  
    "name": "database2",  
    "image": "my-repo/database",  
    "cpu": 100,  
    "memory": 100,  
    "essential": true,  
    "mountPoints": [  
      {  
        "sourceVolume": "database_scratch",  
        "containerPath": "/var/scratch"  
      }  
    ]  
  }  
]
```

To expose a path and its contents in a Dockerfile to a container

In this example, you have a Dockerfile that writes data that you want to mount inside a container. This example works for tasks that are hosted on Fargate or Amazon EC2 instances.

1. Create a Dockerfile. The following example uses the public Amazon Linux 2 container image and creates a file that's named `examplefile` in the `/var/log/exported` directory that we want to mount inside the container. The `VOLUME` directive should specify an absolute path.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest  
RUN mkdir -p /var/log/exported
```

```
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

By default, the volume permissions are set to 0755 and the owner as root. These permissions can be changed in the Dockerfile. In the following example, the owner of the /var/log/exported directory is set to node.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

2. In the task definition volumes section, define a volume with the name application_logs.

```
"volumes": [
  {
    "name": "application_logs",
  }
]
```

3. In the containerDefinitions section, create the application container definitions. This is so they mount the storage. The containerPath value must match the absolute path that's specified in the VOLUME directive from the Dockerfile.

```
"containerDefinitions": [
  {
    "name": "application1",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  },
  {
    "name": "application2",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  }
]
```

To provide an empty data volume for a container that's tied to the lifecycle of the host Amazon EC2 instance

For tasks that are hosted on Amazon EC2 instances, you can use bind mounts and have the data tied to the lifecycle of the host Amazon EC2 instance. You can do this by using the `host` parameter and specifying a `sourcePath` value. Any files that exist at the `sourcePath` are presented to the containers at the `containerPath` value. Any files that are written to the `containerPath` value are written to the `sourcePath` value on the host Amazon EC2 instance.

Important

Amazon ECS doesn't sync your storage across Amazon EC2 instances. Tasks that use persistent storage can be placed on any Amazon EC2 instance in your cluster that has available capacity. If your tasks require persistent storage after stopping and restarting, always specify the same Amazon EC2 instance at task launch time with the AWS CLI [start-task](#) command. You can also use Amazon EFS volumes for persistent storage. For more information, see [Amazon EFS volumes \(p. 102\)](#).

1. In the task definition `volumes` section, define a bind mount with name and `sourcePath` values. In the following example, the host Amazon EC2 instance contains data at `/ecs/webdata` that you want to mount inside the container.

```
"volumes": [  
  {  
    "name": "webdata",  
    "host": {  
      "sourcePath": "/ecs/webdata"  
    }  
  }  
]
```

2. In the `containerDefinitions` section, define a container with a `mountPoints` value that references the name of the bind mount and the `containerPath` value to mount the bind mount at on the container.

```
"containerDefinitions": [  
  {  
    "name": "web",  
    "image": "nginx",  
    "cpu": 99,  
    "memory": 100,  
    "portMappings": [  
      {  
        "containerPort": 80,  
        "hostPort": 80  
      }  
    ],  
    "essential": true,  
    "mountPoints": [  
      {  
        "sourceVolume": "webdata",  
        "containerPath": "/usr/share/nginx/html"  
      }  
    ]  
  }  
]
```

To mount a defined volume on multiple containers at different locations

You can define a data volume in a task definition and mount that volume at different locations on different containers. For example, your host container has a website data folder at `/data/webroot`. You

might want to mount that data volume as read-only on two different web servers that have different document roots.

1. In the task definition volumes section, define a data volume with the name `webroot` and the source path `/data/webroot`.

```
"volumes": [
  {
    "name": "webroot",
    "host": {
      "sourcePath": "/data/webroot"
    }
  }
]
```

2. In the containerDefinitions section, define a container for each web server with `mountPoints` values that associate the `webroot` volume with the `containerPath` value pointing to the document root for that container.

```
"containerDefinitions": [
  {
    "name": "web-server-1",
    "image": "my-repo/ubuntu-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/var/www/html",
        "readOnly": true
      }
    ]
  },
  {
    "name": "web-server-2",
    "image": "my-repo/sles11-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 8080,
        "hostPort": 8080
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/srv/www/htdocs",
        "readOnly": true
      }
    ]
  }
]
```

To mount volumes from another container using `volumesFrom`

For tasks hosted on Amazon EC2 instances, you can define one or more volumes on a container, and then use the `volumesFrom` parameter in a different container definition within the same task to mount all of the volumes from the `sourceContainer` at their originally defined mount points. The `volumesFrom` parameter applies to volumes defined in the task definition, and those that are built into the image with a Dockerfile.

1. (Optional) To share a volume that is built into an image, use the `VOLUME` instruction in the Dockerfile. The following example Dockerfile uses an `httpd` image, and then adds a volume and mounts it at `dockerfile_volume` in the Apache document root. It is the folder used by the `httpd` web server.

```
FROM httpd
VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]
```

You can build an image with this Dockerfile and push it to a repository, such as Docker Hub, and use it in your task definition. The example `my-repo/httpd_dockerfile_volume` image that's used in the following steps was built with the above Dockerfile.

2. Create a task definition that defines your other volumes and mount points for the containers. In this example `volumes` section, you create an empty volume called `empty`, which the Docker daemon manages. There's also a host volume defined that's called `host_etc`. It exports the `/etc` folder on the host container instance.

```
{
  "family": "test-volumes-from",
  "volumes": [
    {
      "name": "empty",
      "host": {}
    },
    {
      "name": "host_etc",
      "host": {
        "sourcePath": "/etc"
      }
    }
  ],
}
```

In the container definitions section, create a container that mounts the volumes defined earlier. In this example, the web container mounts the `empty` and `host_etc` volumes. This is the container that uses the image built with a volume in the Dockerfile.

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "my-repo/httpd_dockerfile_volume",
    "cpu": 100,
    "memory": 500,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "mountPoints": [
      {
        "sourceVolume": "empty",
        "containerPath": "/usr/local/apache2/htdocs/empty_volume"
      }
    ]
  }
]
```

```
        },
        {
            "sourceVolume": "host_etc",
            "containerPath": "/usr/local/apache2/htdocs/host_etc"
        }
    ],
    "essential": true
},
```

Create another container that uses `volumesFrom` to mount all of the volumes that are associated with the web container. All of the volumes on the web container are likewise mounted on the busybox container. This includes the volume that's specified in the Dockerfile that was used to build the `my-repo/httpd_dockerfile_volume` image.

```
{
    "name": "busybox",
    "image": "busybox",
    "volumesFrom": [
        {
            "sourceContainer": "web"
        }
    ],
    "cpu": 100,
    "memory": 500,
    "entryPoint": [
        "sh",
        "-c"
    ],
    "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date && echo $(date) > /usr/local/apache2/htdocs/host_etc/date && echo $(date) > /usr/local/apache2/htdocs/dockerfile_volume/date"
    ],
    "essential": false
}
```

When this task is run, the two containers mount the volumes, and the command in the busybox container writes the date and time to a file. This file is called `date` in each of the volume folders. The folders are then visible at the website displayed by the web container.

Note

Because the busybox container runs a quick command and then exits, it must be set as `"essential": false` in the container definition. Otherwise, it stops the entire task when it exits.

Managing container swap space

With Amazon ECS, you can control the usage of swap memory space on your Linux-based Amazon EC2 instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled. For those that have it enabled, the maximum amount of swap space that's used can be limited. For example, latency-critical containers can have swap disabled. In contrast, containers with high transient memory demands can have swap turned on to reduce the chances of out-of-memory errors when the container is under load.

The swap configuration for a container is managed by the following container definition parameters.

maxSwap

The total amount of swap memory (in MiB) a container can use. This parameter is translated to the `--memory-swap` option to [docker run](#) where the value is the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container doesn't use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container uses the swap configuration for the container instance that it's running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

swappiness

You can use this to tune a container's memory swappiness behavior. A `swappiness` value of `0` causes swapping to not occur unless required. A `swappiness` value of `100` causes pages to be swapped aggressively. Accepted values are whole numbers between `0` and `100`. If the `swappiness` parameter isn't specified, a default value of `60` is used. If a value isn't specified for `maxSwap`, this parameter is ignored. This parameter maps to the `--memory-swappiness` option to [docker run](#).

In the following example, the JSON syntax is provided.

```
"containerDefinitions": [{}  
    ...  
    "linuxParameters": {  
        "maxSwap": integer,  
        "swappiness": integer  
    },  
    ...  
]
```

Container swap considerations

Consider the following when you use a per-container swap configuration.

- Swap space must be enabled and allocated on the Amazon EC2 instance hosting your tasks for the containers to use. By default, the Amazon ECS optimized AMIs do not have swap enabled. You must enable swap on the instance to use this feature. For more information, see [Instance Store Swap Volumes in the Amazon EC2 User Guide for Linux Instances](#) or [How do I allocate memory to work as swap space in an Amazon EC2 instance by using a swap file?](#)
- The swap space container definition parameters are only supported for task definitions that specify the EC2 launch type. These parameters are not supported for task definitions intended only for Amazon ECS on Fargate use.
- This feature is only supported for Linux containers. Windows containers are not supported currently.
- If the `maxSwap` and `swappiness` container definition parameters are omitted from a task definition, each container has a default `swappiness` value of `60`. Moreover, the total swap usage is limited to two times the memory reservation of the container.
- If you're using tasks on Amazon Linux 2023 the `swappiness` parameter isn't supported.

Amazon EC2 Windows task definition considerations

The following parameters aren't supported for Amazon EC2 Windows task definitions:

- `containerDefinitions`
- `disableNetworking`
- `dnsServers`

- dnsSearchDomains
- extraHosts
- links
- linuxParameters
- privileged
- readonlyRootFilesystem
- user
- ulimits
- volumes
 - dockerVolumeConfiguration
- cpu
 - We recommend specifying container-level CPU for Windows containers.
- memory
 - We recommend specifying container-level memory for Windows containers.
- proxyConfiguration
- ipcMode
- pidMode

Additional configuration for Windows IAM roles for tasks

The IAM roles for tasks with Windows features requires additional configuration, but much of this configuration is similar to configuring IAM roles for tasks on Linux container instances. For more information see [the section called "Additional configuration for Windows task role" \(p. 637\)](#).

Creating a task definition using the console

Create your task definitions using the Amazon ECS console. To make the task definition creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

You can create a task definition by stepping through the console, or by editing a JSON file.

JSON validation

The Amazon ECS console JSON editor validates the following in the JSON file:

- The file is a valid JSON file
- The file does not contain any extraneous keys
- The file contains the familyName parameter
- There is at least one entry under containerDefinitions

AWS CloudFormation stacks

The following behavior applies to task definitions created in the new console before January 12, 2023.

When you create a task definition, the Amazon ECS console automatically creates a CloudFormation stack that has a name that begins with "ECS-Console-V2-TaskDefinition-". If you used the AWS CLI or

SDK to deregister the task definition, then you must manually delete the task definition stack. For more information, see [Deleting a Stack](#) in the *AWS CloudFormation User Guide*.

Task definitions created after January 12, 2023 will not have a CloudFormation stack automatically created.

Amazon ECS console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition**, **Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. For **Launch type**, choose the application environment. The console default is **AWS Fargate (serverless)**. Amazon ECS performs validation using this value to ensure the task definition parameters are valid for the infrastructure type.
6. For **Operating system/Architecture**, choose the operating system and CPU architecture for the task.

To run your task on a 64-bit ARM architecture, select **Linux/ARM64**. For more information, see [the section called "Runtime platform" \(p. 801\)](#).

To run your **AWS Fargate** tasks on Windows containers, choose a supported Windows operating system. For more information, see [the section called "Task Operating Systems" \(p. 68\)](#).

7. For **Task size**, choose the CPU and memory values to reserve for the task. The CPU value is specified as vCPUs and memory is specified as GB.

For tasks hosted on Fargate, the following table shows the valid CPU and memory combinations.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

For tasks hosted on Amazon EC2, supported task CPU values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

Note

Task-level CPU and memory parameters are ignored for Windows containers.

8. For **Network mode**, choose the network mode to use. The default is **awsvpc** mode. For more information, see [Amazon ECS task networking](#).

If you choose **bridge**, under **Port mappings**, for **Host port**, enter the port number on the container instance to reserve for your container.

9. (Optional) Expand the **Task roles** section to specify the following:

- a. For **Task role**, choose the IAM role to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS APIs.
- b. For **Task execution role**, choose the role.

For information about when to use a task execution role, see [the section called “Task execution IAM role” \(p. 626\)](#). If you do not need the role, choose **None**.

10. For each container to define in your task definition, complete the following steps.

- a. For **Name**, enter a name for the container.
- b. For **Image URI**, enter the image to use to start a container. Images in the Amazon ECR Public Gallery registry may be specified using the Amazon ECR Public registry name only. For example, if `public.ecr.aws/ecs/amazon-ecs-agent:latest` is specified, the Amazon Linux container hosted on Amazon ECR Public Gallery is used. For all other repositories, specify the repository using either the `repository-url/image:tag` or `repository-url/image@digest` formats.
- c. If your image is in a private registry outside of Amazon ECR, under **Private registry**, turn on **Private registry authentication**. Then, in **Secrets Manager ARN or name**, enter the Amazon Resource Name (ARN) of the secret.
- d. For **Essential container**, if your task definition has two or more containers defined, you may specify whether the container should be considered essential. If a container is marked as **Essential**, if that container stops then the task is stopped. Each task definition must contain at least one essential container.
- e. A port mapping allows the container to access ports on the host to send or receive traffic. Under **Port mappings**, do one of the following:
 - When you use the **awsvpc** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.
 - When you use the **bridge** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.

Choose **Add more port mappings** to specify additional container port mappings.

- f. To give the container read-only access to its root file system, for **Read only root file system**, select **Read only**.
- g. (Optional) To define the container-level CPU, GPU, and memory limits that are different from task-level values under **Resource allocation limits**, do the following:
 - For **CPU**, enter the number of CPU units the Amazon ECS container agent reserves for the container.
 - For **GPU**, enter the number of GPU units for the container instance.

An Amazon EC2 instance with GPU support has 1 GPU unit for every GPU. For more information, see [the section called “Working with GPUs on Amazon ECS” \(p. 142\)](#).

- For **Memory hard limit**, enter the amount of memory, in GB to present to the container. If the container attempts to exceed the hard limit, the container stops.
- The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container, so you should not specify fewer than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier 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.

- For **Memory soft limit**, enter the soft limit (in GB) of memory to reserve for the container.

When system memory is under contention, Docker attempts to keep the container memory to this soft limit. If you don't specify task-level memory, you must specify a non-zero integer for one or both of **Memory hard limit** and **Memory soft limit**. If you specify both, **Memory hard limit** must be greater than **Memory soft limit**.

This is not supported on Windows containers.

- h. (Optional) Expand the **Environment variables** section to specify environment variables to inject into the container. You can specify environment variables either individually using key-value pairs or in bulk by specifying an environment variable file hosted in an Amazon S3 bucket. For information on how to format an environment variable file, see [Passing environment variables to a container \(p. 197\)](#).
- i. (Optional) Select the **Use log collection** option to specify a log configuration. For each available log driver, there are log driver options to specify. The default option sends container logs to CloudWatch Logs. The other log driver options are configured using AWS FireLens. For more information, see [Custom log routing \(p. 166\)](#).

The following describes each container log destination in more detail.

- **Amazon CloudWatch** — Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided which creates a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.
 - **Export logs to Splunk** — Configure the task to send container logs to the splunk driver that sends the logs to a remote service. You need to enter the URL to your Splunk web web service and the Splunk token is specified as a secret option because it can be treated as sensitive data.
 - **Export logs to Amazon Kinesis Data Firehose** — Configure the task to send container logs to Kinesis Data Firehose. The default log driver options are provided which sends logs to an Kinesis Data Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
 - **Export logs to Amazon Kinesis Data Streams** — Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided which sends logs to an Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
 - **Export logs to Amazon OpenSearch Service** — Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided. For more information, see [Forwarding logs to an Amazon OpenSearch Service domain \(p. 193\)](#).
 - **Export logs to Amazon S3** — Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided but you must specify a valid Amazon S3 bucket name.
- j. (Optional) Configure additional container parameters.

To configure this option	Do this	
Healthcheck These are the commands that determine if a container is healthy	Expand HealthCheck , and then configure the following items: <ul style="list-style-type: none"> For Command, enter a comma-separated list of commands. You can start the commands with CMD to run the command arguments directly, or CMD-SHELL to run the command with the container's default shell. If neither is specified, CMD is used. For Interval, enter the number of seconds between each health check. The valid values are between 5 and 30. For Timeout, enter the period of time (in seconds) to wait for a health check to succeed before it's considered a failure. The valid values are between 2 and 60. For Start period, enter the period of time (in seconds) to wait for a container to bootstrap before the health check commands run. The valid values are between 0 and 300. For Retries, enter the number of times to retry the health check commands when there is a failure. The value values are between 1 and 10. 	

To configure this option	Do this	
<p>Docker configuration</p> <p>These override the values in the Dockerfile.</p>	<p>Expand Docker configuration, and then configure the following items:</p> <ul style="list-style-type: none"> For Command, enter an executable command for a container. <p>This parameter maps to Cmd in the Create a container section of the Docker Remote API and the COMMAND option to docker run. This will override the CMD instruction in a Dockerfile.</p> <ul style="list-style-type: none"> For Entry point, enter the Docker ENTRYPPOINT that is passed to the container. <p>This parameter maps to Entrypoint in the Create a container section of the Docker Remote API and the --entrypoint option to docker run. This will override the ENTRYPPOINT instruction in a Dockerfile.</p> <ul style="list-style-type: none"> For Working directory, enter the directory that the container will run any entry point and command instructions provided. <p>This parameter maps to WorkingDir in the Create a container section of the Docker Remote API and the --workdir option to docker run. This will override the WORKDIR instruction in a Dockerfile.</p>	

To configure this option	Do this	
Ulimits These values overwrite the default resource quota setting for the operating system. This parameter maps to Ulimits in the Create a container section of the Docker Remote API and the --ulimit option to docker run .	Expand Resource limits (ulimits) , and then choose Add ulimit . For Limit name choose the limit. Then, for Soft limit and Hard limit , enter the values. To add additional ulimits, choose Add ulimit .	
Docker labels This option adds metadata to your container. This parameter maps to Labels in the Create a container section of the Docker Remote API and the --label option to docker run .	Expand Docker labels , choose Add key value pair , and then enter the Key and Value . To add additional ulimits, choose Add key value pair .	
Container startup order This option defines dependencies for container startup and shutdown. A container can contain multiple dependencies.	Expand Startup dependency ordering , and then configure the following: a. Choose Add container dependency . b. For Container , choose the container. c. For Condition , choose the startup dependency condition. To add an additional dependency, choose Add container dependency .	

To configure this option	Do this	
Container timeouts These options determine when to start and stop a container.	Expand Container timeouts , and then configure the following: <ul style="list-style-type: none"> To configure the time wait before giving up on resolving dependencies for a container, for Start time, enter the number of seconds. To configure the time to wait before the container is stopped if it doesn't exit normally on its own, for Stop time, enter the number of seconds. 	

- k. (Optional) Choose **Add more containers** to add additional containers to the task definition. Choose **Next** after you define all your containers.
11. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate as well as add a data volume configuration for the task.
- To expand the available ephemeral storage beyond the default value of 20 GiB for your Fargate tasks, for **Amount**, enter a value up to 200 GiB.
12. (Optional) To add a data volume configuration for the task definition, choose **Add volume**, and then configure the volume type.

Volume type	Steps	
Bind mount	a. For Volume type , choose Bind mount . b. For Volume name , enter a name for the data volume. The data volume name is used when creating a container mount point. c. Choose Add mount point , and then configure the following: <ul style="list-style-type: none"> For Container, choose the container for the mount point. For Source volume, choose the data volume to mount to the container. For Container path, enter the path on the container to mount the volume. For Read only, select whether the container 	

Volume type	Steps	
	<p>has read-only access to the volume.</p> <p>d. To add additional mount points, Add mount point.</p>	

Volume type	Steps
EFS	<p>a. For Volume type, choose EFS.</p> <p>b. For Volume name, enter a name for the data volume.</p> <p>c. For File system ID, choose the Amazon EFS file system ID.</p> <p>d. (Optional) For Root directory, enter the directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume is used.</p> <p>If you plan to use an EFS access point, leave this field blank.</p> <p>e. (Optional) For Access point, choose the access point ID to use.</p> <p>f. (Optional) To encrypt the data between the Amazon EFS file system and the Amazon ECS host or to use the task execution role when mounting the volume, choose Advanced configurations, and then configure the following:</p> <ul style="list-style-type: none"> To encrypt the data between the Amazon EFS file system and the Amazon ECS host, select Transit encryption, and then for Port, enter the port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, it uses the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the <i>Amazon Elastic File System User Guide</i>.

Volume type	Steps
	<ul style="list-style-type: none"> • To use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system, select IAM authorization. <p>g. Choose Add mount point, and then configure the following:</p> <ul style="list-style-type: none"> • For Container, choose the container for the mount point. • For Source volume, choose the data volume to mount to the container. • For Container path, enter the path on the container to mount the volume. • For Read only, select whether the container has read-only access to the volume. <p>h. To add additional mount points, Add mount point.</p>

Volume type	Steps
Docker	<p>a. For Volume type, choose Docker volume.</p> <p>b. For Volume name, enter a name for the data volume. The data volume name is used when creating a container mount point.</p> <p>c. For Driver, enter the Docker volume configuration. Windows containers only support the use of the local driver. To use bind mounts, specify a host.</p> <p>d. For Scope, choose the volume lifecycle.</p> <ul style="list-style-type: none"> • To have the lifecycle last when the task starts and stops, choose Task. • To have the volume persist after the task stops, choose Shared. <p>e. Choose Add mount point, and then configure the following:</p> <ul style="list-style-type: none"> • For Container, choose the container for the mount point. • For Source volume, choose the data volume to mount to the container. • For Container path, enter the path on the container to mount the volume. • For Read only, select whether the container has read-only access to the volume. <p>f. To add additional mount points, Add mount point.</p>

Volume type	Steps
FSx for Windows File Server	<p>a. For Volume type, choose FSx for Windows File Server.</p> <p>b. For File system ID, choose the FSx for Windows File Server file system ID.</p> <p>c. For Root directory, enter the directory, enter the directory within the FSx for Windows File Server file system to mount as the root directory inside the host..</p> <p>d. For Credential parameter, choose how the credentials are stored.</p> <ul style="list-style-type: none"> • To use Secrets Manager, enter the Amazon Resource Name (ARN) of a Secrets Manager secret. • To use Systems Manager, enter Amazon Resource Name (ARN) of a Systems Manager parameter. <p>e. For Domain, enter the fully qualified domain name that's hosted by an AWS Directory Service Managed Microsoft AD (Active Directory) or self-hosted EC2 AD.</p> <p>f. Choose Add mount point, and then configure the following:</p> <ul style="list-style-type: none"> • For Container, choose the container for the mount point. • For Source volume, choose the data volume to mount to the container. • For Container path, enter the path on the container to mount the volume. • For Read only, select whether the container has read-only access to the volume.

Volume type	Steps
	<p>g. To add additional mount points, Add mount point.</p>

13. To add a volume from another container, choose **Add volume from**, and then configure the following:
 - For **Container**, choose the container.
 - For **Source**, choose the container which has the volume you want to mount.
 - For **Read only**, select whether the container has read-only access to the volume.
14. (Optional) To configure your application trace and metric collection settings using the AWS Distro for OpenTelemetry integration, expand **Monitoring**, and then select the **Use metric collection** to collect and send metrics for your tasks to either Amazon CloudWatch or Amazon Managed Service for Prometheus. When this option is selected, Amazon ECS creates an AWS Distro for OpenTelemetry container sidecar which is preconfigured to send the application metrics. For more information, see [Collecting application metrics \(p. 579\)](#).
 - a. When **Amazon CloudWatch** is selected, your custom application metrics are routed to CloudWatch as custom metrics. For more information, see [Exporting application metrics to Amazon CloudWatch \(p. 579\)](#).

Important
When exporting application metrics to Amazon CloudWatch, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch \(p. 579\)](#).
 - b. When you select **Amazon Managed Service for Prometheus (Prometheus libraries instrumentation)**, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, enter the remote write endpoint URL for your Prometheus workspace. For **Scraping target**, enter the host and port the AWS Distro for OpenTelemetry collector can use to scrape for metrics data. For more information, see [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 582\)](#).

Important
When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus \(p. 582\)](#).
 - c. When you select **Amazon Managed Service for Prometheus (OpenTelemetry instrumentation)**, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, enter the remote write endpoint URL for your Prometheus workspace. For more information, see [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 582\)](#).

Important
When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus \(p. 582\)](#).
15. (Optional) Expand the **Tags** section to add tags, as key-value pairs, to the task definition.
 - [Add a tag] Choose **Add tag**, and then do the following:

- For **Key**, enter the key name.
 - For **Value**, enter the key value.
 - [Remove a tag] Next to the tag, choose **Remove tag**.
16. Choose **Create** to register the task definition.

Amazon ECS console JSON editor

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, edit your JSON file,

The JSON must pass the validation checks specified in [the section called “JSON validation” \(p. 125\)](#).
5. Choose **Create**.

Updating a task definition using the console

A *task definition revision* is a copy of the current task definition with the new parameter values replacing the existing ones. All parameters that you do not modify are in the new revision.

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.

When you create a revision, you can modify the following container properties and environment properties.

- Container image URI
- Port mappings
- Environment variables
- Task size
- Container size
- Task role
- Task execution role
- Volumes and container mount points
- Private registry

JSON validation

The Amazon ECS console JSON editor validates the following in the JSON file:

- The file is a valid JSON file
- The file does not contain any extraneous keys
- The file contains the `familyName` parameter
- There is at least one entry under `containerDefinitions`

Amazon ECS console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.

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, and then choose **Create new revision, Create new revision**.
5. On the **Create new task definition revision** page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, and then make the changes.
6. Verify the information, and then choose **Update**.
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 using the console \(p. 465\)](#).

Amazon ECS console JSON editor

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new revision, Create new revision with JSON**.
4. In the JSON editor box, edit your JSON file,

The JSON must pass the validation checks specified in [the section called "JSON validation" \(p. 139\)](#).
5. Choose **Create**.

Deregistering a task definition revision using the console

If you decide that you no longer need a specific task definition revision in Amazon ECS, you can deregister the task definition revision 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 revision, it is immediately marked as INACTIVE. Existing tasks and services that reference an INACTIVE task definition revision continue to run without disruption. Existing services that reference an INACTIVE task definition revision can still scale up or down by modifying the service's desired count.

You can't use an INACTIVE task definition revision to run new tasks or create new services. You also can't update an existing service to reference an INACTIVE task definition revision (even though there may be up to a 10-minute window following deregistration where these restrictions have not yet taken effect).

Note

When you deregister all revisions in a task family, the task definition family is moved to the INACTIVE list. Adding a new revision of an INACTIVE task definition moves the task definition family back to the ACTIVE list.

At this time, INACTIVE task definition revisions remain discoverable in your account indefinitely. However, this behavior is subject to change in the future. Therefore, you should not rely on INACTIVE task definition revisions persisting beyond the lifecycle of any associated tasks and services.

AWS CloudFormation stacks

The following behavior applies to task definitions created in the new console before January 12, 2023.

When you create a task definition, the Amazon ECS console automatically creates a CloudFormation stack that has a name that begins with "ECS-Console-V2-TaskDefinition-". If you used the AWS CLI or SDK to deregister the task definition, then you must manually delete the task definition stack. For more information, see [Deleting a Stack](#) in the *AWS CloudFormation User Guide*.

Task definitions created after January 12, 2023 will not have a CloudFormation stack automatically created.

To deregister a new task definition (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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 family that contains one or more revisions that you want to deregister.
5. On the **task definition Name** page, select the revisions to delete, and then choose **Actions**, **Deregister**.
6. Verify the information in the **Deregister** window, and then choose **Deregister** to finish.

Deleting a task definition revision using the console

If you decide that you no longer need a specific task definition revision in Amazon ECS, you can delete the task definition revision so that it no longer displays in your `ListTaskDefinitions` API calls or in the console when you want to run a task or update a service.

When you delete a task definition revision, it immediately transitions from the INACTIVE to `DELETE_IN_PROGRESS`. Existing tasks and services that reference a `DELETE_IN_PROGRESS` task definition revision continue to run without disruption.

You can't use a `DELETE_IN_PROGRESS` task definition revision to run new tasks or create new services. You also can't update an existing service to reference a `DELETE_IN_PROGRESS` task definition revision.

When you delete all INACTIVE task definition revisions, the task definition name is not displayed in the console and not returned in the API. If a task definition revisions is in the `DELETE_IN_PROGRESS` state, the task definition name is displayed in the console and returned in the API. The task definition name is retained by Amazon ECS and the revision is incremented the next time you create a task definition with that name.

Amazon ECS resources that can block a deletion

A task definition deletion request will not complete if there are any Amazon ECS resources that depend on the task definition revision. The following resources might prevent a task definition from being deleted:

- Amazon ECS tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS deployments and task sets - The task definition is required when a scaling event is initiated for an Amazon ECS deployment or task set.

If your task definition remains in the `DELETE_IN_PROGRESS` state, you can use the console, or the AWS CLI to identify, and then stop the resources which block the task definition deletion.

Task definition deletion after the blocked resource is removed

The following rules apply after you remove the resources that block the task definition deletion:

- Amazon ECS tasks - The task definition deletion can take up to 1 hour to complete after the task is stopped.
- Amazon ECS deployments and task sets - The task definition deletion can take up to 24 hours to complete after the deployment or task set is deleted.

To delete task definitions (Amazon ECS console)

You must deregister a task definition revision before you delete it. For more information, see [the section called "Deregistering a task definition revision using the console" \(p. 140\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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 family that contains one or more revisions that you want to delete.
5. On the **task definition Name** page, select the revisions to delete, and then choose **Actions, Delete**.
6. Verify the information in the **Delete** window, and then choose **Delete** to finish.

Task definition use cases

Learn more about how to write task definitions for various AWS services and features.

Topics

- [Working with GPUs on Amazon ECS \(p. 142\)](#)
- [Using video transcoding on Amazon ECS \(p. 146\)](#)
- [Using machine learning on Amazon ECS \(p. 153\)](#)
- [Working with 64-bit ARM workloads on Amazon ECS \(p. 159\)](#)
- [Using the awslogs log driver \(p. 161\)](#)
- [Custom log routing \(p. 166\)](#)
- [Private registry authentication for tasks \(p. 195\)](#)
- [Passing environment variables to a container \(p. 197\)](#)
- [Passing sensitive data to a container \(p. 200\)](#)

Working with GPUs on Amazon ECS

Amazon ECS supports workloads that use GPUs, when you create clusters with container instances that support GPUs. Amazon EC2 GPU-based container instances that use the p2, p3, g3, g4, g5, and g5g instance types provide access to NVIDIA GPUs. For more information, see [Linux Accelerated Computing Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

Amazon ECS provides a GPU-optimized AMI that comes with pre-configured NVIDIA kernel drivers and a Docker GPU runtime. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

You can designate a number of GPUs in your task definition for task placement consideration at a container level. Amazon ECS schedules to available container instances that support GPUs and pin physical GPUs to proper containers for optimal performance.

The following Amazon EC2 GPU-based instance types are supported. For more information, see [Amazon EC2 P2 Instances](#), [Amazon EC2 P3 Instances](#), [Amazon EC2 P4d Instances](#), [Amazon EC2 G3 Instances](#), [Amazon EC2 G4 Instances](#), [Amazon EC2 G5 Instances](#), and [Amazon EC2 G5g Instances](#).

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
p2.xlarge	1	12	4	61
p2.8xlarge	8	96	32	488
p2.16xlarge	16	192	64	732
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
p4d.24xlarge	8	320	96	1152
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192
g4dn.16xlarge	1	16	64	256
g5.xlarge	1	24	4	16
g5.2xlarge	1	24	8	32
g5.4xlarge	1	24	16	64
g5.8xlarge	1	24	32	128
g5.16xlarge	1	24	64	256
g5.12xlarge	4	96	48	192
g5.24xlarge	4	96	96	384
g5.48xlarge	8	192	192	768
g5g.xlarge	1	24	4	16
g5g.2xlarge	1	16	8	16

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
g5g.4xlarge	1	16	16	32
g5g.8xlarge	1	16	32	64
g5g.16xlarge	1	16	64	128

Topics

- [Considerations \(p. 144\)](#)
- [Specifying GPUs in your task definition \(p. 145\)](#)

Considerations

We recommend that you consider the following before you begin working with GPUs on Amazon ECS.

- Your clusters can contain a mix of GPU and non-GPU container instances.
- You can run GPU workloads on external instances. When registering an external instance with your cluster, ensure the `--enable-gpu` flag is included on the installation script. For more information, see [Registering an external instance to a cluster \(p. 342\)](#).
- You must set `ECS_ENABLE_GPU_SUPPORT` to `true` in your agent configuration file. For more information, see [the section called "Container agent configuration" \(p. 370\)](#).
- When running a task or creating a service, you can use instance type attributes when you configure task placement constraints to determine the container instances the task is to be launched on. By doing this, you can more effectively use your resources. For more information, see [Amazon ECS task placement \(p. 434\)](#).

The following example launches a task on a `p2.xlarge` container instance in your default cluster.

```
aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == \
    p2.xlarge" --region us-east-2
```

- For each container that has a GPU resource requirement that's specified in the container definition, Amazon ECS sets the container runtime to be the NVIDIA container runtime.
- The NVIDIA container runtime requires some environment variables to be set in the container to function properly. For a list of these environment variables, see [nvidia-container-runtime](#). Amazon ECS sets the `NVIDIA_VISIBLE_DEVICES` environment variable value to be a list of the GPU device IDs that Amazon ECS assigns to the container. For the other required environment variables, Amazon ECS doesn't set them. So, make sure that your container image sets them or they're set in the container definition.
- The `g4` instance type family is supported on version 20190913 and later of the Amazon ECS GPU-optimized AMI. For more information, see [Amazon ECS-optimized AMI changelog \(p. 258\)](#). It's not supported in the Create Cluster workflow in the Amazon ECS console. To use these instance types, you must either use the Amazon EC2 console, AWS CLI, or API and manually register the instances to your cluster.
- The `p4d.24xlarge` instance type only works with CUDA 11 or later.
- The Amazon ECS GPU-optimized AMI has IPv6 enabled, which causes issues when using yum. This can be resolved by configuring yum to use IPv4 with the following command.

```
echo "ip_resolve=4" >> /etc/yum.conf
```

- When you build a container image that doesn't use the NVIDIA/CUDA base images, you must set the NVIDIA_DRIVER_CAPABILITIES container runtime variable to one of the following values:
 - utility, compute
 - all
- For information about how to set the variable, see [Controlling the NVIDIA Container Runtime](#) on the NVIDIA website.
- GPUs are not supported on Windows containers.

Specifying GPUs in your task definition

To use the GPUs on a container instance and the Docker GPU runtime, make sure that you designate the number of GPUs your container requires in the task definition. As containers that support GPUs are placed, the Amazon ECS container agent pins the desired number of physical GPUs to the appropriate container. The number of GPUs reserved for all containers in a task cannot exceed the number of available GPUs on the container instance the task is launched on. For more information, see [Creating a task definition using the console \(p. 125\)](#).

Important

If your GPU requirements aren't specified in the task definition, the task uses the default Docker runtime.

The following shows the JSON format for the GPU requirements in a task definition:

```
{  
    "containerDefinitions": [  
        {  
            ...  
            "resourceRequirements" : [  
                {  
                    "type" : "GPU",  
                    "value" : "2"  
                }  
            ],  
        },  
        ...  
    ]  
}
```

The following example demonstrates the syntax for a Docker container that specifies a GPU requirement. This container uses two GPUs, runs the nvidia-smi utility, and then exits.

```
{  
    "containerDefinitions": [  
        {  
            "memory": 80,  
            "essential": true,  
            "name": "gpu",  
            "image": "nvidia/cuda:11.0.3-base",  
            "resourceRequirements": [  
                {  
                    "type": "GPU",  
                    "value": "2"  
                }  
            ],  
            "command": [  
                "sh",  
                "-c",  
                "nvidia-smi"  
            ],  
            "cpu": 100  
        }  
    ]  
}
```

```
        },
      ],
      "family": "example-ecs-gpu"
}
```

Using video transcoding on Amazon ECS

To use video transcoding workloads on Amazon ECS, register [Amazon EC2 VT1](#) instances. After you registered these instances, you can run live and pre-rendered video transcoding workloads as tasks on Amazon ECS. Amazon EC2 VT1 instances use Xilinx U30 media transcoding cards to accelerate live and pre-rendered video transcoding workloads.

Note

For instructions on how to run video transcoding workloads in containers other than Amazon ECS, see the [Xilinx documentation](#).

Considerations

Before you begin deploying VT1 on Amazon ECS, consider the following:

- Your clusters can contain a mix of VT1 and non-VT1 instances.
- You need a Linux application that uses Xilinx U30 media transcoding cards with accelerated AVC (H.264) and HEVC (H.265) codecs.

Important

Applications that use other codecs might not have improved performance on VT1 instances.

- Only one transcoding task can run on a U30 card. Each card has two devices that are associated with it. You can run as many transcoding tasks as there are cards for each of your VT1 instance.
- When creating a service or running a standalone task, you can use instance type attributes when configuring task placement constraints. This ensures that the task is launched on the container instance that you specify. Doing so helps ensure that you use your resources effectively and that your tasks for video transcoding workloads are on your VT1 instances. For more information, see [Amazon ECS task placement \(p. 434\)](#).

In the following example, a task is run on a `vt1.3xlarge` instance on your default cluster.

```
aws ecs run-task \
  --cluster default \
  --task-definition vt1-3xlarge-xffmpeg-processor \
  --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == vt1.3xlarge"
```

- You configure a container to use the specific U30 card available on the host container instance. You can do this by using the `linuxParameters` parameter and specifying the device details. For more information, see [Task definition requirements \(p. 147\)](#).

Using a VT1 AMI

You have two options for running an AMI on Amazon EC2 for Amazon ECS container instances. The first option is to use the Xilinx official AMI on the AWS Marketplace. The second option is to build your own AMI from the sample repository.

- [Xilinx offers AMIs on the AWS Marketplace](#).
- Amazon ECS provides a sample repository that you can use to build an AMI for video transcoding workloads. This AMI comes with Xilinx U30 drivers. You can find the repository that contains Packer scripts on [GitHub](#). For more information about Packer, see the [Packer documentation](#).

Task definition requirements

To run video transcoding containers on Amazon ECS, your task definition must contain a video transcoding application that uses the accelerated H.264/AVC and H.265/HEVC codecs. You can build a container image by following the steps on the [Xilinx GitHub](#).

The task definition must be specific to the instance type. The instance types are 3xlarge, 6xlarge, and 24xlarge. You must configure a container to use specific Xilinx U30 devices that are available on the host container instance. You can do so using the `linuxParameters` parameter. The following table details the cards and device SoCs that are specific to each instance type.

Instance Type	vCPUs	RAM (GiB)	U30 accelerator cards	Addressable XCU30 SoC devices	Device Paths
vt1.3xlarge	12	24	1	2	/dev/dri/renderD128,/dev/dri/renderD129
vt1.6xlarge	24	48	2	4	/dev/dri/renderD128,/dev/dri/renderD129,/dev/dri/renderD130,/dev/dri/renderD131
vt1.24xlarge	96	182	8	16	/dev/dri/renderD128,/dev/dri/renderD129,/dev/dri/renderD130,/dev/dri/renderD131,/dev/dri/renderD132,/dev/dri/renderD133,/dev/dri/renderD134,/dev/dri/renderD135,/dev/dri/renderD136,/dev/dri/renderD137,/dev/dri/renderD138,/dev/dri/renderD139,/dev/dri/renderD140,/dev/dri/renderD141,/

Instance Type	vCPUs	RAM (GiB)	U30 accelerator cards	Addressable XCU30 SoC devices	Device Paths
					dev/dri/renderD142/ dev/dri/renderD143

Important

If the task definition lists devices that the EC2 instance doesn't have, the task fails to run. When the task fails, the following error message appears in the stoppedReason:
`CannotStartContainerError: Error response from daemon: error gathering device information while adding custom device "/dev/dri/renderD130": no such file or directory.`

In the following example, the syntax that's used for a task definition of a Linux container on Amazon EC2 is provided. This task definition is for container images that are built following the procedure that's provided in the [Xilinx documentation](#). If you use this example, replace `image` with your own image, and copy your video files into the instance in the `/home/ec2-user` directory.

`vt1.3xlarge`

1. Create a text file that's named `vt1-3xlarge-ffmpeg-linux.json` with the following content.

```
{
  "family": "vt1-3xlarge-xffmpeg-processor",
  "requiresCompatibilities": ["EC2"],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": "attribute:ecs.os-type == linux"
    },
    {
      "type": "memberOf",
      "expression": "attribute:ecs.instance-type == vt1.3xlarge"
    }
  ],
  "containerDefinitions": [
    {
      "entryPoint": [
        "/bin/bash",
        "-c"
      ],
      "command": [
        "/video/ecs_ffmpeg_wrapper.sh"
      ],
      "linuxParameters": {
        "devices": [
          {
            "containerPath": "/dev/dri/renderD128",
            "hostPath": "/dev/dri/renderD128",
            "permissions": [
              "read",
              "write"
            ]
          },
          {
            "containerPath": "/dev/dri/renderD129",
            "hostPath": "/dev/dri/renderD129",
            "permissions": [
              "read",
              "write"
            ]
          }
        ]
      }
    }
  ]
}
```

```
        "permissions": [
            "read",
            "write"
        ]
    }
],
"mountPoints": [
{
    "containerPath": "/video",
    "sourceVolume": "video_file"
}
],
"cpu": 0,
"memory": 12000,
"image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-xffmpeg",
"essential": true,
"name": "xilinx-xffmpeg"
],
"volumes": [
{
    "name": "video_file",
    "host": {
        "sourcePath": "/home/ec2-user"
    }
}
]
```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-3xlarge-xffmpeg-processor --cli-
input-json file://vt1-3xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.6xlarge

1. Create a text file that's named vt1-6xlarge-ffmpeg-linux.json with the following content.

```
{
    "family": "vt1-6xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.6xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": [
                "/video/ecs_ffmpeg_wrapper.sh"
            ],
            "environment": [
                {
                    "name": "AWS_ECS_TASK_ARN",
                    "value": "${taskArn}"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-region": "us-east-1",
                    "awslogs-group": "ecs-logs",
                    "awslogs-stream-prefix": "task"
                }
            }
        }
    ]
}
```

```

"linuxParameters": {
  "devices": [
    {
      "containerPath": "/dev/dri/renderD128",
      "hostPath": "/dev/dri/renderD128",
      "permissions": [
        "read",
        "write"
      ]
    },
    {
      "containerPath": "/dev/dri/renderD129",
      "hostPath": "/dev/dri/renderD129",
      "permissions": [
        "read",
        "write"
      ]
    },
    {
      "containerPath": "/dev/dri/renderD130",
      "hostPath": "/dev/dri/renderD130",
      "permissions": [
        "read",
        "write"
      ]
    },
    {
      "containerPath": "/dev/dri/renderD131",
      "hostPath": "/dev/dri/renderD131",
      "permissions": [
        "read",
        "write"
      ]
    }
  ],
  "mountPoints": [
    {
      "containerPath": "/video",
      "sourceVolume": "video_file"
    }
  ],
  "cpu": 0,
  "memory": 12000,
  "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-xffmpeg",
  "essential": true,
  "name": "xilinx-xffmpeg"
},
"volumes": [
  {
    "name": "video_file",
    "host": {
      "sourcePath": "/home/ec2-user"
    }
  }
]
}

```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-6xlarge-xffmpeg-processor --cli-
input-json file://vt1-6xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.24xlarge

1. Create a text file that's named `vt1-24xlarge-ffmpeg-linux.json` with the following content.

```
{  
    "family": "vt1-24xlarge-xffmpeg-processor",  
    "requiresCompatibilities": ["EC2"],  
    "placementConstraints": [  
        {  
            "type": "memberOf",  
            "expression": "attribute:ecs.os-type == linux"  
        },  
        {  
            "type": "memberOf",  
            "expression": "attribute:ecs.instance-type == vt1.24xlarge"  
        }  
    ],  
    "containerDefinitions": [  
        {  
            "entryPoint": [  
                "/bin/bash",  
                "-c"  
            ],  
            "command": [  
                "/video/ecs_ffmpeg_wrapper.sh"  
            ],  
            "linuxParameters": {  
                "devices": [  
                    {  
                        "containerPath": "/dev/dri/renderD128",  
                        "hostPath": "/dev/dri/renderD128",  
                        "permissions": [  
                            "read",  
                            "write"  
                        ]  
                    },  
                    {  
                        "containerPath": "/dev/dri/renderD129",  
                        "hostPath": "/dev/dri/renderD129",  
                        "permissions": [  
                            "read",  
                            "write"  
                        ]  
                    },  
                    {  
                        "containerPath": "/dev/dri/renderD130",  
                        "hostPath": "/dev/dri/renderD130",  
                        "permissions": [  
                            "read",  
                            "write"  
                        ]  
                    },  
                    {  
                        "containerPath": "/dev/dri/renderD131",  
                        "hostPath": "/dev/dri/renderD131",  
                        "permissions": [  
                            "read",  
                            "write"  
                        ]  
                    },  
                    {  
                        "containerPath": "/dev/dri/renderD132",  
                        "hostPath": "/dev/dri/renderD132",  
                        "permissions": [  
                            "read",  
                            "write"  
                        ]  
                    }  
                ]  
            }  
        }  
    ]  
}
```

```
        "read",
        "write"
    ],
},
{
    "containerPath": "/dev/dri/renderD133",
    "hostPath": "/dev/dri/renderD133",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD134",
    "hostPath": "/dev/dri/renderD134",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD135",
    "hostPath": "/dev/dri/renderD135",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD136",
    "hostPath": "/dev/dri/renderD136",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD137",
    "hostPath": "/dev/dri/renderD137",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD138",
    "hostPath": "/dev/dri/renderD138",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD139",
    "hostPath": "/dev/dri/renderD139",
    "permissions": [
        "read",
        "write"
    ],
{
    "containerPath": "/dev/dri/renderD140",
    "hostPath": "/dev/dri/renderD140",
    "permissions": [
        "read",
        "write"
    ]
}
```

```
        ],
      },
      {
        "containerPath": "/dev/dri/renderD141",
        "hostPath": "/dev/dri/renderD141",
        "permissions": [
          "read",
          "write"
        ]
      },
      {
        "containerPath": "/dev/dri/renderD142",
        "hostPath": "/dev/dri/renderD142",
        "permissions": [
          "read",
          "write"
        ]
      },
      {
        "containerPath": "/dev/dri/renderD143",
        "hostPath": "/dev/dri/renderD143",
        "permissions": [
          "read",
          "write"
        ]
      }
    ],
    "mountPoints": [
      {
        "containerPath": "/video",
        "sourceVolume": "video_file"
      }
    ],
    "cpu": 0,
    "memory": 12000,
    "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-xffmpeg",
    "essential": true,
    "name": "xilinx-xffmpeg"
  }
],
"volumes": [
  {
    "name": "video_file",
    "host": {
      "sourcePath": "/home/ec2-user"
    }
  }
]
```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-24xlarge-xffmpeg-processor --cli-input-json file://vt1-24xlarge-xffmpeg-linux.json --region us-east-1
```

Using machine learning on Amazon ECS

To use machine learning workloads on Amazon ECS, register Amazon EC2 instances with specialized hardware, or external computers with specialized hardware, as container instances.

Topics

- [Using AWS Neuron on Amazon Linux 2 on Amazon ECS \(p. 154\)](#)
- [Using deep learning DL1 instances on Amazon ECS \(p. 158\)](#)

Using AWS Neuron on Amazon Linux 2 on Amazon ECS

You can register [Amazon EC2 Trn1](#), [Amazon EC2 Inf1](#), and [Amazon EC2 Inf2](#) instances to your clusters for machine learning workloads.

Amazon EC2 Trn1 instances are powered by [AWS Trainium](#) chips. These instances provide high performance and low cost training for machine learning in the cloud. You can train a machine learning inference model using a machine learning framework with AWS Neuron on a Trn1 instance. Then, you can run the model on a Inf1 instance, or an Inf2 instance to use the acceleration of the AWS Inferentia chips.

The Amazon EC2 Inf1 instances and Inf2 instances are powered by [AWS Inferentia](#) chips. They provide high performance and lowest cost inference in the cloud.

Machine learning models are deployed to containers using [AWS Neuron](#), which is a specialized Software Developer Kit (SDK). The SDK consists of a compiler, runtime, and profiling tools that optimize the machine learning performance of AWS machine learning chips. AWS Neuron supports popular machine learning frameworks such as TensorFlow, PyTorch, and Apache MXNet.

Considerations

Before you begin deploying Neuron on Amazon ECS, consider the following:

- Your clusters can contain a mix of Trn1, Inf1, Inf2 and other instances.
- You need a Linux application in a container that uses a machine learning framework that supports AWS Neuron.

Important

Applications that use other frameworks might not have improved performance on Trn1, Inf1, and Inf2 instances.

- Only one inference or inference-training task can run on each [AWS Trainium](#) or [AWS Inferentia](#) chip. For Inf1, each chip has 4 NeuronCores. For Trn1 and Inf2 each chip has 2 NeuronCores. You can run as many tasks as there are chips for each of your Trn1, Inf1, and Inf2 instances.
- When creating a service or running a standalone task, you can use instance type attributes when you configure task placement constraints. This ensures that the task is launched on the container instance that you specify. Doing so can help you optimize overall resource utilization and ensure that tasks for inference workloads are on your Trn1, Inf1, and Inf2 instances. For more information, see [Amazon ECS task placement \(p. 434\)](#).

In the following example, a task is run on an Inf1.xlarge instance on your default cluster.

```
aws ecs run-task \
    --cluster default \
    --task-definition ecs-inference-task-def \
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == \
    Inf1.xlarge"
```

- Neuron resource requirements can't be defined in a task definition. Instead, you configure a container to use specific AWS Trainium or AWS Inferentia chips available on the host container instance. Do this by using the `linuxParameters` parameter and specifying the device details. For more information, see [Task definition requirements \(p. 155\)](#).

Using the Amazon ECS optimized Amazon Linux 2 (Neuron) AMI

Amazon ECS provides an Amazon ECS optimized AMI that's based on Amazon Linux 2 for AWS Trainium and AWS Inferentia workloads. It comes with the AWS Neuron drivers and runtime for Docker. This AMI makes running machine learning inference workloads easier on Amazon ECS.

We recommend using the Amazon ECS optimized Amazon Linux 2 (Neuron) AMI when launching your Amazon EC2 Trn1, Inf1, and Inf2 instances.

You can retrieve the current Amazon ECS optimized Amazon Linux 2 (Neuron) AMI using the AWS CLI with the following command.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/recommended
```

Task definition requirements

To deploy Neuron on Amazon ECS, your task definition must contain the container definition for a pre-built container serving the inference model for TensorFlow. It's provided by AWS Deep Learning Containers. This container contains the AWS Neuron runtime and the TensorFlow Serving application. At startup, this container fetches your model from Amazon S3, launches Neuron TensorFlow Serving with the saved model, and waits for prediction requests. In the following example, the container image has TensorFlow 1.15 and Ubuntu 18.04. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub. For more information, see [Using AWS Neuron TensorFlow Serving](#).

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04
```

Alternatively, you can build your own Neuron sidecar container image. For more information, see [Tutorial: Neuron TensorFlow Serving](#) in the *AWS Deep Learning AMI Developer Guide*.

The task definition must be specific to a single instance type. You must configure a container to use specific AWS Trainium or AWS Inferentia devices that are available on the host container instance. You can do so using the `linuxParameters` parameter. The following table details the chips that are specific to each instance type.

Instance Type	vCPUs	RAM (GiB)	AWS ML accelerator chips	Device Paths
trn1.2xlarge	8	32	1	/dev/neuron0
trn1.32xlarge	128	512	16	/dev/neuron0, /dev/neuron1, /dev/neuron2, /dev/neuron3, /dev/neuron4, /dev/neuron5, /dev/neuron6, /dev/neuron7, /dev/neuron8, /dev/neuron9, /dev/neuron10, /dev/neuron11, /dev/neuron12, /dev/neuron13, /dev/neuron14, /dev/neuron15

Instance Type	vCPUs	RAM (GiB)	AWS ML accelerator chips	Device Paths
inf1.xlarge	4	8	1	/dev/neuron0
inf1.2xlarge	8	16	1	/dev/neuron0
inf1.6xlarge	24	48	4	/dev/neuron0, /dev/neuron1, /dev/neuron2, /dev/neuron3
inf1.24xlarge	96	192	16	/dev/neuron0, /dev/neuron1, /dev/neuron2, /dev/neuron3, /dev/neuron4, /dev/neuron5, /dev/neuron6, /dev/neuron7, /dev/neuron8, /dev/neuron9, /dev/neuron10, /dev/neuron11, /dev/neuron12, /dev/neuron13, /dev/neuron14, /dev/neuron15
inf2.xlarge	8	16	1	/dev/neuron0
inf2.8xlarge	32	64	1	/dev/neuron0
inf2.24xlarge	96	384	6	/dev/neuron0, /dev/neuron1, /dev/neuron2, /dev/neuron3, /dev/neuron4, /dev/neuron5,
inf2.48xlarge	192	768	12	/dev/neuron0, /dev/neuron1, /dev/neuron2, /dev/neuron3, /dev/neuron4, /dev/neuron5, /dev/neuron6, /dev/neuron7, /dev/neuron8, /dev/neuron9, /dev/neuron10, /dev/neuron11

The following is an example Linux task definition for `inf1.xlarge`, displaying the syntax to use.

```
{
    "family": "ecs-neuron",
```

```

"requiresCompatibilities": ["EC2"],
"placementConstraints": [
    {
        "type": "memberOf",
        "expression": "attribute:ecs.os-type == linux"
    },
    {
        "type": "memberOf",
        "expression": "attribute:ecs.instance-type == inf1.xlarge"
    }
],
"executionRoleArn": "${YOUR_EXECUTION_ROLE}",
"containerDefinitions": [
    {
        "entryPoint": [
            "/usr/local/bin/entrypoint.sh",
            "--port=8500",
            "--rest_api_port=9000",
            "--model_name=resnet50_neuron",
            "--model_base_path=s3://your-bucket-of-models/resnet50_neuron/"
        ],
        "portMappings": [
            {
                "hostPort": 8500,
                "protocol": "tcp",
                "containerPort": 8500
            },
            {
                "hostPort": 8501,
                "protocol": "tcp",
                "containerPort": 8501
            },
            {
                "hostPort": 0,
                "protocol": "tcp",
                "containerPort": 80
            }
        ],
        "linuxParameters": {
            "devices": [
                {
                    "containerPath": "/dev/neuron0",
                    "hostPath": "/dev/neuron0",
                    "permissions": [
                        "read",
                        "write"
                    ]
                }
            ],
            "capabilities": {
                "add": [
                    "IPC_LOCK"
                ]
            }
        },
        "cpu": 0,
        "memoryReservation": 1000,
        "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04",
        "essential": true,
        "name": "resnet50"
    }
]
}

```

Using deep learning DL1 instances on Amazon ECS

To use deep learning workloads on Amazon ECS, register [Amazon EC2 DL1](#) instances to your clusters. Amazon EC2 DL1 instances are powered by Gaudi accelerators from Habana Labs (an Intel company). Use the Habana SynapseAI SDK to connect to the Habana Gaudi accelerators. The SDK supports the popular machine learning frameworks, TensorFlow and PyTorch.

Considerations

Before you begin deploying DL1 on Amazon ECS, consider the following:

- Your clusters can contain a mix of DL1 and non-DL1 instances.
- When creating a service or running a standalone task, you can use instance type attributes specifically when you configure task placement constraints to ensure that your task is launched on the container instance that you specify. Doing so ensures that your resources are used effectively and that your tasks for deep learning workloads are on your DL1 instances. For more information, see [Amazon ECS task placement \(p. 434\)](#).

The following example runs a task on a `dl1.24xlarge` instance on your default cluster.

```
aws ecs run-task \
    --cluster default \
    --task-definition ecs-dl1-task-def \
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == \
    dl1.24xlarge"
```

Using a DL1 AMI

You have three options for running an AMI on Amazon EC2 DL1 instances for Amazon ECS:

- AWS Marketplace AMIs that are provided by Habana [here](#).
- Habana Deep Learning AMIs that are provided by Amazon Web Services. Because it's not included, you need to install the Amazon ECS container agent separately.
- Use Packer to build a custom AMI that's provided by the [GitHub repo](#). For more information, see [the Packer documentation](#).

Task definition requirements

To run Habana Gaudi accelerated deep learning containers on Amazon ECS, your task definition must contain the container definition for a pre-built container that serves the deep learning model for TensorFlow or PyTorch using Habana SynapseAI that's provided by AWS Deep Learning Containers.

The following container image has TensorFlow 2.7.0 and Ubuntu 20.04. A complete list of pre-built Deep Learning Containers that's optimized for the Habana Gaudi accelerators is maintained on GitHub. For more information, see [Habana Training Containers](#).

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training-habana:2.7.0-hpu-py38-synapseai1.2.0-ubuntu20.04
```

The following is an example task definition for Linux containers on Amazon EC2, displaying the syntax to use. This example uses an image containing the Habana Labs System Management Interface Tool (HL-SMI) found here: `vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/tensorflow-installer-tf-cpu-2.6.0:1.1.0-614`

```
{
```

```

"family": "dl-test",
"requiresCompatibilities": ["EC2"],
"placementConstraints": [
    {
        "type": "memberOf",
        "expression": "attribute:ecs.os-type == linux"
    },
    {
        "type": "memberOf",
        "expression": "attribute:ecs.instance-type == dl1.24xlarge"
    }
],
"networkMode": "host",
"cpu": "10240",
"memory": "1024",
"containerDefinitions": [
    {
        "entryPoint": [
            "sh",
            "-c"
        ],
        "command": [
            "hl-smi"
        ],
        "cpu": 8192,
        "environment": [
            {
                "name": "HABANA_VISIBLE_DEVICES",
                "value": "all"
            }
        ],
        "image": "vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/tensorflow-installer-tf-cpu-2.6.0:1.1.0-614",
        "essential": true,
        "name": "tensorflow-installer-tf-hpu"
    }
]
}

```

Working with 64-bit ARM workloads on Amazon ECS

Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by [AWS Graviton2](#) processors,. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.

Considerations

Before you begin deploying task definitions that use the 64-bit ARM architecture, consider the following:

- The applications can use the Fargate or EC2 launch types.
- The applications can only use the Linux operating system.
- For the Fargate type, the applications must use Fargate platform version 1.4.0 or later .
- The applications can use Fluent Bit or CloudWatch for monitoring.
- For the Fargate launch type, the following AWS Regions do not support 64-bit ARM workloads:
 - US East (N. Virginia), the use1-az3 Availability Zone
 - Asia Pacific (Melbourne)

- Asia Pacific (Jakarta)
 - China (Beijing)
 - China (Ningxia)
 - Europe (Zurich)
 - Africa (Cape Town)
 - Middle East (Bahrain)
 - AWS GovCloud (US-East)
 - AWS GovCloud (US-West)
- For the Amazon EC2 launch type, see the following to verify that the Region that you're in supports the instance type you want to use:
- [Amazon EC2 M6g Instances](#)
 - [Amazon EC2 T4g Instances](#)
 - [Amazon EC2 C6g Instances](#)
 - [Amazon EC2 R6gd Instances](#)
 - [Amazon EC2 X2gd Instances](#)

You can also use the `Amazon EC2 describe-instance-type-offerings` command with a filter to view the instance offering for your Region.

```
aws ec2 describe-instance-type-offerings --filters Name=instance-type,Values=instance-type --region region
```

The following example checks for the M6 instance type availability in the US East (N. Virginia) (`us-east-1`) Region.

```
aws ec2 describe-instance-type-offerings --filters Name=instance-type,Values=M6 --region us-east-1
```

For more information, see [describe-instance-type-offerings](#) in the *Amazon EC2 Command Line Reference*.

Specifying the ARM architecture in your task definition

To use the ARM architecture, specify `ARM64` for the `cpuArchitecture` task definition parameter.

In the following example, the ARM architecture is specified in a task definition. It's in JSON format.

```
{  
    "runtimePlatform": {  
        "operatingSystemFamily": "LINUX",  
        "cpuArchitecture": "ARM64"  
    },  
    ...  
}
```

In the following example, a task definition for the ARM architecture displays "hello world."

```
{  
    "family": "arm64-testapp",  
    "networkMode": "awsvpc",  
    "containerDefinitions": [  
        {  
            "name": "arm-container",  
            ...  
        }  
    ]  
}
```

```
        "image": "arm64v8/busybox",
        "cpu": 100,
        "memory": 100,
        "essential": true,
        "command": [ "echo hello world" ],
        "entryPoint": [ "sh", "-c" ]
    }
],
"requiresCompatibilities": [ "FARGATE" ],
"cpu": "256",
"memory": "512",
"runtimePlatform": {
    "operatingSystemFamily": "LINUX",
    "cpuArchitecture": "ARM64"
},
"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
}
```

Interfaces for configuring ARM

You can configure the ARM CPU architecture for Amazon ECS task definitions using one of the following interfaces:

- New Amazon ECS console
- AWS Command Line Interface (AWS CLI)
- AWS SDKs
- AWS Copilot

Using the awslogs log driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. If you're using the Fargate launch type for your tasks, you can view the logs from your containers. If you're using the EC2 launch type, you can view different logs from your containers in one convenient location, and it prevents your container logs from taking up disk space on your container instances. This topic goes over how you can get started using the awslogs log driver in your task definitions.

Note

The type of information that is logged by the containers in your task depends mostly on their ENTRYPPOINT command. By default, the logs that are captured show the command output that you typically might 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 Logs. For more information about 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.

To send system logs from your Amazon ECS container instances to CloudWatch Logs, see [???](#) (p. 354). For more information about CloudWatch Logs, see [Monitoring Log Files](#) and [CloudWatch Logs quotas](#) in the *Amazon CloudWatch Logs User Guide*.

Turning on the awslogs log driver for your containers

If you're using the Fargate launch type for your tasks, you need to add the required logConfiguration parameters to your task definition to turn on the awslogs log driver. For more information, see [Specifying a log configuration in your task definition \(p. 165\)](#).

If you're using the EC2 launch type for your tasks and want to turn on the awslogs log driver, your Amazon ECS container instances require at least version 1.9.0 of the container agent. For information

about how to check your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

Note

If you aren't using the Amazon ECS optimized AMI (with at least version 1.9.0-1 of the `ecs-init` package) for your container instances, you also need to specify that the `awslogs` logging driver is available on the container instance when you start the agent by using the following environment variable in your `docker run` statement or environment variable file. For more information, see [Installing the Amazon ECS container agent \(p. 356\)](#).

```
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
```

Your Amazon ECS container instances also require `logs:CreateLogStream` and `logs:PutLogEvents` permission on the IAM role that you can launch your container instances with. If you created your Amazon ECS container instance role before `awslogs` log driver support was enabled in Amazon ECS, you might need to add this permission. The `ecsTaskExecutionRole` is used when it's assigned to the task and likely contains the correct permissions. For information about checking your task execution role, see [Checking for the task execution \(ecsTaskExecutionRole\) role in the IAM console \(p. 627\)](#). If your container instances use the managed IAM policy for container instances, your container instances likely have the correct permissions. For information about how to check your Amazon ECS container instance role and attach the managed IAM policy for container instances, see [Checking for the container instance \(ecsInstanceRole\) in the IAM console \(p. 639\)](#).

Creating a log group

The `awslogs` log driver can send log streams to an existing log group in CloudWatch Logs or create a new log group on your behalf. The AWS Management Console provides an auto-configure option, which creates a log group on your behalf using the task definition family name with `ecs` as the prefix. Alternatively, you can manually specify your log configuration options and specify the `awslogs-create-group` option with a value of `true`, which creates the log groups on your behalf.

Note

To use the `awslogs-create-group` option to have your log group created, your task execution IAM role policy or EC2 instance role policy must include the `logs:CreateLogGroup` permission.

The following code shows how to set the `awslogs-create-group` option.

```
{
  "containerDefinitions": [
    {
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "firelens-container",
          "awslogs-region": "us-west-2",
          "awslogs-create-group": "true",
          "awslogs-stream-prefix": "firelens"
        }
      }
    }
}
```

Using the auto-configuration feature to create a log group

When you register a task definition, in the Amazon ECS console, you can allow Amazon ECS to auto-configure your CloudWatch logs. Doing this causes a log group to be created on your behalf using the task definition family name with `ecs` as the prefix. For more information, see [the section called "Creating a task definition using the console" \(p. 125\)](#).

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 to be created automatically. If this option isn't specified, it defaults to false.

Note

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

awslogs-region

Required: Yes

Specify the AWS Region that the awslogs log driver is to send your Docker logs to. You can choose to send all of your logs from clusters in different Regions to a single region in CloudWatch Logs. This is so that they're all visible in one location. Otherwise, you can separate them by Region for more granularity. Make sure that the specified log group exists in the Region that you specify with this option.

awslogs-group

Required: Yes

Make sure to specify a log group that the awslogs log driver sends its log streams to. For more information, see [Creating a log group \(p. 162\)](#).

awslogs-stream-prefix

Required: Optional for the EC2 launch type, required for the Fargate launch type.

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

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

If you don't specify a prefix with this option, then the log stream is named after the container ID that's assigned by the Docker daemon on the container instance. Because it's difficult to trace logs back to the container that sent them with just the Docker container ID (which is only available on the container instance), we recommend that you specify a prefix with this option.

For Amazon ECS services, you can use the service name as the prefix. Doing so, you can 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 that the container belongs to.

You must specify a stream-prefix for your logs to have your logs appear in the Log pane when using the Amazon ECS console.

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. 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.

For more information, see [awslogs-datetime-format](#).

You cannot configure both the awslogs-datetime-format and awslogs-multiline-pattern options.

Note

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

`awslogs-multiline-pattern`

Required: No

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

For more information, see [awslogs-multiline-pattern](#).

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

You cannot configure both the awslogs-datetime-format and awslogs-multiline-pattern options.

Note

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

`mode`

Required: No

Valid values: non-blocking | blocking

Default value: blocking

This option defines the delivery mode of log messages from the container to CloudWatch Logs. The delivery mode you choose affects application availability when the flow of logs from container to CloudWatch is interrupted.

If you use the default blocking mode and the flow of logs to CloudWatch is interrupted, calls from container code to write to the `stdout` and `stderr` streams will block. The logging thread of the application will block as a result. This may cause the application to become unresponsive and lead to container healthcheck failure.

If you use the non-blocking mode, the container's logs are instead stored in an in-memory intermediate buffer configured with the `max-buffer-size` option. This prevents the application from becoming unresponsive when logs cannot be sent to CloudWatch.

`max-buffer-size`

Required: No

Default value: 1m

When non-blocking mode is used, the `max-buffer-size` log option controls the size of the buffer that's used for intermediate message storage. Make sure to specify an adequate buffer size. When the buffer fills up, further logs cannot be stored. Logs that cannot be stored are lost.

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 using the console \(p. 125\)](#).

The task definition JSON that follows has a `logConfiguration` object specified for each container. One is for the WordPress container that sends logs to a log group called `awslogs-wordpress`. The other is for a MySQL container that sends logs to a log group that's 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-create-group": "true",  
                    "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-create-group": "true",  
                    "awslogs-group": "awslogs-mysql",  
                    "awslogs-region": "us-west-2",  
                    "awslogs-stream-prefix": "awslogs-example"  
                }  
            }  
        }  
    ],  
    "family": "awslogs-example"  
}
```

After you have registered a task definition with the awslogs log driver in a container definition log configuration, you can run a task or create a service with that task definition to start sending logs to CloudWatch Logs. For more information, see [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#) and [Creating an Amazon ECS service in the classic console \(p. 899\)](#).

Viewing awslogs container logs in CloudWatch Logs

For tasks using the EC2 launch type, after your container instance role has the proper permissions to send logs to CloudWatch Logs, your container agents are updated to at least version 1.9.0, and you have configured and started a task with containers that use the awslogs log driver, your configured containers should be sending their log data to CloudWatch Logs. You can view and search these logs in the console.

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

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster that contains the task to view.
3. On the **Cluster: *cluster_name*** page, choose **Tasks**, and then select the task to view.
4. On the **Task: *task_id*** page, under **Container details**, choose **Log configuration** to view the logs.
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 a log group \(p. 162\)](#).
4. Choose a log stream to view.

Custom log routing

You can use FireLens for Amazon ECS to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. The AWS Partner Network is a global community of partners that leverages programs, expertise, and resources to build, market, and sell customer offerings. For more information see [AWS Partner](#). FireLens works with [Fluentd](#) and [Fluent Bit](#). We provide the AWS for Fluent Bit image or you can use your own Fluentd or Fluent Bit image.

Creating Amazon ECS task definitions with a FireLens configuration is supported using the AWS SDKs, AWS CLI, and AWS Management Console.

Considerations

Consider the following when using FireLens for Amazon ECS:

- FireLens for Amazon ECS is supported for tasks that are hosted on both AWS Fargate on Linux and Amazon EC2 on Linux. Windows containers don't support FireLens.

For information about how to configure centralized logging for Windows containers, see [Centralized logging for Windows containers on Amazon ECS using Fluent Bit](#).

- FireLens for Amazon ECS is supported in AWS CloudFormation templates. For more information, see [AWS::ECS::TaskDefinition FirelensConfiguration](#) in the *AWS CloudFormation User Guide*

- FireLens listens on port 24224, so to ensure that the FireLens log router isn't reachable outside of the task, you must not allow inbound traffic on port 24224 in the security group your task uses. For tasks that use the awsvpc network mode, this is the security group associated with the task. For tasks using the host network mode, this is the security group that's associated with the Amazon EC2 instance hosting the task. For tasks that use the bridge network mode, don't create any port mappings that use port 24224.
- For tasks that use the bridge network mode, the container with the FireLens configuration must start before any application containers that rely on it start. To control the start order of your containers, use dependency conditions in your task definition. For more information, see [Container dependency \(p. 830\)](#).

Note

If you use dependency condition parameters in container definitions with a FireLens configuration, ensure that each container has a START or HEALTHY condition requirement.

- By default, FireLens adds the cluster and task definition name and the Amazon Resource Name (ARN) of the cluster as metadata keys to your stdout/stderr container logs. The following is an example of the metadata format.

```
"ecs_cluster": "cluster-name",
"ecs_task_arn": "arn:aws:ecs:region:111122223333:task/cluster-
name/f2ad7dba413f45ddb4EXAMPLE",
"ecs_task_definition": "task-def-name:revision",
```

If you do not want the metadata in your logs, set enable-ecs-log-metadata to false in the firelensConfiguration section of the task definition.

```
"firelensConfiguration": {
    "type": "fluentbit",
    "options": {
        "enable-ecs-log-metadata": "false",
        "config-file-type": "file",
        "config-file-value": "/extra.conf"
    }
}
```

Required IAM permissions

To use this feature, you must create an IAM role for your tasks that provides the permissions necessary to use any AWS services that the tasks require. For example, if a container is routing logs to Kinesis Data Firehose, the task requires permission to call the firehose:PutRecordBatch API. For more information, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

The following example IAM policy adds the required permissions for routing logs to Kinesis Data Firehose.

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

}

Your task may also require the Amazon ECS task execution role under the following conditions. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

- If your task is hosted on Fargate and you are pulling container images from Amazon ECR or referencing sensitive data from AWS Secrets Manager in your log configuration, then you must include the task execution IAM role.
- If you are specifying a custom configuration file that's hosted in Amazon S3, your task execution IAM role must include the `s3:GetObject` permission for the configuration file and the `s3:GetBucketLocation` permission on the Amazon S3 bucket that the file is in. For more information, see [Specifying Permissions in a Policy](#) in the *Amazon Simple Storage Service User Guide*.

The following example IAM policy adds the required permissions for retrieving a file from Amazon S3. Specify the name of your Amazon S3 bucket and configuration file name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/folder_name/config_file_name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket"
      ]
    }
  ]
}
```

Fluentd buffer limit

When you create a task definition, you can specify the number of events that are buffered in memory by specifying the value (in bytes) in the `log-driver-buffer-limit`. For more information, see [Fluentd logging driver](#) in the Docker documentation.

Use this option when there's high throughput, because Docker might run out of buffer memory and discard buffer messages so it can add new messages. The lost logs might make it difficult to troubleshoot. Setting the buffer limit might help to prevent this issue.

The following shows the syntax for specifying the `log-driver-buffer-limit`:

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-
bit:stable",
      "name": "log_router",
```

```

    "firelensConfiguration": {
        "type": "fluentbit"
    },
    "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
            "awslogs-group": "firelens-container",
            "awslogs-region": "us-west-2",
            "awslogs-create-group": "true",
            "awslogs-stream-prefix": "firelens"
        }
    },
    "memoryReservation": 50
},
{
    "essential": true,
    "image": "httpd",
    "name": "app",
    "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
            "Name": "firehose",
            "region": "us-west-2",
            "delivery_stream": "my-stream",
            "log-driver-buffer-limit": "2097152"
        }
    },
    "dependsOn": [
        {
            "containerName": "log_router",
            "condition": "START"
        }
    ],
    "memoryReservation": 100
}
]
}

```

Consider the following when using FireLens for Amazon ECS with the buffer limit option:

- This option is supported on the Amazon EC2 launch type and the Fargate launch type with platform version 1.4.0 or later.
- The option is only valid when `logDriver` is set to `awsfirelens`.
- The default buffer limit is 1 MiB.
- The valid values are 0 and 536870912 (512 MiB).
- The total amount of memory allocated at the task level must be greater than the amount of memory that's allocated for all the containers in addition to the memory buffer limit. The total amount of buffer memory specified must be less than 536870912 (512MiB) when you don't specify the container memory and `memoryReservertion` values. More specifically, you can have an app container with the `awsfirelens` log driver and the `log-driver-buffer-limit` option set to 300 MiB. However, you won't be allowed to run tasks if you have more than two containers with the `log-driver-buffer-limit` set to 300 MiB ($300 \text{ MiB} * 2 > 512 \text{ MiB}$).

Using Fluent logger libraries or Log4j over TCP

When the `awsfirelens` log driver is specified in a task definition, the Amazon ECS container agent injects the following environment variables into the container:

FLUENT_HOST

The IP address that's assigned to the FireLens container.

FLUENT_PORT

The port that the Fluent Forward protocol is listening on.

You can use the FLUENT_HOST and FLUENT_PORT environment variables to log directly to the log router from code instead of going through stdout. For more information, see [fluent-logger-golang](#) on GitHub.

- [the section called “Using the AWS for Fluent Bit image” \(p. 170\)](#)
- [the section called “Creating a task definition that uses a FireLens configuration” \(p. 172\)](#)
- [the section called “Filtering logs using regular expressions” \(p. 175\)](#)
- [the section called “Example logging option task definitions” \(p. 191\)](#)

Using the AWS for Fluent Bit image

AWS provides a Fluent Bit image with plugins for both CloudWatch Logs and Kinesis Data Firehose. We recommend using Fluent Bit as your log router because it has a lower resource utilization rate than Fluentd. For more information, see [CloudWatch Logs for Fluent Bit](#) and [Amazon Kinesis Firehose for Fluent Bit](#).

The **AWS for Fluent Bit** image is available on Amazon ECR on both the Amazon ECR Public Gallery and in an Amazon ECR repository in most AWS Regions for high availability.

Amazon ECR Public Gallery

The AWS for Fluent Bit image is available on the Amazon ECR Public Gallery. This is the recommended location to download the AWS for Fluent Bit image because it's a public repository and available to be used from all AWS Regions. For more information, see [aws-for-fluent-bit](#) on the Amazon ECR Public Gallery.

Linux

The AWS for Fluent Bit image in the Amazon ECR Public Gallery supports Amazon Linux operating system with the ARM 64, or x86-64 architecture.

You can pull the AWS for Fluent Bit image from the Amazon ECR Public Gallery by specifying the repository URL with the desired image tag. The available image tags can be found on the **Image tags** tab on the Amazon ECR Public Gallery.

The following shows the syntax to use for the Docker CLI.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

For example, you can pull the latest stable AWS for Fluent Bit image using this Docker CLI command.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:stable
```

Note

Unauthenticated pulls are allowed, but have a lower rate limit than authenticated pulls. To authenticate using your AWS account before pulling, use the following command.

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS  
--password-stdin public.ecr.aws
```

Windows

The AWS for Fluent Bit image in the Amazon ECR Public Gallery supports the AMD64 architecture with the following operating systems:

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

Windows containers that are on AWS Fargate don't support FireLens.

You can pull the AWS for Fluent Bit image from the Amazon ECR Public Gallery by specifying the repository URL with the desired image tag. The available image tags can be found on the **Image tags** tab on the Amazon ECR Public Gallery.

The following shows the syntax to use for the Docker CLI.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

For example, you can pull the newest stable AWS for Fluent Bit image using this Docker CLI command.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:windowsservercore-stable
```

Note

Unauthenticated pulls are allowed, but have a lower rate limit than authenticated pulls. To authenticate using your AWS account before pulling, use the following command.

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS  
--password-stdin public.ecr.aws
```

Amazon ECR

The AWS for Fluent Bit image is available on Amazon ECR for high availability. These images are available in most AWS Regions, including AWS GovCloud (US).

Linux

The latest stable AWS for Fluent Bit image URI can be retrieved using the following command.

```
aws ssm get-parameters \  
  --names /aws/service/aws-for-fluent-bit/stable \  
  --region us-east-1
```

All versions of the AWS for Fluent Bit image can be listed using the following command to query the Systems Manager Parameter Store parameter.

```
aws ssm get-parameters-by-path \  
  --path /aws/service/aws-for-fluent-bit \  
  --region us-east-1
```

```
--region us-east-1
```

The newest stable AWS for Fluent Bit image can be referenced in an AWS CloudFormation template by referencing the Systems Manager parameter store name. The following is an example:

```
Parameters:  
FireLensImage:  
  Description: Fluent Bit image for the FireLens Container  
  Type: AWS::SSM::Parameter::Value<String>  
  Default: /aws/service/aws-for-fluent-bit/stable
```

Windows

The latest stable AWS for Fluent Bit image URI can be retrieved using the following command.

```
aws ssm get-parameters \  
  --names /aws/service/aws-for-fluent-bit/windowsservercore:stable \  
  --region us-east-1
```

All versions of the AWS for Fluent Bit image can be listed using the following command to query the Systems Manager Parameter Store parameter.

```
aws ssm get-parameters-by-path \  
  --path /aws/service/aws-for-fluent-bit/windowsservercore \  
  --region us-east-1
```

The latest stable AWS for Fluent Bit image can be referenced in an AWS CloudFormation template by referencing the Systems Manager parameter store name. The following is an example:

```
Parameters:  
FireLensImage:  
  Description: Fluent Bit image for the FireLens Container  
  Type: AWS::SSM::Parameter::Value<String>  
  Default: /aws/service/aws-for-fluent-bit/windowsservercore:stable
```

Creating a task definition that uses a FireLens configuration

To use custom log routing with FireLens, you must specify the following in your task definition:

- A log router container that contains a FireLens configuration. We recommend that the container be marked as **essential**.
- One or more application containers that contain a log configuration specifying the `awsfirelens` log driver.
- A task IAM role Amazon Resource Name (ARN) that contains the permissions needed for the task to route the logs.

When creating a new task definition using the AWS Management Console, there is a FireLens integration section that makes it easy to add a log router container. For more information, see [Creating a task definition using the console \(p. 125\)](#).

Amazon ECS converts the log configuration and generates the Fluentd or Fluent Bit output configuration. The output configuration is mounted in the log routing container at `/fluent-bit/etc/fluent-bit.conf` for Fluent Bit and `/fluentd/etc/fluent.conf` for Fluentd.

Important

FireLens listens on port 24224. Therefore, to ensure that the FireLens log router isn't reachable outside of the task, you must not allow ingress traffic on port 24224 in the security group your task uses. For tasks that use the `awsvpc` network mode, this is the security group that's associated with the task. For tasks that use the `host` network mode, this is the security group that's associated with the Amazon EC2 instance hosting the task. For tasks that use the `bridge` network mode, don't create any port mappings that use port 24224.

The following task definition example defines a log router container that uses Fluent Bit to route its logs to CloudWatch Logs. It also defines an application container that uses a log configuration to route logs to Amazon Kinesis Data Firehose and sets the memory that's used to buffer events to the 2 MiB.

```
{
    "family": "firelens-example-firehose",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit"
            },
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "firelens-container",
                    "awslogs-region": "us-west-2",
                    "awslogs-create-group": "true",
                    "awslogs-stream-prefix": "firelens"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "httpd",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "firehose",
                    "region": "us-west-2",
                    "delivery_stream": "my-stream",
                    "log-driver-buffer-limit": "2097152"
                }
            },
            "memoryReservation": 100
        }
    ]
}
```

The key-value pairs specified as options in the `logConfiguration` object are used to generate the Fluentd or Fluent Bit output configuration. The following is a code example from a Fluent Bit output definition.

```
[OUTPUT]
  Name    firehose
  Match   app-firelens*
  region  us-west-2
  delivery_stream my-stream
```

Note

FireLens manages the match configuration. This configuration isn't specified in your task definition.

Using Amazon ECS metadata

When specifying a FireLens configuration in a task definition, you can optionally toggle the value for `enable-ecs-log-metadata`. By default, Amazon ECS adds additional fields in your log entries that help identify the source of the logs. You can turn off this action by setting `enable-ecs-log-metadata` to `false`.

- `ecs_cluster` – The name of the cluster that the task is part of.
- `ecs_task_arn` – The full Amazon Resource Name (ARN) of the task that the container is part of.
- `ecs_task_definition` – The task definition name and revision that the task is using.
- `ec2_instance_id` – The Amazon EC2 instance ID that the container is hosted on. This field is only valid for tasks using the EC2 launch type.

The following shows the syntax required when specifying an Amazon ECS log metadata setting value.

```
{  
    "containerDefinitions": [  
        {  
            "essential": true,  
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",  
            "name": "log_router",  
            "firelensConfiguration": {  
                "type": "fluentbit",  
                "options": {  
                    "enable-ecs-log-metadata": "true | false"  
                }  
            }  
        }  
    ]  
}
```

Specifying a custom configuration file

In addition to the auto-generated configuration file that FireLens creates on your behalf, you can also specify a custom configuration file. The configuration file format is the native format for the log router that you're using. For more information, see [Fluentd Config File Syntax](#) and [Fluent Bit Configuration File](#).

In your custom configuration file, for tasks using the bridge or awsvpc network mode, don't set a Fluentd or Fluent Bit forward input over TCP because FireLens adds it to the input configuration.

Your FireLens configuration must contain the following options to specify a custom configuration file:

`config-file-type`

The source location of the custom configuration file. The available options are `s3` or `file`.

Note

Tasks that are hosted on AWS Fargate only support the `file` configuration file type.

`config-file-value`

The source for the custom configuration file. If the `s3` config file type is used, the config file value is the full ARN of the Amazon S3 bucket and file. If the `file` config file type is used, the config file value is the full path of the configuration file that exists either in the container image or on a volume that's mounted in the container.

Important

When using a custom configuration file, you must specify a different path than the one FireLens uses. Amazon ECS reserves the /fluent-bit/etc/fluent-bit.conf filepath for Fluent Bit and /fluentd/etc/fluent.conf for Fluentd.

The following example shows the syntax required when specifying a custom configuration.

Important

To specify a custom configuration file that's hosted in Amazon S3, ensure you have created a task execution IAM role with the proper permissions. For more information, see [Required IAM permissions \(p. 167\)](#).

The following shows the syntax required when specifying a custom configuration.

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "config-file-type": "s3 | file",
          "config-file-value": "arn:aws:s3:::mybucket/fluent.conf | filepath"
        }
      }
    }
  ]
}
```

Note

Tasks hosted on AWS Fargate only support the file configuration file type.

Filtering logs using regular expressions

Fluentd and Fluent Bit both support filtering of logs based on their content. FireLens provides a simple method for enabling this filtering. In the log configuration options in a container definition, you can specify the special keys exclude-pattern and include-pattern that take regular expressions as their values. The exclude-pattern key causes all logs that match its regular expression to be dropped. With include-pattern, only logs that match its regular expression are sent. These keys can be used together.

The following example demonstrates how to use this filter.

```
{
  "containerDefinitions": [
    {
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
          "@type": "cloudwatch_logs",
          "log_group_name": "firelens-testing",
          "auto_create_stream": "true",
          "use_tag_as_stream": "true",
          "region": "us-west-2",
          "exclude-pattern": "[a-z][aeiou].*$",
          "include-pattern": "^.*[aeiou]$"
        }
      }
    }
  ]
}
```

```
    ]  
}
```

Concatenate multiline or stack-trace log messages

Beginning with AWS for Fluent Bit version 2.22.0, a multiline filter is included. The multiline filter helps concatenate log messages that originally belong to one context but were split across multiple records or log lines. For more information about the multiline filter, see the [Fluent Bit documentation](#).

Common examples of split log messages are:

- Stack traces.
- Applications that print logs on multiple lines.
- Log messages that were split because they were longer than the specified runtime max buffer size.
You can concatenate log messages split by the container runtime by following the example on GitHub: [FireLens Example: Concatenate Partial/Split Container Logs](#).

Required IAM permissions

You have the necessary IAM permissions for the container agent to pull the container images from Amazon ECR and for the container to route logs to CloudWatch Logs.

For these permissions, you must have the following roles:

- A task IAM role.
- An Amazon ECS task execution IAM role.

This task role grants the FireLens log router container the permissions needed to route the logs to the destination. In this example we are routing the logs to CloudWatch Logs. To create this role, create a policy with the permissions to create a log stream, log group, and write log events. Then associate the policy to the role.

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. Choose **JSON** and paste the following permissions:

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

4. Choose **Next: Tags** and add any tags to the policy to help you organize them. Then choose **Next: Review**.
5. On the **Review policy** page, for **Name** type a unique name for the policy. In this example, we will use `ecs-policy-for-firelens`. You may specify an optional description for the policy as well.
6. Choose **Create policy** to finish.
7. In the navigation pane, choose **Roles** and then choose **Create Roles**.

8. In the **Trusted entity type** section, choose **AWS service**.
9. For **Use case**, choose **Elastic Container Service..**
10. Choose **Elastic Container Service Task** and then **Next**.
11. Associate the role with the `ecs-policy-for-fielens` policy you created and choose **Next**.
12. Enter a unique name for the role. In this example, use: **ecs-task-role-for-fielens**.

Verify that you have an Amazon ECS task execution IAM role

You must have a task execution role to grant the container agent permission to pull the container images from Amazon ECR.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then search for `ecsTaskExecutionRole`.
3. If you do not see the `ecsTaskExecutionRole` role, you must create the role. For information about how to create the role, see [Amazon ECS task execution IAM role](#) in the *Amazon Elastic Container Service Developer Guide*.

Determine when to use the multiline log setting

The following are example log snippets that you see in the CloudWatch Logs console with the default log setting. You can look at the line that starts with `log` to determine if you need the multiline filter. When the context is the same, you can use the multiline log setting. In this example, the context is "

```
2022-09-20T15:47:56:595-05-00 {"container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "source": "stdout", "log": "": "      at com.myproject.modele.(MyProject.badMethod.java:22)", {
    "container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
    "container_name": "": "example-app",
    "source": "stdout",
    "log": "": "      at com.myproject.model.MyProject.badMethod(MyProject.java:22)",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:region:123456789012:task/default/b23c940d29ed4714971cba72cEXAMPLE",
    "ecs_task_definition": "fielense-example-multiline:3"
}
```

```
2022-09-20T15:47:56:595-05-00 {"container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "stdout", "log": "": "      at com.myproject.modele.(MyProject.oneMoreMethod.java:18)", {
    "container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
    "container_name": "": "example-app",
    "source": "stdout",
    "log": "": "      at com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:region:123456789012:task/default/b23c940d29ed4714971cba72cEXAMPLE",
    "ecs_task_definition": "fielense-example-multiline:3"
}
```

After you use the multiline log setting, the output will look similar to the example below.

```
2022-09-20T15:47:56:595-05-00 {"container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "stdout", ...
```

```
{  
    "container_id": "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",  
    "container_name": ": \"example-app\"",  
    "source": "stdout",  
    "log": "September 20, 2022 06:41:48 Exception in thread \"main\"  
java.lang.RuntimeException: Something has gone wrong, aborting!\n        at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n        at com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)  
com.myproject.module.MyProject.main(MyProject.java:6)",  
    "ecs_cluster": "default",  
    "ecs_task_arn": "arn:aws:region:123456789012:task/default/  
b23c940d29ed4714971cba72cEXAMPLE",  
    "ecs_task_definition": "firelense-example-multiline:2"  
}
```

Parse and concatenate options

To parse logs and concatenate lines that were split because of newlines, you can use either of these two options.

- Use your own parser file that contains the rules to parse and concatenate lines that belong to the same message.
- Use a Fluent Bit built-in parser. For a list of languages supported by the Fluent Bit built-in parsers, see [Fluent Bit documentation](#).

The following tutorial walks you through the steps for each use case. The steps show you how to concatenate multilines and send the logs to Amazon CloudWatch. You can specify a different destination for your logs.

Example: Use a parser that you create

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

Build and upload the image for a Fluent Bit container

This image will include the parser file where you specify the regular expression and a configuration file that references the parser file.

1. Create a folder with the name `FluentBitDockerImage`.
2. Within the folder, create a parser file that contains the rules to parse the log and concatenate lines that belong in the same message.
 - a. Paste the following contents in the parser file:

```
[MULTILINE_PARSER]  
name      multiline-regex-test  
type      regex  
flush_timeout 1000  
#
```

```

# Regex rules for multiline parsing
# -----
#
# configuration hints:
#
# - first state always has the name: start_state
# - every field in the rule must be inside double quotes
#
# rules |   state name   | regex pattern           | next state
# -----|-----|-----|
rule      "start_state"  "/(Dec \d+ \d+:\d+:\d+)(.*)/"  "cont"
rule      "cont"         "/^\s+at.*/"                  "cont"

```

As you customize your regex pattern, we recommend you use a regular expression editor to test the expression.

- b. Save the file as `parsers_multiline.conf`.
3. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the parser file that you created in the previous step.

For more information about the custom configuration file, see [Specifying a custom configuration file](#) in the *Amazon Elastic Container Service Developer Guide*

- a. Paste the following contents in the file:

```

[SERVICE]
flush          1
log_level      info
parsers_file   /parsers_multiline.conf

[FILTER]
name           multiline
match          *
multiline.key_content log
multiline.parser   multiline-regex-test

```

Note

You must use the absolute path of the parser.

- b. Save the file as `extra.conf`.
4. Within the `FluentBitDockerImage` folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.
- a. Paste the following contents in the file:

```

FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest

ADD parsers_multiline.conf /parsers_multiline.conf
ADD extra.conf /extra.conf

```

- b. Save the file as `Dockerfile`.

5. Using the Dockerfile, build a custom Fluent Bit image with the parser and custom configuration files included.

Note

You can place the parser file and configuration file anywhere in the Docker image except `/fluent-bit/etc/fluent-bit.conf` as this file path is used by FireLens.

- a. Build the image: `docker build -t fluent-bit-multiline-image .`

Where: `fluent-bit-multiline-image` is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=fluent-bit-multiline-image`
If successful, the output shows the image and the latest tag.
6. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.
 - a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`
Where: fluent-bit-multiline-repo is the name for the repository and us-east-1 is the region in this example.
The output gives you the details of the new repository.
 - b. Tag your image with the `repositoryUri` value from the previous output: `docker tag fluent-bit-multiline-image repositoryUri`
Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`
 - c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`
In the output, the repository name changes from fluent-bit-multiline-repo to the `repositoryUri`.
 - d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-password | docker login --username AWS --password-stdin registry-ID.dkr.ecr.region.amazonaws.com`
Example: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`
A successful login message appears.
 - e. Push the image to Amazon ECR: `docker push registry-ID.dkr.ecr.region.amazonaws.com/repository name`
Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

When you run the task, the application simulates runs, then fails and creates a stack trace.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.
 - a. Within the `multiline-app` folder, create a file and name it `main.py`.
 - b. Paste the following contents in the file:

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0
```

```

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")

```

- c. Save the main.py file.
3. Create a sample log file.
 - a. Within the multiline-app folder, create a file and name it test.log.
 - b. Paste the following contents in the file:

```

single line...
Dec 14 06:41:08 Exception in thread "main" java.lang.RuntimeException: Something
has gone wrong, aborting!
at com.myproject.module.MyProject.badMethod(MyProject.java:22)
at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)
at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)
at com.myproject.module.MyProject.someMethod(MyProject.java:10)
at com.myproject.module.MyProject.main(MyProject.java:6)
another line...

```

- c. Save the test.log file.
4. Within the multiline-app folder, create the Dockerfile.
 - a. Paste the following contents in the file:

```

FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]

```

- b. Save the Dockerfile file.
5. Using the Dockerfile, build an image.
 - a. Build the image: docker build -t multiline-app-image .
Where: multiline-app-image is the name for the image in this example.
 - b. Verify that the image was created correctly: docker images --filter reference=multiline-app-image

If successful, the output shows the image and the latest tag.
6. Upload the image to Amazon Elastic Container Registry.
 - a. Create an Amazon ECR repository to store the image: aws ecr create-repository --repository-name multiline-app-repo --region us-east-1

Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.

The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag multiline-app-image repositoryUri`

Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.

- d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN",
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            }
        }
    ]
}
```

```

        },
        "memoryReservation": 100
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}

```

Replace the following in the `multiline-task-definition.json` task definition:

a. *task role ARN*

To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. *execution role ARN*

To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. *aws_account_id*

To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. *us-east-1*

Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region region`
4. Open the console at <https://console.aws.amazon.com/ecs/v2>.
5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.
6. Choose the latest version.
7. Choose the **Deploy, Run task**.
8. On the **Run Task** page, For **Cluster**, choose the cluster, and then under **Networking**, for **Subnets**, choose the available subnets for your task.
9. Choose **Create**.

Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation pane, expand **Logs** and choose **Log groups**.
3. Choose the `multiline-test/applicatio` log group.
4. Choose the log. View messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

The following log snippet shows lines concatenated in a single Java stack trace event:

```
{
    "container_id": "xxxxxx",
    "container_name": "app",

```

```
        "source": "stdout",
        "log": "Dec 14 06:41:08 Exception in thread \"main\"  
java.lang.RuntimeException: Something has gone wrong, aborting!\n    at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)\n    at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)\n    at com.myproject.module.MyProject.someMethod(MyProject.java:10)\n    at com.myproject.module.MyProject.main(MyProject.java:6)",
        "ecs_cluster": "default",
        "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
        "ecs_task_definition": "firelens-example-multiline:2"
}
```

The following log snippet shows how the same message appears with just a single line if you run an Amazon ECS container that is not configured to concatenate multiline log messages.

```
{
    "log": "Dec 14 06:41:08 Exception in thread \"main\" java.lang.RuntimeException:  
Something has gone wrong, aborting!",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
}
```

Example: Use a Fluent Bit built-in parser

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

Build and upload the image for a Fluent Bit container

This image will include a configuration file that references the Fluent Bit parser.

1. Create a folder with the name `FluentBitDockerImage`.
2. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the Fluent Bit built-in parser file.

For more information about the custom configuration file, see [Specifying a custom configuration file](#) in the *Amazon Elastic Container Service Developer Guide*

- a. Paste the following contents in the file:

```
[FILTER]
name          multiline
match         *
multiline.key_content log
multiline.parser   go
```

- b. Save the file as `extra.conf`.

3. Within the FluentBitDockerImage folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.

- a. Paste the following contents in the file:

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest
ADD extra.conf /extra.conf
```

- b. Save the file as Dockerfile.

4. Using the Dockerfile, build a custom Fluent Bit image with the custom configuration file included.

Note

You can place the configuration file anywhere in the Docker image except /fluent-bit/etc/fluent-bit.conf as this file path is used by FireLens.

- a. Build the image: `docker build -t fluent-bit-multiline-image .`

Where: fluent-bit-multiline-image is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=fluent-bit-multiline-image`

If successful, the output shows the image and the latest tag.

5. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.

- a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

Where: fluent-bit-multiline-repo is the name for the repository and us-east-1 is the region in this example.

The output gives you the details of the new repository.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag fluent-bit-multiline-image repositoryUri`

Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from fluent-bit-multiline-repo to the `repositoryUri`.

- d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-password | docker login --username AWS --password-stdin registry ID.dkr.ecr.region.amazonaws.com`

Example: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

A successful login message appears.

- e. Push the image to Amazon ECR: `docker push registry ID.dkr.ecr.region.amazonaws.com/repository name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.
 - a. Within the `multiline-app` folder, create a file and name it `main.py`.
 - b. Paste the following contents in the file:

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

- c. Save the `main.py` file.
3. Create a sample log file.
 - a. Within the `multiline-app` folder, create a file and name it `test.log`.
 - b. Paste the following contents in the file:

```
panic: my panic

goroutine 4 [running]:
panic(0x45cb40, 0x47ad70)
    /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8 sp=0xc42003f710
    pc=0x422f7c
main.main.func1(0xc420024120)
    foo.go:6 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8
    pc=0x44b4d1
created by main.main
    foo.go:5 +0x58

goroutine 1 [chan receive]:
runtime.gopark(0x4739b8, 0xc420024178, 0x46fc7, 0xc, 0xc420028e17, 0x3)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00
    pc=0x42503c
runtime.goparkunlock(0xc420024178, 0x46fc7, 0xc, 0x1000f010040c217, 0x3)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
    pc=0x42512e
runtime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)
    /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
    pc=0x4046b4
runtime.chanrecv1(0xc420024120, 0x0)
    /usr/local/go/src/runtime/chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20
    pc=0x40439b
```

```

main.main()
    foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef
runtime.main()
    /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
    pc=0x424bad
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc420053fe8 sp=0xc420053fe0
    pc=0x44b4d1

goroutine 2 [force gc (idle)]:
runtime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
    pc=0x42503c
runtime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
    pc=0x42512e
runtime.forcegchelper()
    /usr/local/go/src/runtime/proc.go:238 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8
    pc=0x424e5c
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0
    pc=0x44b4d1
created by runtime.init.4
    /usr/local/go/src/runtime/proc.go:227 +0x35

goroutine 3 [GC sweep wait]:
runtime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30
    pc=0x42503c
runtime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60
    pc=0x42512e
runtime.bgsweep(0xc42001e150)
    /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8 sp=0xc42003efa0
    pc=0x419973
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0 sp=0xc42003efd8
    pc=0x44b4d1
created by runtime.gcenable
    /usr/local/go/src/runtime/mgc.go:216 +0x58
one more line, no multiline

```

- c. Save the test.log file.
4. Within the multiline-app folder, create the Dockerfile.
- a. Paste the following contents in the file:

```

FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]

```

- b. Save the Dockerfile file.
5. Using the Dockerfile, build an image.
- a. Build the image: docker build -t multiline-app-image .

Where: `multiline-app-image` is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=multiline-app-image`

If successful, the output shows the image and the latest tag.

6. Upload the image to Amazon Elastic Container Registry.

- a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.

The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag multiline-app-image repositoryUri`

Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.

- d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository_name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN",
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-log-forwarder:latest",
            "name": "log_forwarder",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            }
        }
    ]
}
```

```

        "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "cloudwatch_logs",
                "region": "us-east-1",
                "log_group_name": "multiline-test/application",
                "auto_create_group": "true",
                "log_stream_prefix": "multiline-"
            }
        },
        "memoryReservation": 100
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}

```

Replace the following in the `multiline-task-definition.json` task definition:

a. ***task role ARN***

To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. ***execution role ARN***

To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. ***aws_account_id***

To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. ***us-east-1***

Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. Open the console at <https://console.aws.amazon.com/ecs/v2>.
5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.
6. Choose the latest version.
7. Choose the **Deploy, Run task**.
8. On the **Run Task** page, For **Cluster**, choose the cluster, and then under **Networking**, for **Subnets**, choose the available subnets for your task.
9. Choose **Create**.

Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation pane, expand **Logs** and choose **Log groups**.

3. Choose the `multiline-test/applicatio` log group.
4. Choose the log and view the messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

The following log snippet shows a Go stack trace that is concatenated into a single event:

```
{
    "log": "panic: my panic\n\ngoroutine 4 [running]:\npanic(0x45cb40,\n0x47ad70)\n    /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8\nsp=0xc42003f710 pc=0x422f7c\nmain.main.func1(0xc420024120)\n    foo.go:6\n    +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339\nruntime.goexit()\n    /usr/\nlocal/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8\n    pc=0x44b4d1\ncreated by main.main\n    foo.go:5 +0x58\n\ngoroutine 1 [chan receive]:\n\nruntime.gopark(0x4739b8, 0xc420024178, 0x46fc7, 0xc, 0xc420028e17, 0x3)\n    /usr/\nlocal/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00 pc=0x42503c\n\nruntime.goparkunlock(0xc420024178, 0x46fc7, 0xc, 0x1000f010040c217, 0x3)\n    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e50\npc=0x42512e\nruntime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)\n    /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70\npc=0x4046b4\nruntime.chanrecv1(0xc420024120, 0x0)\n    /usr/local/go/src/runtime/\nchan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20 pc=0x40439b\nmain.main()\n    foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef\n\nruntime.main()\n    /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80\npc=0x424bad\nruntime.goexit()\n    /usr/local/go/src/runtime/asm_amd64.s:2337\n    +0x1 fp=0xc420053fe8 sp=0xc420053fe0 pc=0x44b4d1\n\ngoroutine 2 [force gc\n(idle)]:\n\nruntime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)\n    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738\n    pc=0x42503c\nruntime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc42000114, 0x1)\n    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768\n    pc=0x42512e\nruntime.forcegchelper()\n    /usr/local/go/src/runtime/proc.go:238 +0xcc\n    fp=0xc42003e7e0 sp=0xc42003e7a8 pc=0x424e5c\n\nruntime.goexit()\n    /usr/local/go/src/\nruntime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0 pc=0x44b4d1\n\ncreated by runtime.init.4\n    /usr/local/go/src/runtime/proc.go:227 +0x35\n\ngoroutine 3 [GC\nsweep wait]:\n\nruntime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)\n    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30\n    pc=0x42503c\nruntime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)\n    /usr/\nlocal/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60 pc=0x42512e\n\nruntime.bgsweep(0xc42001e150)\n    /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3\n    fp=0xc42003efd8 sp=0xc42003efa0 pc=0x419973\n\nruntime.goexit()\n    /usr/local/go/src/\nruntime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0 sp=0xc42003efd8 pc=0x44b4d1\n\ncreated by\n\nruntime.gcenable\n    /usr/local/go/src/runtime/mgc.go:216 +0x58",
        "container_id": "xxxxxxxx-xxxxxx",
        "container_name": "app",
        "source": "stdout",
        "ecs_cluster": "default",
        "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
        "ecs_task_definition": "firelens-example-multiline:2"
    }
}
```

The following log snippet shows how the same event appears if you run an ECS container that is not configured to concatenate multiline log messages. The `log` field contains a single line.

```
{
    "log": "panic: my panic",
    "container_id": "xxxxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
}
```

Note

If your logs go to log files instead of the standard output, we recommend specifying the `multiline.parser` and `multiline.key_content` configuration parameters in the [Tail input plugin](#) instead of the Filter.

Example logging option task definitions

The following are some example task definitions demonstrating common custom log routing options. For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

Note

The following `logConfiguration` task definition parameter shown in these examples is used to send your AWS for Fluent Bit logs to CloudWatch. AWS recommends this configuration so that you have additional information in CloudWatch to troubleshoot AWS for Fluent Bit issues.

```
"logConfiguration": {  
    "logDriver": "awslogs",  
    "options": {  
        "awslogs-group": "firelens-container",  
        "awslogs-region": "us-west-2",  
        "awslogs-create-group": "true",  
        "awslogs-stream-prefix": "firelens"  
    },  
},
```

Topics

- [Forwarding logs to CloudWatch Logs \(p. 191\)](#)
- [Forwarding logs to an Amazon Kinesis Data Firehose delivery stream \(p. 192\)](#)
- [Forwarding logs to an Amazon OpenSearch Service domain \(p. 193\)](#)
- [Parsing container logs that are serialized JSON \(p. 194\)](#)
- [Forwarding to an external Fluentd or Fluent Bit \(p. 195\)](#)

Forwarding logs to CloudWatch Logs

Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see [What Is Amazon CloudWatch Logs?](#) in the [Amazon CloudWatch Logs User Guide](#).

In the log configuration options, specify the log group name and the AWS Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group": "true"`, to set the fluentd-buffer-limit use `log-driver-buffer-limit`. You can also specify the task ID as the log stream prefix, which assists in filtering. For more information, see [Fluent Bit Plugin for CloudWatch Logs](#).

```
{  
    "family": "firelens-example-cloudwatch",  
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecs\_task\_iam\_role",  
    "containerDefinitions": [  
        {  
            "essential": true,  
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-  
bit:latest",  
            "name": "log_router",  
            "firelensConfiguration": {  
                "type": "fluentbit"  
            }  
        }  
    ]  
}
```

```

        },
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "firelens-container",
                "awslogs-region": "us-west-2",
                "awslogs-create-group": "true",
                "awslogs-stream-prefix": "firelens"
            }
        },
        "memoryReservation": 50
    },
    {
        "essential": true,
        "image": "httpd",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "cloudwatch",
                "region": "us-west-2",
                "log_group_name": "firelens-blog",
                "auto_create_group": "true",
                "log_stream_prefix": "from-fluent-bit",
                "log-driver-buffer-limit": "2097152"
            }
        },
        "memoryReservation": 100
    }
]
}

```

Forwarding logs to an Amazon Kinesis Data Firehose delivery stream

Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

The following task definition example demonstrates how to specify a log configuration that forwards logs to an Amazon Kinesis Data Firehose delivery stream. The Kinesis Data Firehose delivery stream must already exist. For more information, see [Creating an Amazon Kinesis Data Firehose Delivery Stream](#) in the [Amazon Kinesis Data Firehose Developer Guide](#).

In the log configuration options, specify the delivery stream name and the Region it exists in. For more information, see [Fluent Bit Plugin for Amazon Kinesis Firehose](#).

```
{
    "family": "firelens-example-firehose",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit"
            },
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "firelens-container",
                    "awslogs-region": "us-west-2",
                    "awslogs-create-group": "true",
                    "awslogs-stream-prefix": "firelens"
                }
            }
        }
    ]
}
```

```

        },
        "memoryReservation": 50
    },
    {
        "essential": true,
        "image": "httpd",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "firehose",
                "region": "us-west-2",
                "delivery_stream": "my-stream"
            }
        },
        "memoryReservation": 100
    }
]
}

```

Forwarding logs to an Amazon OpenSearch Service domain

Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

The following task definition example demonstrates how to specify a log configuration that forwards logs to an Amazon OpenSearch Service domain. The Amazon OpenSearch Service domain must already exist. For more information, see [What is Amazon OpenSearch Service](#) in the *Amazon OpenSearch Service Developer Guide*.

In the log configuration options, specify the log options required for OpenSearch Service integration. For more information, see [Fluent Bit for Amazon OpenSearch Service](#).

```
{
    "family": "firelens-example-opensearch",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-
bit:stable",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit"
            },
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "firelens-container",
                    "awslogs-region": "us-west-2",
                    "awslogs-create-group": "true",
                    "awslogs-stream-prefix": "firelens"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "httpd",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "es",

```

```

        "Host": "vpc-fake-domain-ke7thhz07jawzhmz6mb7ite7y.us-west-2.es.amazonaws.com",
        "Port": "443",
        "Index": "my_index",
        "Type": "my_type",
        "AWS_Auth": "On",
        "AWS_Region": "us-west-2",
        "tls": "On"
    }
},
"memoryReservation": 100
]
}
}

```

Parsing container logs that are serialized JSON

Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

Beginning with AWS for Fluent Bit version 1.3, there's a JSON parser that's included in the AWS for Fluent Bit image. The following example shows how to reference the JSON parser in the FireLens configuration of your task definition.

```

"firelensConfiguration": {
    "type": "fluentbit",
    "options": {
        "config-file-type": "file",
        "config-file-value": "/fluent-bit/configs/parse-json.conf"
    }
},

```

The Fluent Bit config file parses any logs that are in JSON (for example, if the logs at your destination looked like the following without JSON parsing).

```

{
    "source": "stdout",
    "log": "{\"requestID\": \"b5d716fca19a4252ad90e7b8ec7cc8d2\", \"requestInfo\":
{\\"ipAddress\\": \"204.16.5.19\", \\"path\\": \"/activate\", \\"user\\": \"TheDoctor\"}}",
    "container_id": "e54cccfac2b87417f71877907f67879068420042828067ae0867e60a63529d35",
    "container_name": "/ecs-demo-6-container2-a4eafbb3d4c7f1e16e00",
    "ecs_cluster": "mycluster",
    "ecs_task_arn": "arn:aws:ecs:us-east-2:01234567891011:task/
mycluster/3de392df-6bfa-470b-97ed-aa6f482cd7a6",
    "ecs_task_definition": "demo:7"
    "ec2_instance_id": "i-06bc83dbc2ac2fdf8"
}

```

With the JSON parsing, the log looks like the following.

```

{
    "source": "stdout",
    "container_id": "e54cccfac2b87417f71877907f67879068420042828067ae0867e60a63529d35",
    "container_name": "/ecs-demo-6-container2-a4eafbb3d4c7f1e16e00",
    "ecs_cluster": "mycluster",
    "ecs_task_arn": "arn:aws:ecs:us-east-2:01234567891011:task/
mycluster/3de392df-6bfa-470b-97ed-aa6f482cd7a6",
    "ecs_task_definition": "demo:7"
    "ec2_instance_id": "i-06bc83dbc2ac2fdf8",
    "requestID": "b5d716fca19a4252ad90e7b8ec7cc8d2",
    "requestInfo": {

```

```
        "ipAddress": "204.16.5.19",
        "path": "/activate",
        "user": "TheDoctor"
    }
}
```

The serialized JSON is expanded into top level fields in the final JSON output. For more information about JSON parsing, see [Parser](#) in the Fluent Bit documentation.

Forwarding to an external Fluentd or Fluent Bit

Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

The following task definition example demonstrates how to specify a log configuration that forwards logs to an external Fluentd or Fluent Bit host. Specify the host and port for your environment.

```
{
    "family": "firelens-example-forward",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit"
            },
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "firelens-container",
                    "awslogs-region": "us-west-2",
                    "awslogs-create-group": "true",
                    "awslogs-stream-prefix": "firelens"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "httpd",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "forward",
                    "Host": "Fluentdhost",
                    "Port": "24224"
                }
            },
            "memoryReservation": 100
        }
    ]
}
```

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 task definition. This provides a way to reference container images that exist in private registries outside of AWS that require authentication in your task

definitions. This feature is supported by tasks hosted on Fargate, Amazon EC2 instances, and external instances using Amazon ECS Anywhere.

Important

If your task definition references an image that's stored in Amazon ECR, this topic doesn't apply. For more information, see [Using Amazon ECR Images with Amazon ECS](#) in the *Amazon Elastic Container Registry User Guide*.

For tasks hosted on Amazon EC2 instances, this feature requires version 1.19.0 or later of the container agent. However, we recommend using the latest container agent version. For information about how to check your agent version and update to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

For tasks hosted on Fargate, this feature requires platform version 1.2.0 or later. For information, see [AWS Fargate platform versions \(p. 77\)](#).

Within your container definition, specify the `repositoryCredentials` object with the details of the secret that you created. The secret you reference can be from a different AWS Region or a different account than the task using it.

Note

When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same AWS Region as the task that you're launching then you can use either the full ARN or name of the secret. If the secret exists in a different account, the full ARN of the secret must be specified.

When using the AWS Management Console, the full ARN of the secret must be specified always.

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

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

Note

Another method of enabling private registry authentication uses Amazon ECS container agent environment variables to authenticate to private registries. This method is only supported for tasks hosted on Amazon EC2 instances. For more information, see [Private registry authentication for container instances \(p. 371\)](#).

Required IAM permissions for private registry authentication

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 \(p. 626\)](#).

To provide access to the secrets that you create, 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 Amazon Resource Name (ARN) for your custom key must be added as a resource.

The following is an example inline policy that adds the permissions.

```
{
```

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

Using 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**, enter 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 instructions on 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 that you entered as a new secret in Secrets Manager.

Register a task definition and under **Private registry**, turn on **Private registry authentication**. Then, in **Secrets Manager ARN or name**, enter the Amazon Resource Name (ARN) of the secret. You must use a For more information, see [Required IAM permissions for private registry authentication \(p. 196\)](#). For more information, see [the section called “Creating a task definition using the console” \(p. 125\)](#).

Passing environment variables to a container

Important

We recommend storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

Environment variables specified in the task definition are readable by all users and roles that are allowed the `DescribeTaskDefinition` action for the task definition.

Environment variable files are objects in Amazon S3 and all Amazon S3 security considerations apply. See the below section [the section called "Required IAM permissions" \(p. 199\)](#).

You can pass environment variables to your containers in the following ways:

- Individually using the `environment` container definition parameter. This maps to the `--env` option to [`docker run`](#).
- In bulk, using the `environmentFiles` container definition parameter to list one or more files that contain the environment variables. The file must be hosted in Amazon S3. This maps to the `--env-file` option to [`docker run`](#).

By specifying environment variables in a file, you can bulk inject environment variables. Within your container definition, specify the `environmentFiles` object with a list of Amazon S3 buckets containing your environment variable files. The files must use an `.env` file extension and there is a limit of ten files to a task definition.

We don't enforce a size limit on the environment variables, but a large environment variables file might fill up the disk space. Each task that uses an environment variables file causes a copy of the file to be downloaded to disk. We remove the file as part of the task cleanup.

For information about the supported environment variables, see [Advanced container definition parameters- Environment](#).

The following is a snippet of a task definition showing how to specify individual environment variables.

```
{
  "family": "",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      ...
      "environment": [
        {
          "name": "variable",
          "value": "value"
        }
      ],
      ...
    },
    ...
  ]
}
```

The following is a snippet of a task definition showing how to specify an environment variable file.

```
{
  "family": "",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      ...
      "environmentFiles": [
        {
          "value": "arn:aws:s3:::s3_bucket_name/envfile_object_name.env",
          "type": "s3"
        }
      ]
    }
  ]
}
```

```
        ],
        ...
    ],
    ...
}
```

Considerations for specifying environment variable files

Consider the following when specifying an environment variable file in a container definition.

- For Amazon ECS tasks on Amazon EC2, your container instances require that the container agent is version 1.39.0 or later to use this feature. For information about how to check your agent version and update to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).
- For Amazon ECS tasks on AWS Fargate, your tasks must use platform version 1.4.0 or later (Linux) to use this feature. For more information, see [AWS Fargate platform versions \(p. 77\)](#).

Verify that the variable is supported for the operating system platform. For more information, see [the section called “Container definitions” \(p. 804\)](#) and [the section called “Other task definition parameters” \(p. 841\)](#).

- The file must use the .env file extension and UTF-8 encoding.
- Each line in an environment file must contain an environment variable in VARIABLE=VALUE format. Spaces or quotation marks are included as part of the values. Lines beginning with # are treated as comments and are ignored. For more information about the environment variable file syntax, see [Declare default environment variables in file](#).

The following is the appropriate syntax.

```
#This is a comment and will be ignored
VARIABLE=VALUE
ENVIRONMENT=PRODUCTION
```

- If there are environment variables specified using the environment parameter in a container definition, they take precedence over the variables contained within an environment file.
- If multiple environment files are specified and they contain the same variable, they're processed in order of entry. This means that the first value of the variable is used and subsequent values of duplicate variables are ignored. We recommend that you use unique variable names.
- If an environment file is specified as a container override, it's used. Moreover, any other environment files that are specified in the container definition is ignored.

Required IAM permissions

The Amazon ECS task execution role is required to use this feature. This allows the container agent to pull the environment variable file from Amazon S3. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

To provide access to the Amazon S3 objects that you create, manually add the following permissions as an inline policy to the task execution role. Use the Resource parameter to scope the permission to the Amazon S3 buckets that contain the environment variable files. For more information, see [Adding and Removing IAM Policies](#).

- s3:GetObject
- s3:GetBucketLocation

In the following example, the permissions are added to the inline policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket/folder_name/env_file_name"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        }  
    ]  
}
```

Passing sensitive data to a container

You can safely pass sensitive data, such as credentials to a database, into your container. To start, first store the sensitive data as a secret in AWS Secrets Manager or as a parameter in AWS Systems Manager Parameter Store. Then, use one of the following ways to expose the secret to the container.

Topics

- [Retrieve secrets programmatically through your application \(p. 200\)](#)
- [Retrieve secrets through environment variables \(p. 203\)](#)
- [Retrieve secrets for logging configuration \(p. 212\)](#)

Retrieve secrets programmatically through your application

Instead of hardcoding sensitive information in plain text in your application, you can use Secrets Manager or Systems Manager Parameter Store to store the sensitive data.

We recommend this method of retrieving sensitive data because if the Secrets Manager secret is subsequently updated, the application automatically retrieves the latest version of the secret.

Topics

- [Using Secrets Manager \(p. 200\)](#)
- [Using AWS Systems Manager Parameter Store \(p. 201\)](#)

Using Secrets Manager

Use Secrets Manager to protect sensitive data and rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle.

After you create a Secrets Manager secret, update your application code to retrieve the secret.

Considerations

Review the following considerations before securing sensitive data in Secrets Manager.

- Only secrets that store text data, which are secrets created with the `SecretString` parameter of the [CreateSecret](#) API, are supported. Secrets that store binary data, which are secrets created with the `SecretBinary` parameter of the [CreateSecret](#) API are not supported.
- Use interface VPC endpoints to enhance security controls. You must create the interface VPC endpoints for Secrets Manager. For information about the VPC endpoint, see [Create VPC endpoints](#) in the *AWS Secrets Manager User Guide*.
- The VPC your task uses must use DNS resolution.

Required IAM permissions

To use this feature, you must have the Amazon ECS task role and reference it in your task definition. For more information, see [Task IAM role \(p. 63\)](#).

To provide access to the Secrets Manager secrets that you create, manually add the following permission to the task execution role. For information about how to manage permissions, see [Adding and Removing IAM identity permissions](#) in the *IAM User Guide*.

- `secretsmanager:GetSecretValue`– Required if you are referencing a Secrets Manager secret. Adds the permission to retrieve the secret from Secrets Manager.

The following example policy adds the required permissions.

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

Create the Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For information about how to create secrets, see [Create an AWS Secrets Manager secret](#) in the *AWS Secrets Manager User Guide*.

Update your application to programmatically retrieve Secrets Manager secrets

You can retrieve secrets with a call to the Secrets Manager APIs directly from your application. For information about how update your code to request the secret, see [Retrieve secrets from AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

To retrieve the sensitive data stored in the AWS Secrets Manager, see [Code examples for AWS Secrets Manager using AWS SDKs](#) in the *AWS SDK Code Examples Code Library*.

Using AWS Systems Manager Parameter Store

Systems Manager Parameter Store provides secure storage and management of secrets. You can store data such as passwords, database strings, Amazon Elastic Compute Cloud (Amazon EC2) instance IDs and

Amazon Machine Image (AMI) IDs, and license codes as parameter values. You can store values as plain text or encrypted data.

After you store the secret using Systems Manager Parameter Store, update your application code to retrieve the secret.

Considerations

Review the following considerations before securing sensitive data in Systems Manager Parameter Store.

- Only secrets that store text data are supported. Secrets that store binary data are not supported.
- Use interface VPC endpoints to enhance security controls.
- The VPC your task uses must use DNS resolution.

Required IAM permissions

To use this feature, you must have the Amazon ECS task role and reference it in your task definition. This allows the container agent to pull the necessary Systems Manager resources. For more information, see [Task IAM role \(p. 631\)](#).

Important

For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

To provide access to the Systems Manager Parameter Store parameters that you create, manually add the following permissions as a policy to the task execution role. For information about how to manage permissions, see [Adding and Removing IAM identity permissions](#) in the *IAM User Guide*.

- `ssm:GetParameters` — Required if you are referencing a Systems Manager Parameter Store parameter in a task definition. Adds the permission to retrieve Systems Manager parameters.
- `secretsmanager:GetSecretValue` — Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition. Adds the permission to retrieve the secret from Secrets Manager.
- `kms:Decrypt` — Required only if your secret uses a customer managed key and not the default key. The ARN for your custom key should be added as a resource. Adds the permission to decrypt the customer managed key .

The following example policy adds the required permissions:

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

```
    ]  
}
```

Create the secret

You can use the Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see [Create a Systems Manager parameter \(console\)](#) or [Create a Systems Manager parameter \(AWS CLI\)](#) in the *AWS Systems Manager User Guide*.

Update your application to programmatically retrieve Systems Manager Parameter Store secrets

To retrieve the sensitive data stored in the Systems Manager Parameter Store parameter, see [Code examples for Systems Manager using AWS SDKs](#) in the *AWS SDK Code Examples Code Library*.

Retrieve secrets through environment variables

Instead of hardcoding sensitive information in plain text in your application, you can use Secrets Manager or AWS Systems Manager Parameter Store to store the sensitive data. Then, you can create an environment variable in the container definition and enter the ARN of the Secrets Manager or AWS Systems Manager secret as the value. This allows your container to retrieve sensitive data at runtime.

Topics

- [Using Secrets Manager \(p. 203\)](#)
- [Using AWS Systems Manager \(p. 209\)](#)

Using Secrets Manager

When you inject a secret as an environment variable, you can specify the full contents of a secret, a specific JSON key within a secret, or a specific version of a secret to inject. This helps you control the sensitive data exposed to your container. For more information about secret versioning, see [Key Terms and Concepts for AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

Considerations

The following should be considered when using an environment variable to inject an Secrets Manager secret into a container.

- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Amazon ECS tasks on AWS Fargate, the following should be considered:
 - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version 1.3.0 or later. For information, see [AWS Fargate platform versions \(p. 77\)](#).
 - To inject a specific JSON key or version of a secret as an environment variable or in a log configuration, you must use platform version 1.4.0 or later (Linux) or 1.0.0 (Windows). For information, see [AWS Fargate platform versions \(p. 77\)](#).
- For Amazon ECS tasks on EC2, the following should be considered:
 - To inject a secret using a specific JSON key or version of a secret, your container instance must have version 1.37.0 or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version 1.22.0 or later of the container agent.

- Use interface VPC endpoints to enhance security controls. You must create the interface VPC endpoints for Secrets Manager. For information about the VPC endpoint, see [Create VPC endpoints](#) in the *AWS Secrets Manager User Guide*.
- For Windows tasks that are configured to use the awslogs logging driver, you must also set the ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE environment variable on your container instance. This can be done with User Data with the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE", $TRUE,
"Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers '["json-
file", "awslogs"]'
</powershell>
```

IAM permissions

To use this feature, you must have the Amazon ECS task execution role and reference it in your task definition. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

To provide access to the Secrets Manager 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—Required if you are referencing a Secrets Manager secret. Adds the permission to retrieve the secret from Secrets Manager.
- kms:Decrypt—Required only if your secret uses a customer managed key and not the default key. The ARN for your customer managed key should be added as a resource. Adds the permission to decrypt the customer managed key.

The following example policy adds the required permissions:

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

Create the AWS Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For more information, see [Create an AWS Secrets Manager secret](#) in the *AWS Secrets Manager User Guide*.

Add the environment variable to the container definition

Within your container definition, you can specify the following:

- The secrets object containing the name of the environment variable to set in the container
- The Amazon Resource Name (ARN) of the Secrets Manager secret
- Additional parameters that contain the sensitive data to present to the container

The following example shows the full syntax that must be specified for the Secrets Manager secret.

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-stage:version-id
```

The following section describes the additional parameters. These parameters are optional, but if you do not use them, you must include the colons : to use the default values. Examples are provided below for more context.

json-key

Specifies the name of the key in a key-value pair with the value that you want to set as the environment variable value. Only values in JSON format are supported. If you do not specify a JSON key, then the full contents of the secret is used.

version-stage

Specifies the staging label of the version of a secret that you want to use. If a version staging label is specified, you cannot specify a version ID. If no version stage is specified, the default behavior is to retrieve the secret with the AWSCURRENT staging label.

Staging labels are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has one or more staging labels and an ID. For more information, see [Key Terms and Concepts for AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

version-id

Specifies the unique identifier of the version of a secret that you want to use. If a version ID is specified, you cannot specify a version staging label. If no version ID is specified, the default behavior is to retrieve the secret with the AWSCURRENT staging label.

Version IDs are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has an ID. For more information, see [Key Terms and Concepts for AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

Example container definitions

The following examples show ways in which you can reference Secrets Manager secrets in your container definitions.

Example referencing a full secret

The following is a snippet of a task definition showing the format when referencing the full text of a Secrets Manager secret.

```
{  
    "containerDefinitions": [  
        {  
            "secrets": [  
                {  
                    "name": "environment_variable_name",  
                    "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"  
                }  
            ]  
        }  
    ]  
}
```

To access the value of this secret from within the container you would need to call the `$environment_variable_name`.

Example referencing a specific key within a secret

The following shows an example output from a [get-secret-value](#) command that displays the contents of a secret along with the version staging label and version ID associated with it.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981ddEXAMPLE",
    "SecretString": "{\"username1\":\"password1\", \"username2\":\"password2\",
    \"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

Reference a specific key from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
    "containerDefinitions": [
        "secrets": [
            {
                "name": "environment_variable_name",
                "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
            }
        ]
    ]
}
```

Example referencing a specific secret version

The following shows an example output from a [describe-secret](#) command that displays the unencrypted contents of a secret along with the metadata for all versions of the secret.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981ddEXAMPLE": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE": [
            "AWSPREVIOUS"
        ]
    }
}
```

Reference a specific version staging label from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
```

```

    "containerDefinitions": [
        "secrets": [
            {
                "name": "environment_variable_name",
                "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::AWSVIOUS:"
            }
        ]
    }
}

```

Reference a specific version ID from the previous output in a container definition by specifying the key name at the end of the ARN.

```

{
    "containerDefinitions": [
        "secrets": [
            {
                "name": "environment_variable_name",
                "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
            }
        ]
    }
}

```

Example referencing a specific key and version staging label of a secret

The following shows how to reference both a specific key within a secret and a specific version staging label.

```

{
    "containerDefinitions": [
        "secrets": [
            {
                "name": "environment_variable_name",
                "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1:AWSVIOUS:"
            }
        ]
    }
}

```

To specify a specific key and version ID, use the following syntax.

```

{
    "containerDefinitions": [
        "secrets": [
            {
                "name": "environment_variable_name",
                "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
            }
        ]
    }
}

```

Create a task definition that references sensitive data

You can use the Amazon ECS console to create a task definition that references a Secrets Manager secret.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition**, **Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.

5. For each container to define in your task definition, complete the following steps.
 - a. For **Name**, enter a name for the container.
 - b. For **Image URI**, enter the image to use to start a container. Images in the Amazon ECR Public Gallery registry may be specified using the Amazon ECR Public registry name only. For example, if `public.ecr.aws/ecs/amazon-ecs-agent:latest` is specified, the Amazon Linux container hosted on Amazon ECR Public Gallery is used. For all other repositories, specify the repository using either the `repository-url/image:tag` or `repository-url/image@digest` formats.
 - c. For **Essential container**, if your task definition has two or more containers defined, you may specify whether the container should be considered essential. If a container is marked as *essential*, if that container stops then the task is stopped. Each task definition must contain at least one essential container.
 - d. A port mapping allows the container to access ports on the host to send or receive traffic. Under **Port mappings**, do one of the following:
 - When you use the **awsvpc** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.
 - When you use the **bridge** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container. You select the **bridge** network mode on the next page. After you select it, choose **Previous**, and then for **Host port**, specify the port number on the container instance to reserve for your container.
- Choose **Add more port mappings** to specify additional container port mappings.
- e. For sensitive data to inject as environment variables, under **Environment**, for **Environment variables**, complete the following fields:
 - i. For **Key**, enter the name of the environment variable to set in the container. This corresponds to the name field in the secrets section of a container definition.
 - ii. For **Value type**, choose **ValueFrom**. For **Add value**, enter the ARN of the Secrets Manager secret that contains the data to present to your container as an environment variable.
- f. (Optional) Choose **Add more containers** to add additional containers to the task definition. Choose **Next** once all containers have been defined.
6. For **App environment**, **Task size**, and **Container size**, fill out the remaining required fields and any optional fields.
7. (Optional) Expand the **Task roles, network mode** section to specify the following:
 - For **Task role**, choose the IAM role to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS APIs.
8. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate as well as add a data volume configuration for the task.
 - To expand the available ephemeral storage beyond the default value of 20 GiB for your Fargate tasks, for **Amount**, enter a value up to 200 GiB.
9. For sensitive data referenced in the log configuration for a container, under **Use log collection**, for **Log configuration**, complete the following configuration:
 - a. Select the log option, and then under **Key**, choose **Add**.
 - b. For **Key**, enter the name of the log configuration option to set.
 - c. For **Value**, enter the full ARN of the Secrets Manager secret that contains the data to present to your log configuration as a log option.
 - d. For **Value type**, choose **ValueFrom**.
10. Choose **Next** to review the task definition.

11. On the **Review and create** page, review each task definition section. Choose **Edit** to make changes. After the task definition is complete, choose **Create** to register the task definition.

Using AWS Systems Manager

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in AWS Systems Manager Parameter Store parameters and then referencing them in your container definition.

Considerations

The following should be considered when using an environment variable to inject an AWS Systems Manager secret into a container.

- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Amazon ECS tasks on AWS Fargate, the following should be considered:
 - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version 1.3.0 or later. For information, see [AWS Fargate platform versions \(p. 77\)](#).
 - To inject a specific JSON key or version of a secret as an environment variable or in a log configuration, you must use platform version 1.4.0 or later (Linux) or 1.0.0 (Windows). For information, see [AWS Fargate platform versions \(p. 77\)](#).
- For Amazon ECS tasks on EC2, the following should be considered:
 - To inject a secret using a specific JSON key or version of a secret, your container instance must have version 1.37.0 or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version 1.22.0 or later of the container agent.

- Use interface VPC endpoints to enhance security controls. You must create the interface VPC endpoints for Secrets Manager. For information about the VPC endpoint, see [Create VPC endpoints](#) in the [AWS Secrets Manager User Guide](#).
- For Windows tasks that are configured to use the awslogs logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with User Data using the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE", $TRUE,
"Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers '[{"json-
file", "awslogs"}]'
</powershell>
```

IAM permissions

To use this feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager resources. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Important

For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add

it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions to the task execution role. For information about how to manage permissions, see [Adding and Removing IAM identity permissions](#) in the *IAM User Guide*.

- `ssm:GetParameters` — Required if you are referencing a Systems Manager Parameter Store parameter in a task definition. Adds the permission to retrieve Systems Manager parameters.
- `secretsmanager:GetSecretValue` — Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition. Adds the permission to retrieve the secret from Secrets Manager.
- `kms:Decrypt` — Required only if your secret uses a customer managed key and not the default key. The ARN for your custom key should be added as a resource. Adds the permission to decrypt the customer managed key.

The following example policy adds the required permissions:

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

Create the AWS Systems Manager parameter

You can use the Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see [Create a Systems Manager parameter \(console\)](#) or [Create a Systems Manager parameter \(AWS CLI\)](#) in the *AWS Systems Manager User Guide*.

Add the environment variable to the container definition

Within your container definition, specify secrets with the name of the environment variable to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container. For more information, see [secrets \(p. 816\)](#).

The following is a snippet of a task definition showing the format when referencing a Systems Manager Parameter Store parameter. 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 parameter. If the parameter exists in a different Region, then specify the full ARN.

```
{  
    "containerDefinitions": [{  
        "secrets": [{  
            "name": "environment_variable_name",  
            "value": "value"  
        }]  
    }]
```

```
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"  
    }]  
}  
}
```

Create a task definition that references sensitive data

You can use the Amazon ECS console to create a task definition that references a Systems Manager Parameter Store parameter.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition, Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. For each container to define in your task definition, complete the following steps.
 - a. For **Name**, enter a name for the container.
 - b. For **Image URI**, enter the image to use to start a container. Images in the Amazon ECR Public Gallery registry may be specified using the Amazon ECR Public registry name only. For example, if `public.ecr.aws/ecs/amazon-ecs-agent:latest` is specified, the Amazon Linux container hosted on Amazon ECR Public Gallery is used. For all other repositories, specify the repository using either the `repository-url/image:tag` or `repository-url/image@digest` formats.
 - c. For **Essential container**, if your task definition has two or more containers defined, you may specify whether the container should be considered essential. If a container is marked as *essential*, if that container stops then the task is stopped. Each task definition must contain at least one essential container.
 - d. A port mapping allows the container to access ports on the host to send or receive traffic. Under **Port mappings**, do one of the following:
 - When you use the **awsvpc** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.
 - When you use the **bridge** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container. You select the **bridge** network mode on the next page. After you select it, choose **Previous**, and then for **Host port**, specify the port number on the container instance to reserve for your container.

Choose **Add more port mappings** to specify additional container port mappings.

- e. For sensitive data to inject as environment variables, under **Environment**, for **Environment variables**, complete the following fields:
 - i. For **Key**, enter the name of the environment variable to set in the container. This corresponds to the name field in the secrets section of a container definition.
 - ii. For **Value type**, choose **ValueFrom**. For **Value**, enter the name or full ARN of the AWS Systems Manager Parameter Store parameter that contains the data to present to your log configuration as a log option.

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 the name of the secret. If the parameter exists in a different Region, then specify the full ARN.

- f. (Optional) Choose **Add more containers** to add additional containers to the task definition. Choose **Next** once all containers have been defined.
6. For **App environment**, **Task size**, and **Container size**, fill out the remaining required fields and any optional fields.

7. (Optional) Expand the **Task roles, network mode** section to specify the following:
 - For **Task role**, choose the IAM role to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS APIs.
 8. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate as well as add a data volume configuration for the task.
 - To expand the available ephemeral storage beyond the default value of 20 GiB for your Fargate tasks, for **Amount**, enter a value up to 200 GiB.
 9. For sensitive data referenced in the log configuration for a container, under **Use log collection**, for **Log configuration**, complete the following configuration:
 - a. Select the log option, and then under **Key**, choose **Add**.
 - b. For **Key**, enter the name of the log configuration option to set.
 - c. For **Value**, enter the name or full ARN of the AWS Systems Manager Parameter Store parameter that contains the data to present to your log configuration as a log option.
- 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 the name of the secret. If the parameter exists in a different Region, then specify the full ARN.
- d. For **Value type**, choose **ValueFrom**.
 10. Choose **Next** to review the task definition.
 11. On the **Review and create** page, review each task definition section. Choose **Edit** to make changes. After the task definition is complete, choose **Create** to register the task definition.

Retrieve secrets for logging configuration

Using Secrets Manager

Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of the Secrets Manager secret containing the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing an Secrets Manager secret.

```
{  
  "containerDefinitions": [  
    {  
      "logConfiguration": [  
        {  
          "logDriver": "splunk",  
          "options": {  
            "splunk-url": "https://your_splunk_instance:8088"  
          },  
          "secretOptions": [  
            {  
              "name": "splunk-token",  
              "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-  
AbCdEf"  
            }  
          ]  
        ]  
      }  
    ]  
}
```

Using AWS Systems Manager

You can inject sensitive data in a log configuration. Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set

in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

Important

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 parameter. If the parameter exists in a different Region, then specify the full ARN.

The following is a snippet of a task definition showing the format when referencing a Systems Manager Parameter Store parameter.

```
{  
  "containerDefinitions": [{  
    "logConfiguration": [{  
      "logDriver": "fluenta",  
      "options": {  
        "tag": "fluentd demo"  
      },  
      "secretOptions": [{  
        "name": "fluentd-address",  
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:/parameter_name"  
      }]  
    }]  
  }]
```

Example task definitions

This section provides some JSON task definition examples and snippets that you can use to start creating your own task definitions.

You can copy the examples, and then paste them when you use the **Configure via JSON** option in the console. Make sure to customize the examples, such as using your account ID. You can include the snippets in your task definition JSON. For more information, see [Creating a task definition using the console \(p. 125\)](#) and [Task definition parameters \(p. 799\)](#).

For more task definition examples, see [AWS Sample Task Definitions](#) on GitHub.

Topics

- [Example: Webserver \(p. 213\)](#)
- [Example: splunk log driver \(p. 215\)](#)
- [Example: fluentd log driver \(p. 215\)](#)
- [Example: gelf log driver \(p. 216\)](#)
- [Example: Amazon ECR image and task definition IAM role \(p. 216\)](#)
- [Example: Entrypoint with command \(p. 217\)](#)
- [Example: Container dependency \(p. 217\)](#)
- [Windows sample task definitions \(p. 218\)](#)

Example: Webserver

The following is an example task definition using the Linux containers on 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",
    "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
    "requiresCompatibilities": [
        "FARGATE"
    ]
}
```

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

```
{
    "containerDefinitions": [
        {
            "command": [
                "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<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>'; C:\\ServiceMonitor.exe w3svc"
            ],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
        }
    ]
}
```

```

    "essential": true,
    "cpu": 2048,
    "memory": 4096,
    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
    "name": "sample_windows_app",
    "portMappings": [
        {
            "hostPort": 80,
            "containerPort": 80,
            "protocol": "tcp"
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {
        "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
    },
    "requiresCompatibilities": [
        "FARGATE"
    ]
}

```

Example: splunk log driver

The following snippet demonstrates how to use the splunk log driver in a task definition that sends the logs to a remote service. The Splunk token parameter is specified as a secret option because it can be treated as sensitive data. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

```

"containerDefinitions": [],
"logConfiguration": {
    "logDriver": "splunk",
    "options": {
        "splunk-url": "https://cloud.splunk.com:8080",
        "tag": "tag_name",
    },
    "secretOptions": [
        {
            "name": "splunk-token",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:splunk-token-KnrBkD"
        }
    ],
}

```

Example: fluentd log driver

The following snippet demonstrates how to use the fluentd log driver in a task definition that sends the logs to a remote service. The fluentd-address value is specified as a secret option as it may be treated as sensitive data. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

```

"containerDefinitions": [],
"logConfiguration": {
    "logDriver": "fluentd",
    "options": {
        "tag": "fluentd demo"
    },
    "secretOptions": [
        {
            "name": "fluentd-address",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-KnrBkD"
        }
    ],
}

```

```

        }],
      },
      "entryPoint": [],
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        },
        {
          "hostPort": 24224,
          "protocol": "tcp",
          "containerPort": 24224
        }
      ],
    ],
  ]
}

```

Example: gelf log driver

The following snippet demonstrates how to use the gelf log driver in a task definition that sends the logs to a remote host running Logstash that takes Gelf logs as an input. For more information, see [logConfiguration \(p. 821\)](#).

```

"containerDefinitions": [
  {
    "logConfiguration": {
      "logDriver": "gelf",
      "options": [
        "gelf-address": "udp://logstash-service-address:5000",
        "tag": "gelf task demo"
      ]
    },
    "entryPoint": [],
    "portMappings": [
      {
        "hostPort": 5000,
        "protocol": "udp",
        "containerPort": 5000
      },
      {
        "hostPort": 5000,
        "protocol": "tcp",
        "containerPort": 5000
      }
    ]
  ]
}

```

Example: Amazon ECR image and task definition IAM role

The following snippet uses an Amazon ECR image called aws-nodejs-sample with the v1 tag from the 123456789012.dkr.ecr.us-west-2.amazonaws.com registry. The container in this task inherits IAM permissions from the arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole role. For more information, see [Task IAM role \(p. 631\)](#).

```

{
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-sample:v1",
      "memory": 200,
      "cpu": 10,
      "essential": true
    }
  ]
}

```

```
        ],
        "family": "example_task_3",
        "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
    }
```

Example: Entrypoint with command

The following snippet demonstrates the syntax for a Docker container that uses an entry point and a command argument. This container pings google.com four times and then exits.

```
{
    "containerDefinitions": [
        {
            "memory": 32,
            "essential": true,
            "entryPoint": [
                "ping"
            ],
            "name": "alpine_ping",
            "readonlyRootFilesystem": true,
            "image": "alpine:3.4",
            "command": [
                "-c",
                "4",
                "example.com"
            ],
            "cpu": 16
        }
    ],
    "family": "example_task_2"
}
```

Example: Container dependency

This snippet demonstrates the syntax for a task definition with multiple containers where container dependency is specified. In the following task definition, the envoy container must reach a healthy status, determined by the required container health check parameters, before the app container will start. For more information, see [Container dependency \(p. 830\)](#).

```
{
    "family": "appmesh-gateway",
    "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
    "proxyConfiguration": {
        "type": "APPMESH",
        "containerName": "envoy",
        "properties": [
            {
                "name": "IgnoredUID",
                "value": "1337"
            },
            {
                "name": "ProxyIngressPort",
                "value": "15000"
            },
            {
                "name": "ProxyEgressPort",
                "value": "15001"
            }
        ]
    }
}
```

```

        },
        {
            "name": "AppPorts",
            "value": "9080"
        },
        {
            "name": "EgressIgnoredIPs",
            "value": "169.254.170.2,169.254.169.254"
        }
    ]
},
"containerDefinitions": [
{
    "name": "app",
    "image": "application_image",
    "portMappings": [
        {
            "containerPort": 9080,
            "hostPort": 9080,
            "protocol": "tcp"
        }
    ],
    "essential": true,
    "dependsOn": [
        {
            "containerName": "envoy",
            "condition": "HEALTHY"
        }
    ],
    {
        "name": "envoy",
        "image": "840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-
prod",
        "essential": true,
        "environment": [
            {
                "name": "APPMESH_VIRTUAL_NODE_NAME",
                "value": "mesh/meshName/virtualNode/virtualNodeName"
            },
            {
                "name": "ENVOY_LOG_LEVEL",
                "value": "info"
            }
        ],
        "healthCheck": {
            "command": [
                "CMD-SHELL",
                "echo hello"
            ],
            "interval": 5,
            "timeout": 2,
            "retries": 3
        }
    }
],
"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
"networkMode": "awsvpc"
}
]
}

```

Windows sample task definitions

The following is a sample task definition to help you get started with Windows containers on Amazon ECS.

Example Amazon ECS Console Sample Application for Windows

The following task definition is the Amazon ECS console sample application that is produced in the first-run wizard for Amazon ECS; it has been ported to use the `microsoft/iis` Windows container image.

```
{  
    "family": "windows-simple-iis",  
    "containerDefinitions": [  
        {  
            "name": "windows_sample_app",  
            "image": "mcr.microsoft.com/windows/servercore/iis",  
            "cpu": 1024,  
            "entryPoint": ["powershell", "-Command"],  
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<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>'; C:\\ServiceMonitor.exe w3svc"],  
            "portMappings": [  
                {  
                    "protocol": "tcp",  
                    "containerPort": 80  
                }  
            ],  
            "memory": 1024,  
            "essential": true  
        }  
    ],  
    "networkMode": "awsvpc",  
    "memory": "1024",  
    "cpu": "1024"  
}
```

Amazon ECS clusters

An Amazon ECS cluster is a logical grouping of tasks or services. In addition to tasks and services, a cluster consists of the following resources:

- The infrastructure capacity which can be any of the following:
 - Amazon EC2 instances in the AWS cloud
 - Serverless (AWS Fargate (Fargate)) in the AWS cloud
 - On-premises virtual machines (VM) or servers
- The network (VPC and subnet) where your tasks and services run

When you use Amazon EC2 instances for the capacity, the subnet can be in Availability Zones, Local Zones, Wavelength Zones or AWS Outposts.

- An optional namespace

The namespace is used for service-to-service communication with Service Connect.

- A monitoring option

CloudWatch Container Insights comes at an additional cost and is a fully managed service. It automatically collects, aggregates, and summarizes Amazon ECS metrics and logs.

Topics

- [Amazon ECS capacity providers \(p. 220\)](#)
- [Amazon ECS cluster auto scaling \(p. 226\)](#)
- [Cluster management \(p. 235\)](#)

Amazon ECS capacity providers

Amazon ECS capacity providers manage the scaling of infrastructure for tasks in your clusters. Each cluster can have one or more capacity providers and an optional capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.

Capacity providers are available for tasks that run on Fargate or on Amazon EC2 instances. You cannot use capacity providers for tasks that run on external container instances (Amazon ECS Anywhere).

Capacity provider concepts

Capacity providers consist of the following components.

Capacity provider

A *capacity provider* defines the cluster capacity that Amazon ECS scales up and down of the infrastructure you specify. You must first associate the capacity provider with a cluster before you use the capacity provider.

You use a capacity provider in a capacity provider strategy to determine the infrastructure that a task runs on. Every task must have a capacity provider strategy, a launch type, or use the default capacity provider strategy that's associated with the selected cluster. You must reference the

capacity provider strategy and not the capacity provider. If a task uses a launch type, the capacity it uses isn't counted by any capacity providers in the cluster.

For AWS Fargate, the capacity providers are a FARGATE and a FARGATE_SPOT capacity provider which AWS creates. You associate the capacity provider with your cluster, and then add them to a capacity provider strategy.

For Amazon ECS on Amazon EC2 users, a capacity provider consists of a capacity provider name, an Auto Scaling group. A capacity provider also consists of all of the settings for managed scaling and managed termination protection. When you turn on managed scaling, Amazon ECS scales Auto Scaling groups in and out on your behalf.

Default capacity provider strategy

You can associate a *default capacity provider strategy* with an Amazon ECS cluster. After you do this, Amazon ECS uses a default capacity provider strategy when you create service or run a standalone task in the cluster and don't specify a launch type or custom capacity provider. We recommend that you define a default capacity provider strategy for each cluster.

Capacity provider strategy

A capacity provider strategy consists of one or more capacity providers. You can specify an optional *base* and *weight* value for finer control. A capacity provider strategy is part of the configuration of a cluster, service, or task. However, you can't create re-useable capacity provider strategies. The capacity provider strategy of each cluster, service, or task capacity provider strategy is independent.

If the default capacity provider strategy for a cluster doesn't meet your capacity requirements, specify a custom *capacity provider strategy* when creating a service or running a standalone task.

Important

When you set a launch type instead of a capacity provider strategy on tasks in clusters where the capacity is managed by capacity providers, those tasks aren't counted for capacity provider scaling actions.

Only capacity providers that are both already associated with a cluster and have an ACTIVE or UPDATING status can be used in a capacity provider strategy. You can associate a capacity provider with a cluster when you create a cluster.

In a capacity provider strategy, the optional *base* value designates how many tasks, at a minimum, run on a specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

The *weight* value determines the relative percentage of the total number of launched tasks that use the specified capacity provider. Consider the following example. You have a strategy that contains two capacity providers, and both have a weight of 1. When the base percentage is reached, the tasks are split evenly across the two capacity providers. Using that same logic, suppose that you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*. Then, for every one task that's run using *capacityProviderA*, there are four tasks that use *capacityProviderB*.

Capacity provider types

For Amazon ECS workloads that are hosted on Fargate, the following predefined capacity providers are available:

- Fargate
- Fargate Spot

For Amazon ECS workloads that are hosted on Amazon EC2 instances, you must create and maintain a capacity provider that consists of the following components:

- A name
- An Auto Scaling group
- The settings for managed scaling and managed termination protection.

Capacity provider considerations

Consider the following when using capacity providers:

- A capacity provider must be associated with a cluster before being specified in a capacity provider strategy.
- When you specify a capacity provider strategy, the number of capacity providers that you can specify is limited to six.
- You can't update a service using an Auto Scaling group capacity provider to use a Fargate capacity provider. The opposite is also the case.
- In a capacity provider strategy, if no weight value is specified for a capacity provider in the console, then the default value of 1 is used. If using the API or AWS CLI, the default value of 0 is used.
- When multiple capacity providers are specified within a capacity provider strategy, at least one of the capacity providers must have a weight value that's greater than zero. Moreover, any capacity providers with a weight of zero aren't used to place tasks. If you specify multiple capacity providers in a strategy with all the same weight of zero, then any RunTask or CreateService actions using the capacity provider strategy fail.
- In a capacity provider strategy, only one capacity provider can have a defined *base* value. If no base value is specified, the default value of zero is used.
- A cluster can contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers. However, a capacity provider strategy can only contain Auto Scaling group or Fargate capacity providers, but not both.
- A cluster can contain a mix of services and standalone tasks that use both capacity providers and launch types. A service can be updated to use a capacity provider strategy rather than a launch type. However, you must force a new deployment when doing so.
- When you use managed termination protection, you must also use managed scaling. Otherwise, managed termination protection doesn't work.

The following sections provide information about the Fargate launch type and EC2 launch type capacity providers.

Topics

- [AWS Fargate capacity providers \(p. 222\)](#)
- [Amazon EC2 Auto Scaling group capacity providers \(p. 224\)](#)

AWS Fargate capacity providers

With Amazon ECS on AWS Fargate capacity providers, you can use both Fargate and Fargate Spot capacity with your Amazon ECS tasks.

With Fargate Spot, you can run interruption tolerant Amazon ECS tasks at a rate that's discounted compared to the Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks are interrupted with a two-minute warning.

Fargate capacity provider considerations

Consider the following when using Fargate capacity providers:

- Windows containers on Fargate don't support the Fargate Spot capacity provider.
- Linux tasks with the ARM64 architecture don't support the Fargate Spot capacity provider. Fargate Spot only supports Linux tasks with the X86_64 architecture.
- You don't need to create Fargate and Fargate Spot capacity providers. They're available to all accounts. To use them, all you need to do is associate them with a cluster.
- To associate Fargate and Fargate Spot capacity providers to a cluster, you must use the Amazon ECS API or AWS CLI. You cannot associate them using the console.
- The Fargate and Fargate Spot capacity providers are reserved and can't be deleted. However, you can disassociate them from a cluster using the PutClusterCapacityProviders API operation.
- You can associate a capacity provider with an existing cluster using the PutClusterCapacityProviders API operation.
- If you use Fargate Spot, your task must use platform version 1.3.0 or later (for Linux). For more information, see [AWS Fargate platform versions \(p. 77\)](#).
- When tasks that use the Fargate and Fargate Spot capacity providers are stopped, the task state change event is sent to Amazon EventBridge. The stopped reason describes the cause. For more information, see [Task state change events \(p. 563\)](#).
- A cluster can contain a mix of Fargate and Auto Scaling group capacity providers. However, a capacity provider strategy can only contain either Fargate or Auto Scaling group capacity providers, but not both. For more information, see [Auto Scaling Group Capacity Providers](#) in the *Amazon Elastic Container Service Developer Guide*.

Handling Fargate Spot termination notices

Understand that the following consequences because Spot capacity might not be available all the time.

- During periods of extremely high demand, Fargate Spot capacity might be unavailable. This can cause Fargate Spot tasks to be delayed. In these events, Amazon ECS services retry launching tasks until the required capacity becomes available. Fargate doesn't replace Spot capacity with on-demand capacity.
- When tasks using Fargate Spot capacity are stopped due to a Spot interruption, a two-minute warning is sent before a task is stopped. The warning is sent as a task state change event to Amazon EventBridge and as a SIGTERM signal to the running task. If you use Fargate Spot as part of a service, then in this scenario the service scheduler receives the interruption signal and attempts to launch additional tasks on Fargate Spot if there's capacity available. A service with only one task is interrupted until capacity is available. For more information about a graceful shutdown, see [Graceful shutdowns with ECS](#).

To ensure that your containers exit gracefully before the task stops, you can configure the following:

- A stopTimeout value of 120 seconds or less can be specified in the container definition that the task is using. The default stopTimeout value is 30 seconds. You can specify a longer stopTimeout value to give yourself more time between the moment that the task state change event is received and the point in time when the container is forcefully stopped. For more information, see [Container timeouts \(p. 831\)](#).
- The SIGTERM signal must be received from within the container to perform any cleanup actions. Failure to process this signal results in the task receiving a SIGKILL signal after the configured stopTimeout and may result in data loss or corruption.

The following is a snippet of a task state change event. This snippet displays the stopped reason and stop code for a Fargate Spot interruption.

```
{  
  "version": "0",
```

```
"id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
"detail-type": "ECS Task State Change",
"source": "aws.ecs",
"account": "111122223333",
"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",
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "STOPPED",
    "lastStatus": "RUNNING",
    "stoppedReason": "Your Spot Task was interrupted.",
    "stopCode": "SpotInterruption",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",
    ...
}
}
```

The following is an event pattern that's used to create an EventBridge rule for Amazon ECS task state change events. You can optionally specify a cluster in the `detail` field. Doing so means that you will receive task state change events for that cluster. For more information, see [Creating an EventBridge Rule in the Amazon EventBridge User Guide](#).

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "clusterArn": [
            "arn:aws:ecs:us-west-2:111122223333:cluster/default"
        ]
    }
}
```

Amazon EC2 Auto Scaling group capacity providers

When you use Amazon EC2 instances for your capacity, you use Auto Scaling groups to manage the Amazon EC2 instances registered to their clusters. Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application.

You can use the managed scaling feature to have Amazon ECS manage the scale-in and scale-out actions of the Auto Scaling group (managed scaling) or you can manage the scaling actions yourself. For more information, see [Amazon ECS cluster auto scaling \(p. 226\)](#).

Auto Scaling group capacity providers considerations

Consider the following when using Auto Scaling group capacity providers in the classic console:

- We recommend that you create a new empty Auto Scaling group to use with a capacity provider rather than using an existing one. If you use an existing Auto Scaling group, any Amazon EC2 instances that are associated with the group that were already running and registered to an Amazon ECS cluster before the Auto Scaling group being used to create a capacity provider might not be properly registered with the capacity provider. This might cause issues when using the capacity provider in a capacity provider strategy. Use `DescribeContainerInstances` to confirm whether a container instance is associated with a capacity provider or not.

Note

To create an empty Auto Scaling group, set the desired count to zero. After you created the capacity provider and associated it with a cluster, you can then scale it out.

When you use the Amazon ECS console **Create Cluster** with the **Amazon EC2 instances** option under **Infrastructure**, Amazon ECS creates an Amazon EC2 Auto Scaling launch configuration and Auto Scaling group on your behalf as part of the AWS CloudFormation stack. They are prefixed with `EC2ContainerService-<ClusterName>`, which make them easy to identify. You can use the Auto Scaling group as a capacity provider for that cluster.

- An Auto Scaling group must have a `MaxSize` greater than zero to scale out.
- The Auto Scaling group can't have instance weighting settings. Instance weighting isn't supported when used with an Amazon ECS capacity provider.
- If the Auto Scaling group can't scale out to accommodate the number of tasks run, the tasks fails to transition beyond the `PROVISIONING` state.
- Don't modify the scaling policy resource associated with your Auto Scaling groups that are managed by capacity providers.
- When you use managed termination protection, you must also use managed scaling. Otherwise, managed termination protection won't work.
- When managed scaling is turned on, the Auto Scaling group capacity provider creates a scaling policy resource to manage the scaling of your Auto Scaling group. You can identify these resources by the `ECSManaged` prefix.

When you use managed termination protection, Amazon ECS only terminates EC2 instances that don't have any running Amazon ECS tasks.

- If managed termination protection is turned on when you create a capacity provider, the Auto Scaling group and each Amazon EC2 instance in the Auto Scaling group must have instance protection from scale in turned on. For more information, see [Instance Protection](#) in the *AWS Auto Scaling User Guide*.
- You can add a warm pool to your Auto Scaling group. A warm pool is a group of pre-initialized Amazon EC2 instances that are ready to be included in the cluster whenever your application needs to scale out. For more information about warm pools, see [Using a warm pool for your Auto Scaling group \(p. 225\)](#).
- If managed scaling is turned on when you create a capacity provider, the Auto Scaling group desired count can be set to `0`. When managed scaling is turned on, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.

For more information about creating an Amazon EC2 Auto Scaling launch template, see [Launch Templates](#) in the *Amazon EC2 Auto Scaling User Guide*.

For more information about creating an Amazon EC2 Auto Scaling group, see [Auto Scaling groups](#) in the *Amazon EC2 Auto Scaling User Guide*.

Using a warm pool for your Auto Scaling group

Amazon ECS supports Amazon EC2 Auto Scaling warm pools. A warm pool is a group of pre-initialized Amazon EC2 instances ready to be placed into service. Whenever your application needs to scale out, Amazon EC2 Auto Scaling uses the pre-initialized instances from the warm pool rather than launching cold instances, allows for any final initialization process to run, and then places the instance into service.

To learn more about warm pools and how to add a warm pool to your Auto Scaling group, see [Warm pools for Amazon EC2 Auto Scaling](#) in the *Amazon EC2 Auto Scaling User Guide*.

When you create or update a warm pool for an Auto Scaling group for Amazon ECS , you cannot set the option that returns instances to the warm pool on scale in (`ReuseOnScaleIn`). For more information, see [put-warm-pool](#) in the *AWS Command Line Interface Reference*.

To use warm pools with your Amazon ECS cluster, set the `ECS_WARM_POOLS_CHECK` agent configuration variable to `true` in the **User data** field of your Amazon EC2 Auto Scaling group launch template. The following shows an example of how the agent configuration variable can be specified in the **User data** field of an Amazon EC2 launch template.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_WARM_POOLS_CHECK=true
EOF
```

The `ECS_WARM_POOLS_CHECK` variable is only supported on agent versions 1.59.0 and later. For more information about the variable, see [Amazon ECS container agent configuration \(p. 370\)](#).

Amazon ECS cluster auto scaling

Important

As of May 27, 2022, Amazon ECS changed how the resources that facilitate cluster auto scaling are managed. To learn more, see [Update on the way Amazon ECS creates resources for cluster auto scaling \(p. 232\)](#).

Amazon ECS can manage the scaling of Amazon EC2 instances that are registered to your cluster. This is referred to as Amazon ECS *cluster auto scaling*. This is done by using an Amazon ECS Auto Scaling group capacity provider with *managed scaling* turned on. When you use an Auto Scaling group capacity provider with managed scaling turned on, you set a target percentage (the `targetCapacity`) for the utilization of the instances in this Auto Scaling group. Amazon ECS creates two custom CloudWatch metrics and a target tracking scaling policy that attaches to your Auto Scaling group. Amazon ECS then manages the scale-in and scale-out actions of the Auto Scaling group based on the resource utilization that your tasks use from this capacity provider. For more information about Auto Scaling group capacity providers, see [the section called "Amazon EC2 Auto Scaling group capacity providers" \(p. 224\)](#).

Note

Amazon ECS cluster auto scaling is only supported with Auto Scaling group capacity providers. For Amazon ECS workloads that are hosted on AWS Fargate, see [the section called "AWS Fargate capacity providers" \(p. 222\)](#).

How cluster auto scaling works

Cluster auto scaling setup

The following is your workflow to use Amazon ECS cluster auto scaling. For more information, see [the section called "Turn on cluster auto scaling" \(p. 233\)](#).

1. Create an Auto Scaling group.
2. Create a capacity provider that uses that Auto Scaling group.
3. Turn on managed scaling for the capacity provider.
4. Associate the capacity provider with a cluster.
5. Run a task or create a service with a capacity provider strategy that uses the capacity provider.

The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.

6. (Optional) Create a default capacity provider strategy for the cluster.

Cluster auto scaling resources

For each Auto Scaling group capacity provider that's associated with a cluster, Amazon ECS creates and manages the following resources:

- A low metric value CloudWatch alarm
- A high metric value CloudWatch alarm
- A target tracking scaling policy

Note

Amazon ECS creates the target tracking scaling policy and attaches it to the Auto Scaling group. To update the target tracking scaling policy, update the capacity provider managed scaling settings, rather than updating the scaling policy directly.

When you turn off managed scaling or disassociate the capacity provider from a cluster, Amazon ECS removes both CloudWatch metrics and the target tracking scaling policy resources.

Cluster auto scaling metrics

The following metrics help to determine what actions to take:

CapacityProviderReservation

The percent of cluster container instances in use for a specific capacity provider. Amazon ECS generates this metric.

Amazon ECS sets the CapacityProviderReservation value to a number between 0-100. Amazon ECS uses the following formula to represent the ratio of how much capacity remains in the Auto Scaling group. Then, Amazon ECS publishes the metric to CloudWatch. For more information about how the metric is calculated, see [Deep Dive on Amazon ECS Cluster Auto Scaling](#)

$$\text{CapacityProviderReservation} = (\text{number of instances needed}) / (\text{number of running instances}) \times 100$$

DesiredCapacity

The amount of capacity for the Auto Scaling group.

Cluster auto scaling process

Amazon ECS performs the cluster auto scaling process for each capacity provider that's associated with your clusters. Amazon ECS performs the process every minute.

Amazon ECS publishes the CapacityProviderReservation metric to CloudWatch in the AWS/ECS/ManagedScaling namespace. The CapacityProviderReservation metric causes one of the following actions to occur:

The CapacityProviderReservation value equals targetCapacity

The Auto Scaling group doesn't need to scale in or scale out. The target utilization percentage has been reached.

The CapacityProviderReservation value is greater than targetCapacity

There are more tasks using a higher percentage of the capacity than your targetCapacity percentage. The increased value of the CapacityProviderReservation metric causes the

associated CloudWatch alarm to act. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. The Auto Scaling group uses this value to launch EC2 instances, and then register them with the cluster.

When the `targetCapacity` is the default value of 100 %, the new tasks are in the PENDING state during the scale-out because there is no available capacity on the instances to run the tasks. After the new instances register with ECS, these tasks will start on the new instances.

The `CapacityProviderReservation` value is less than `targetCapacity`

There are less tasks using a lower percentage of the capacity than your `targetCapacity` percentage and there is at least one instance that can be terminated. The decreased value of the `CapacityProviderReservation` metric causes the associated CloudWatch alarm to act. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. The Auto Scaling group uses this value to terminate EC2 container instances, and then deregister them from the cluster.

The Auto Scaling group follows the group termination policy to determine which instances it terminates first during scale-in events. Additionally it avoids instances with the instance scale-in protection setting turned on. Cluster auto scaling can manage which instances have the instance scale-in protection setting if you turn on managed termination protection. For more information about managed termination protection, see [Managed termination protection \(p. 229\)](#). For more information about how Auto Scaling groups terminate instances, see [Control which Auto Scaling instances terminate during scale in](#) in the *Amazon EC2 Auto Scaling User Guide*.

Cluster auto scaling considerations

Consider the following when using cluster auto scaling:

- Don't change or manage the desired capacity for the Auto Scaling group that's associated with a capacity provider with any scaling policies other than the one Amazon ECS manages.
- Amazon ECS uses the `AWSServiceRoleForECS` service-linked IAM role for the permissions that it requires to call AWS Auto Scaling on your behalf. For more information about using and creating Amazon ECS service-linked IAM roles, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).
- When using capacity providers with Auto Scaling groups, the user, group, or role that creates the capacity providers requires the `autoscaling:CreateOrUpdateTags` permission. This is because Amazon ECS adds a tag to the Auto Scaling group when it associates it with the capacity provider.

Important

Make sure any tooling that you use doesn't remove the `AmazonECSManaged` tag from the Auto Scaling group. If this tag is removed, Amazon ECS can't manage it when scaling your cluster.

- Cluster auto scaling doesn't modify the `MinimumCapacity` or `MaximumCapacity` for the group. For the group to scale out, the value for `MaximumCapacity` must be greater than zero.
- When Auto Scaling (managed scaling) is turned on, a capacity provider can only be connected to one cluster at the same time. If your capacity provider has managed scaling turned off, you can associate it with multiple clusters.
- When managed scaling is turned off, the capacity provider doesn't scale in or scale out. You can use a capacity provider strategy to balance your tasks between capacity providers.
- Amazon ECS uses placement strategy and placement constraints with the existing capacity at the current time. A placement strategy can spread tasks across Availability Zones or Amazon ECS instances. This eventually spreads all the tasks and all the instances so that each running task launches on its own dedicated instance. To prevent this, don't use the `spread` strategy together with the `binpack` strategy. For more information, see [the section called "Task placement strategies" \(p. 435\)](#).

Consider the following when you use the console:

- By default, the Amazon ECS *managed scaling* feature is on. For more information, see [Managed scale-out behavior \(p. 229\)](#).
- By default, *managed termination protection* is off. For more information, see the next section [the section called "Managed termination protection" \(p. 229\)](#).
- By default, Auto Scaling instance scale-in protection is off. For more information, see [Using instance scale-in protection](#) in the *Amazon EC2 Auto Scaling User Guide*.
- The Auto Scaling group that's used with your capacity provider can't use instance weighting settings. Instance weighting isn't supported when used together with an Amazon ECS capacity provider.

Managed termination protection

Important

You must turn on Auto Scaling *instance scale-in protection* on the Auto Scaling group to use the managed termination protection feature of cluster auto scaling.

Amazon ECS cluster auto scaling scales in the Auto Scaling group when the CapacityProviderReservation value is less than targetCapacity percentage that you set. Cluster auto scaling can control which instances are terminated if you turn on *managed termination protection*. When you use managed termination protection, Amazon ECS only terminates EC2 instances that don't have any running Amazon ECS tasks. Tasks that are run by a service that uses the DAEMON scheduling strategy are ignored and an instance can be terminated by cluster auto scaling even when the instance is running these tasks. This is because all of the instances in the cluster are running these tasks.

When you use managed termination protection, Amazon ECS first turns on the *instance scale-in protection* option for the EC2 instances in the Auto Scaling group. Then, Amazon ECS places the tasks on the instances. When all non-daemon tasks are stopped on an instance, Amazon ECS initiates the scale-in process and turns off scale-in protection for the EC2 instance. The Auto Scaling group can then terminate the instance.

Auto Scaling *instance scale-in protection* controls which EC2 instances can be terminated by Auto Scaling. Instances with the scale-in feature turned on can't be terminated during the scale-in process. For more information about Auto Scaling instance scale-in protection, see [Using instance scale-in protection](#) in the *Amazon EC2 Auto Scaling User Guide*.

For more information about how cluster auto scaling manages scale-in, see the next section [Managed scale-in behavior \(p. 231\)](#).

Managed termination protection considerations

Consider the following when using managed termination protection with the new console:

- By default, managed termination protection is turned off for new capacity providers that you create in the console.
- The console doesn't turn on the instance scale-in protection of the Auto Scaling group that you select. By default, Auto Scaling instance scale-in protection is off. You must turn on Auto Scaling instance scale-in protection on the Auto Scaling group to use managed termination protection. If you don't turn on scale-in protection, then turning on managed termination protection can lead to undesirable behavior. For example, you may have instances stuck in draining state. For more information, see [Using instance scale-in protection](#) in the *Amazon EC2 Auto Scaling User Guide*.

Managed scale-out behavior

When you have Auto Scaling group capacity providers that use managed scaling, Amazon ECS estimates the optimal number of instances to add to your cluster. Then, Amazon ECS uses this value to determine how many instances to request. The following describes the scale-out behavior in more detail.

1. Amazon ECS selects a capacity provider for each task by following the capacity provider strategy from the service, from the standalone task, or from the cluster default. Amazon ECS follows the rest of these steps for a single capacity provider.

If a task doesn't have a capacity provider strategy, then that task is ignored by capacity providers. A pending task that doesn't have a capacity provider strategy won't cause any capacity provider to scale out. Tasks or services can't set a capacity provider strategy if that task or service sets a launch type.

2. Group all of the provisioning tasks for this capacity provider so that each group has the same exact resource requirements.
3. When you use multiple instance types in an Auto Scaling group, the instance types in the Auto Scaling group are sorted by their parameters. These parameters include vCPU, memory, elastic network interfaces (ENIs), ports, and GPUs. The smallest and the largest instance types for each parameter are selected. For more information about how to choose the instance type, see [Characterizing your application](#) in the *Amazon ECS Best Practices Guide*.

Important

If a group of tasks have resource requirements that are greater than the smallest instance type in the Auto Scaling group, then that group of tasks can't run with this capacity provider. The capacity provider doesn't scale the Auto Scaling group. The tasks remain in the PENDING state.

To prevent tasks from staying in the PENDING state, we recommend that you create separate Auto Scaling groups and capacity providers for different minimum resource requirements. When you run tasks or create services, only add capacity providers to the capacity provider strategy that can run the task on the smallest instance type in the Auto Scaling group. For other parameters, you can use placement constraints

4. For each group of tasks, Amazon ECS calculates the number of instances that are required to run the unplaced tasks. This calculation uses a binpack strategy. This strategy accounts for the vCPU, memory, elastic network interfaces (ENI), ports, and GPUs requirements of the tasks. It also accounts for the resource availability of the Amazon EC2 instances. The values for the largest instance types are treated as the maximum calculated instance count. The values for the smallest instance type are used as protection. If the smallest instance type can't run at least one instance of the task, the calculation considers the task as not compatible. As a result, the task is excluded from scale-out calculation. When all the tasks aren't compatible with the smallest instance type, cluster auto scaling stops and the CapacityProviderReservation value remains at the targetCapacity value.
5. Amazon ECS publishes the CapacityProviderReservation metric to CloudWatch with respect to the minimumScalingStepSize if either of the following is the case. Either, the maximum calculated instance count is less than the minimum scaling step size. Or, the lower value of either the maximumScalingStepSize or the maximum calculated instance count.
6. CloudWatch alarms use the CapacityProviderReservation metric for capacity providers. When the CapacityProviderReservation metric is greater than the targetCapacity value, alarms also increase the DesiredCapacity of the Auto Scaling group. The targetCapacity value is a capacity provider setting that's sent to the CloudWatch alarm during the cluster auto scaling activation phase.

You can set the targetCapacity when you create the Auto Scaling group, or modify the value after the group is created. The default is 100%.

7. The Auto Scaling group launches additional EC2 instances. To prevent over-provisioning of the scale-out operation, Auto Scaling makes sure that recently launched EC2 instance capacity is stabilized before it launches new instances. Auto Scaling checks if all existing instances have passed the instanceWarmupPeriod (now minus the instance launch time). The scale-out is blocked for Instances that are within the instanceWarmupPeriod.

The default number of seconds for a newly launched instance to warm up is 300.

For more information, see [Deep dive on Amazon ECS cluster auto scaling](#).

Scale-out considerations

Consider the following for the scale-out process:

1. Although there are multiple placement constraints, we recommend that you only use the `distinctInstance` task placement constraint. This prevents the scale-out process from stopping because you're using a placement constraint that's not compatible with the sampled instances.
2. Managed scaling works best if your Auto Scaling group uses the same or similar instance types. For more information, see [Managed scale-out behavior \(p. 229\)](#).
3. When a scale-out process is required and there are no currently running container instances, Amazon ECS always scales-out to two instances initially, and then performs additional scale-out or scale-in processes. Any additional scale-out waits for the instance warmup period. For scale-in processes, Amazon ECS waits 15 minutes after a scale-out process before starting scale-in processes at all times.
4. The second scale-out step needs to wait until the `instanceWarmupPeriod` expires, which might affect the overall scale limit. If you need to reduce this time, make sure that `instanceWarmupPeriod` is large enough for the EC2 instance to launch and start the Amazon ECS agent (which prevents overprovisioning).
5. Cluster auto scaling supports Launch Configuration, Launch Templates, and multiple instance types in the capacity provider Auto Scaling group. You can also use attribute-based instance type selection without multiple instance types.
6. When using an Auto Scaling group with On-Demand instances and multiple instance types or Spot Instances, place the larger instance types higher in the priority list and don't specify a weight. Specifying a weight isn't supported at this time. For more information, see [Auto Scaling groups with multiple instance types](#) in the *AWS Auto Scaling User Guide*.
7. Amazon ECS then launch either the `minimumScalingStepSize`, if the maximum calculated instance count is less than the minimum scaling step size, or the lower of either the `maximumScalingStepSize` or the maximum calculated instance count value.
8. If an Amazon ECS service or the `run-task` API launches a task and the capacity provider container instances don't have enough resources to start the task, then Amazon ECS limits the number of tasks with this status for each cluster and prevents any tasks from exceeding this limit. For more information, see [the section called "Service quotas" \(p. 536\)](#).

Managed scale-in behavior

Amazon ECS monitors container instances for each capacity provider within a cluster. When a container instance isn't running any tasks, the container instance is considered empty and Amazon ECS starts the scale-in process. The following describes the scale-in behavior in more detail:

1. Amazon ECS calculates the number of container instances that are empty. A container instance is considered empty even when daemon tasks are running.
2. Amazon ECS sets the `CapacityProviderReservation` value to a number between 0-100 that uses the following formula to represent the ratio of how big the Auto Scaling group needs to be relative to how big it actually is, expressed as a percentage. Then, Amazon ECS publishes the metric to CloudWatch. For more information about how the metric is calculated, see [Deep Dive on Amazon ECS Cluster Auto Scaling](#)

$$\text{CapacityProviderReservation} = (\text{number of instances needed}) / (\text{number of running instances}) \times 100$$

3. The `CapacityProviderReservation` metric generates a CloudWatch alarm. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. Then, one of the following actions occurs:
 - If you don't use capacity provider managed termination, the Auto Scaling group selects EC2 instances using the Auto Scaling group termination policy and terminates the instances until

the number of EC2 instances reaches the `DesiredCapacity`. The container instances are then deregistered from the cluster.

- If all the container instances use managed termination protection, Amazon ECS removes the scale-in protection on the container instances that are empty. The Auto Scaling group will then be able to terminate the EC2 instances. The container instances are then deregistered from the cluster.

Scale-in considerations

Consider the following for the scale-in process:

- When there are no running non-daemon tasks, Amazon ECS container instances are considered available for scale in.
- CloudWatch scale-in alarms require 15 data points (15 minutes) before the scale-in process for the Auto Scaling group starts. After the scale-in process starts until Amazon ECS needs to reduce the number of registered container instances, the Auto Scaling group sets the `DesireCapacity` value to be greater than one instance and less than 50% each minute.
- When Amazon ECS requests a scale-out (when `CapacityProviderReservation` is greater than 100) while a scale-in process is in progress, the scale-in process is stopped and starts from the beginning if required.

Target tracking considerations

When creating or updating a capacity provider with managed scaling turned on, you can set the `targetCapacity` percentage. This way, you can keep spare capacity so that future tasks can launch more quickly. Spare capacity improves this by not waiting for the Auto Scaling group to launch more instances. Amazon ECS uses the target capacity value to manage the CloudWatch metric that the service creates to facilitate cluster auto scaling. Amazon ECS manages the CloudWatch metric. This way, the Auto Scaling group is treated as a steady state so that no scaling action is required. The values can be from 0-100%. For example, to configure Amazon ECS to keep 10% free capacity on top of that used by Amazon ECS tasks, set the target capacity value to 90%.

Consider the following when setting the `targetCapacity` value on a capacity provider.

- A `targetCapacity` value of less than 100% represents the amount of free capacity (Amazon EC2 instances) that need to be present in the cluster. Free capacity means that there are no running tasks.
- Placement constraints such as Availability Zones, without additional binpack forces Amazon ECS to eventually run one task for each instance, which might not be the desired behavior. To prevent this behavior, don't use the spread strategy together with the binpack strategy.

Update on the way Amazon ECS creates resources for cluster auto scaling

As of May 27, 2022, Amazon ECS changed how it manages the resources that facilitate cluster Auto Scaling. To simplify the experience, Amazon ECS no longer requires an AWS Auto Scaling scaling plan when you turn on managed scaling for an Auto Scaling group capacity provider.

Important

This change has no functional impact on your cluster auto scaling workflows and no pricing or performance impact.

Capacity providers created before May 27, 2022

Capacity providers that were created before May 27, 2022, and that use AWS Auto Scaling scaling plans continue to function as before.

Review the following considerations:

- We don't recommend that you update or delete the ECS-managed AWS Auto Scaling scaling plan or the scaling policy resources that are associated with your capacity providers.
- You can access the scaling plan resource for clusters on the Auto Scaling console and by the `describe-clusters` with attachments. For more information, see the API documentation [DescribeClusters](#).
- You can't add scaling policies to the Auto Scaling group that functions as the cluster capacity provider.
- The number of Auto Scaling plans for each account is limited. For more information, see [Quotas for your scaling plans](#) in the *Amazon EC2 Auto Scaling User Guide*.

Capacity providers created on or after May 27, 2022

As of May 27, 2022, Amazon ECS no longer creates an AWS Auto Scaling scaling plan for newly created capacity providers. Instead, Amazon ECS uses the target tracking scaling policy attached to the Auto Scaling group to perform dynamic scaling based on your target capacity specification. For more information, see [Amazon EC2 Auto Scaling group capacity providers \(p. 224\)](#).

With this new release, you can use an existing Auto Scaling group with a scaling policy for use when creating a new capacity provider. We don't recommend that you modify the ECS managed scaling policy or plan resources. However, when creating new capacity provider resources, if you have customized tooling that made modifications to the AWS Auto Scaling scaling plan, do one of the following:

- (Recommended) Update your capacity provider to modify the Amazon ECS managed scaling settings. For more information, see [UpdateCapacityProvider](#).
- Update the scaling policy associated with your Auto Scaling group to modify the target tracking configuration used. For more information, see [PutScalingPolicy](#).

Turn on cluster auto scaling

You can use the AWS CLI to turn on cluster auto scaling.

Before you begin, create an Auto Scaling group and a capacity provider. For more information, see [the section called "Amazon EC2 Auto Scaling group capacity providers" \(p. 224\)](#).

Associate the capacity provider with the cluster

Use the following steps to associate the capacity provider with the cluster.

1. Use the `put-cluster-capacity-providers` command to associate one or more capacity providers with the cluster. To add the AWS Fargate capacity providers, simply include the `FARGATE` and `FARGATE_SPOT` capacity providers in the request. For more information, see [put-cluster-capacity-providers](#) in the *AWS CLI Command Reference*.

```
aws ecs put-cluster-capacity-providers \
--cluster ClusterName \
--capacity-providers CapacityProviderName FARGATE FARGATE_SPOT \
--default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
```

2. Use the `describe-clusters` command to verify that the association was successful. For more information, see [describe-clusters](#) in the *AWS CLI Command Reference*.

```
aws ecs describe-clusters \
--cluster ClusterName \
--include ATTACHMENTS
```

Turn on managed scaling for the capacity provider

Use the following steps to turn on managed scaling for the capacity provider.

- Use the `update-capacity-provider` command to turn on managed auto scaling for the capacity provider. For more information, see [update-capacity-provider](#) in the *AWS CLI Command Reference*.

```
aws ecs update-capacity-provider \
--capacity-providers CapacityProviderName \
--auto-scaling-group-provider managedScaling=ENABLED
```

Turn off cluster auto scaling

You can use the AWS CLI to turn off cluster auto scaling.

To turn off cluster auto scaling for a cluster, you can either disassociate the capacity provider with managed scaling turned on from the cluster or by updating the capacity provider to turn off managed scaling.

Disassociate the capacity provider with the cluster

Use the following steps to disassociate a capacity provider with a cluster.

1. Use the `put-cluster-capacity-providers` command to disassociate the Auto Scaling group capacity provider with the cluster. The cluster can keep the association with the AWS Fargate capacity providers. For more information, see [put-cluster-capacity-providers](#) in the *AWS CLI Command Reference*.

```
aws ecs put-cluster-capacity-providers \
--cluster ClusterName \
--capacity-providers FARGATE FARGATE_SPOT \
--default-capacity-provider-strategy '[' ]'
```

2. Use the `describe-clusters` command to verify that the disassociation was successful. For more information, see [describe-clusters](#) in the *AWS CLI Command Reference*.

```
aws ecs describe-clusters \
--cluster ClusterName \
--include ATTACHMENTS
```

Turn off managed scaling for the capacity provider

Use the following steps to turn off managed scaling for the capacity provider.

- Use the `update-capacity-provider` command to turn off managed auto scaling for the capacity provider. For more information, see [update-capacity-provider](#) in the *AWS CLI Command Reference*.

```
aws ecs update-capacity-provider \
--capacity-providers CapacityProviderName \
--auto-scaling-group-provider managedScaling=DISABLED
```

Cluster management

The following are general concepts about Amazon ECS clusters.

- Amazon ECS creates a default cluster. You can create additional clusters in an account to keep your resources separate.
- Clusters are AWS Region specific.
- The following are the possible states that a cluster can be in.

ACTIVE

The cluster is ready to accept tasks and, if applicable, you can register container instances with the cluster.

PROVISIONING

The cluster has capacity providers associated with it and the resources needed for the capacity provider are being created.

DEPROVISIONING

The cluster has capacity providers associated with it and the resources needed for the capacity provider are being deleted.

FAILED

The cluster has capacity providers associated with it and the resources needed for the capacity provider have failed to create.

INACTIVE

The cluster has been deleted. Clusters with an INACTIVE status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so make sure you do not rely on INACTIVE clusters persisting.

- A cluster can contain a mix of tasks that are hosted on AWS Fargate, Amazon EC2 instances, or external instances. Tasks can run on Fargate or EC2 infrastructure as a launch type or a capacity provider strategy. If you use EC2 as a launch type, ECS doesn't track and scale the capacity of Amazon EC2 Auto Scaling groups. For more information about launch types, see [Amazon ECS launch types \(p. 85\)](#).
- A cluster can contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers. However, when you specify a capacity provider strategy, they may only contain one or the other but not both. For more information, see [Amazon ECS capacity providers \(p. 220\)](#).
- For tasks that use the EC2 launch type or Auto Scaling group capacity providers, clusters can contain multiple different container instance types. However, each container instance can only be registered to one cluster at a time.
- Custom IAM policies may be created to allow or restrict user access to specific clusters. For more information, see the [Cluster examples \(p. 605\)](#) section in [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).
- *You can configure a default Service Connect namespace for a cluster. After you set a default Service Connect namespace, any new services created in the cluster can be added as client services

in the namespace by turning on Service Connect. No additional configuration is required. For more information, see [Service Connect \(p. 503\)*](#).

- If you use EC2 instances, the cluster capacity can be located in any of the following AWS resources:

For information about how to use these resources with Amazon ECS see [Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts \(p. 664\)](#).

- Availability Zones
- Local Zones
- Wavelength Zones
- AWS Outposts

Creating a cluster for the Fargate launch type using the console

You can create an Amazon ECS cluster using the new Amazon ECS experience. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#) and assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#). The new Amazon ECS experience provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

To make the cluster creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

By default, we create an Amazon ECS cluster for the Fargate launch type with the following properties:

- Uses Fargate and Fargate Spot capacity providers.
- Launches tasks and services in all the default subnets in the default VPC for your selected Region.
- Does not use Container Insights.
- Has three tags configured for AWS CloudFormation.
- Creates a default namespace in AWS Cloud Map that is the same name as the cluster. A namespace allows services that you create in the cluster to connect to the other services in the namespace without additional configuration.

For more information, see [Interconnecting services \(p. 502\)](#).

You can modify the following default options:

- Change the subnets where tasks and services launch into by default.
- Change the default namespace associated with the cluster.
- Turn on Container Insights.

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. For more information, see [the section called "CloudWatch Container Insights" \(p. 572\)](#).

- Add tags to help you identify your clusters.

To create a new cluster (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.

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. Under **Cluster configuration**, for **Cluster name**, enter a unique name.
The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
 6. (Optional) To change the name of the default namespace, for **Namespace**, enter a unique name.
 7. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
 - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.
 - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
 8. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
 9. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.
[Add a tag] Choose **Add tag** and do the following:
 - For **Key**, enter the key name.
 - For **Value**, enter the key value.
- [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

Next steps

After you create the cluster you can create task definitions for your applicants, and then run them as standalone tasks, or part of a service. For more information, see the following:

- [Amazon ECS task definitions \(p. 83\)](#)
- [Running a standalone task using the Amazon ECS console \(p. 430\)](#)
- [Creating a service using the console \(p. 456\)](#)

Creating a cluster for the Amazon EC2 launch type using the console

You can create an Amazon ECS cluster using the new Amazon ECS experience. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#) and assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#). The new Amazon ECS experience provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

To make the cluster creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

You can register Amazon EC2 instances when you create the cluster or register additional instances with the cluster after it has been created.

You can modify the following default options:

- Change the subnets where tasks and services launch into by default.

- Change the default namespace associated with the cluster. A namespace allows services that you create in the cluster can connect to the other services in the namespace without additional configuration. The default namespace is the same as the cluster name.

For more information, see [Interconnecting services \(p. 502\)](#).

- Turn on Container Insights.

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. For more information, see [the section called "CloudWatch Container Insights" \(p. 572\)](#).

- Add tags to help you identify your clusters.

Auto Scaling group options

If you want to use Spot Instances in your Auto Scaling group, you must use the classic console to create the cluster. For more information, see [the section called "Creating a cluster using the classic console" \(p. 879\)](#).

When you use Amazon EC2 instances, you must specify an Auto Scaling group to manage the infrastructure that your tasks and services run on.

When you choose to create a new Auto Scaling group, it is automatically configured for the following behavior:

- Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.
- Amazon ECS will not prevent Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. For more information, see [Instance Protection](#) in the [AWS Auto Scaling User Guide](#).

You configure the following Auto Scaling group properties which determine the type and number of instances to launch for the group:

- The Amazon ECS-optimized AMI.
- The instance type.
- The SSH key pair that proves your identity when you connect to the instance. For information about how to create SSH keys, see [Amazon EC2 key pairs and Linux instances](#) in the [Amazon EC2 User Guide for Linux Instances](#).
- The minimum number of instances to launch for the Auto Scaling group.
- The maximum number of instances that are started for the Auto Scaling group.

In order for the group to scale out, the maximum must be greater than 0.

Amazon ECS creates an Amazon EC2 Auto Scaling launch template and Auto Scaling group on your behalf as part of the AWS CloudFormation stack. The values that you specified for the AMI, the instance types, and the SSH key pair are part of the launch template. The templates are prefixed with `EC2ContainerService-<ClusterName>`, which makes them easy to identify. The Auto Scaling groups are prefixed with `<ClusterName>-ECS-Infra-ECSAutoScalingGroup`.

Instances launched for the Auto Scaling group use the launch template.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.

6. (Optional) To change the name of the default namespace, for **Namespace**, enter a unique name.
7. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
 - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.
 - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
8. (Optional) To add Amazon EC2 instances to your cluster, expand **Infrastructure**, and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider:
 - a. To use an existing Auto Scaling group, from **Auto Scaling group (ASG)**, select the group.
 - b. To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:
 - For **Operating system/Architecture**, choose the Amazon ECS-optimized AMI for the Auto Scaling group instances.
 - For **EC2 instance type**, choose the instance type for your workloads.

Managed scaling works best if your Auto Scaling group uses the same or similar instance types.

 - For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.
 - For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group.
9. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
10. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

[Add a tag] Choose **Add tag** and do the following:

 - For **Key**, enter the key name.
 - For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
11. Choose **Create**.

Next steps

After you create the cluster you can create task definitions for your applicants, and then run them as standalone tasks, or part of a service. For more information, see the following:

- [Amazon ECS task definitions \(p. 83\)](#)
- [Running a standalone task using the Amazon ECS console \(p. 430\)](#)
- [Creating a service using the console \(p. 456\)](#)

Updating a cluster using the console

You can modify the following cluster properties:

- Set a default capacity provider

Each cluster can have one or more capacity providers and an optional capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.

Capacity providers are an alternative to launch types. For more information, see [Capacity provider concepts \(p. 220\)](#).

- Turn on Container Insights.

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. For more information, see [the section called "CloudWatch Container Insights" \(p. 572\)](#).

- Add tags to help you identify your clusters.

To update the cluster (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : name** page, choose **Update cluster**.
5. To set the default capacity provider, under **Default capacity provider strategy**, choose **Add more**.
 - a. For **Capacity provider**, choose the capacity provider.
 - b. (Optional) For **Base**, enter the minimum number of tasks that run on the capacity provider.

You can only set a **Base** value for one capacity provider.
 - c. (Optional) For **Weight**, enter the relative percentage of the total number of launched tasks that use the specified capacity provider.
 - d. (Optional) Repeat the steps for any additional capacity providers.
6. To turn on or off Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. To help identify your cluster, expand **Tags**, and then configure your tags.

[Add a tag] Choose **Add tag** and do the following:

- For **Key**, enter the key name.
- For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

8. Choose **Update**.

Deleting a cluster using the console

If you are finished using a cluster, you can delete it. After you delete the cluster, it transitions to the INACTIVE state. Clusters with an INACTIVE status may remain discoverable in your account for a period

of time. However, this behavior is subject to change in the future, so you should not rely on INACTIVE clusters persisting.

Before you delete a cluster, you must perform the following operations:

- Delete all services in the cluster. For more information, see [the section called "Deleting a service using the classic console" \(p. 913\)](#).
- Stop all currently running tasks. For more information, see [the section called "Stopping tasks using the console" \(p. 434\)](#).
- Deregister all registered container instances in the cluster. For more information, see [the section called "Deregister a container instance" \(p. 335\)](#).
- If you created your cluster with the new console, delete the AWS CloudFormation stack that was created for your cluster. The stack is named **cluster-name-ECS-Infra**. For example, if the cluster name is "example-cluster-new-console", then the stack name is example-cluster-new-console-ECS-Infra. For more information, see [Deleting a stack on the AWS CloudFormation console](#) in the [AWS CloudFormation User Guide](#).
- Delete the namespace. For more information, see [Deleting namespaces](#) in the [AWS Cloud Map Developer Guide](#).

If you created a cluster using the classic console and you receive an error when you use the new console, you might need to delete the cluster using the classic console. For more information, see [the section called "Deleting a cluster using the classic console" \(p. 886\)](#).

To delete a cluster (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. In the upper-right of the page, choose **Delete Cluster**.

A message is displayed when you did not delete all the resources associated with the cluster.

6. In the confirmation box, enter **delete *cluster name***.

Creating a capacity provider for a cluster using the console

After the cluster creation completes, you can create a new capacity provider (Auto Scaling group) for the EC2 launch type.

Before you create the capacity provider, you need to create an Auto Scaling group. For more information, see [Auto Scaling groups](#) in the [Amazon EC2 Auto Scaling User Guide](#).

To create a capacity provider for the cluster (New Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *name*** page, choose **Infrastructure**, and then choose **Create**.
5. On the **Create capacity providers** page, configure the following options.

- a. Under **Basic details**, for **Capacity provider name**, enter a unique capacity provider name.
 - b. Under **Auto Scaling group**, for **Use an existing Auto Scaling group**, choose the Auto Scaling group.
 - c. (Optional) To configure a scaling policy, under **Scaling policies**, configure the following options.
 - To have Amazon ECS manage the scale-in and scale-out actions, select **Turn on managed scaling**.
 - To prevent EC2 instances with running Amazon ECS tasks from being terminated, select **Turn on scaling protection**.
 - For **Set target capacity**, enter the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy.
6. Choose **Create**.

Updating a capacity provider for a cluster using the console

When you use an Auto Scaling group as a capacity provider, you can modify the group's scaling policy.

To update a capacity provider for the cluster (New Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
 2. In the navigation pane, choose **Clusters**.
 3. On the **Clusters** page, choose the cluster.
 4. On the **Cluster : name** page, choose **Infrastructure**, and then choose **Update**.
 5. On the **Create capacity providers** page, configure the following options.
 - Under **Auto Scaling group**, under **Scaling policies**, configure the following options.
 - To have Amazon ECS manage the scale-in and scale-out actions, select **Turn on managed scaling**.
 - To prevent EC2 instances with running Amazon ECS tasks from being terminated, select **Turn on scaling protection**.
 - For **Set target capacity**, enter the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy.
6. Choose **Update**.

Deleting a capacity provider for a cluster using the console

If you are finished using an Auto Scaling group capacity provider, you can delete it. After the group is deleted, the Auto Scaling group capacity provider will transition to the INACTIVE state. Capacity providers with an INACTIVE status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on INACTIVE capacity providers persisting. Before the Auto Scaling group capacity provider is deleted, the capacity provider must be removed from the capacity provider strategy from all services. You can use the `UpdateService` API or the update service workflow in the Amazon ECS console to remove a capacity provider from a service's capacity provider strategy. Use the `force` new deployment option to ensure that any tasks using the Amazon EC2 instance capacity provided by the capacity provider are transitioned to use the capacity from the remaining capacity providers.

To delete a capacity provider for the cluster (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : name** page, choose **Infrastructure**, the Auto Scaling group, and then choose **Delete**.
5. In the confirmation box, enter **delete Auto Scaling group name**
6. Choose **Delete**.

Account settings

You can go into Amazon ECS account settings to opt in or out of specific features. For each AWS Region, you can opt in to, or opt out of, each account setting at the account-level or for a specific user or role.

You might want to opt in or out of specific features if any of the following is relevant to you:

- A user or role can opt in or opt out specific account settings for their individual account.
- A user or role can set the default opt-in or opt-out setting for all users on the account.
- The root user can opt in to, or opt out of, any specific role or user on the account. If the account setting for the root user is changed, it sets the default for all the users and roles that no individual account setting was selected for.

Note

Federated users assume the account setting of the root user and can't have explicit account settings set for them separately.

The following account settings are available. The opt-in and opt-out option must be selected for each account setting separately.

Amazon Resource Names (ARNs) and IDs

Resource names: `serviceLongArnFormat`, `taskLongArnFormat`, and `containerInstanceLongArnFormat`

Amazon ECS is introducing a new format for Amazon Resource Names (ARNs) and resource IDs for Amazon ECS services, tasks, and container instances. The opt-in status for each resource type determines the Amazon Resource Name (ARN) format the resource uses. You must opt in to the new ARN format to use features such as resource tagging for that resource type. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 246\)](#).

Only resources launched after opting in receive the new ARN and resource ID format. All existing resources aren't affected. For Amazon ECS services and tasks to transition to the new ARN and resource ID formats, you must recreate the service or task. To transition a container instance to the new ARN and resource ID format, the container instance must be drained and a new container instance that's registered to the cluster.

Note

Tasks launched by an Amazon ECS service can only receive the new ARN and resource ID format if the service was created on or after November 16, 2018, and the user who created the service has opted in to the new format for tasks.

AWSVPC trunking

Resource name: `awsvpcTrunking`

Amazon ECS supports launching container instances with increased elastic network interface (ENI) density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. You can use this configuration to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.1large` instance with `awsvpcTrunking` enabled has an increased ENI quota of ten. The container instance has a primary network interface, and Amazon ECS creates and attaches a "trunk" network interface to the container instance. The primary network interface and the trunk network interface don't count against the ENI quota. Therefore, you can use this configuration to launch ten tasks on the container instance instead of the current two tasks. For more information, see [Elastic network interface trunking \(p. 283\)](#).

Only resources launched after opting in receive the increased ENI limits. All the existing resources aren't affected. To transition a container instance to the increased ENI quotas, the container instance must be drained and a new container instance registered to the cluster.

CloudWatch Container Insights

Resource name: `containerInsights`

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).

When you opt in to the `containerInsights` account setting, all new clusters have Container Insights enabled by default. You can disable this setting for specific clusters when you create them. You can also change this setting by using the `UpdateClusterSettings` API.

For clusters that contain tasks or services using the EC2 launch type, your container instances must run version 1.29.0 or later of the Amazon ECS agent to use Container Insights. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).

Dual-stack VPC IPv6

Resource name: `dualStackIPv6`

Amazon ECS supports providing tasks with an IPv6 address in addition to the primary private IPv4 address.

For tasks to receive an IPv6 address, the task must use the `awsvpc` network mode, must be launched in a VPC configured for dual-stack mode, and the `dualStackIPv6` account setting must be enabled. For more information about other requirements, see [Using a VPC in dual-stack mode \(p. 94\)](#).

Important

The `dualStackIPv6` account setting can only be changed using either the Amazon ECS API or the AWS CLI. For more information, see [Modifying account settings \(p. 249\)](#).

If you had a running task using the `awsvpc` network mode in an IPv6 enabled subnet between the dates of October 1, 2020 and November 2, 2020, the default `dualStackIPv6` account setting in the Region that the task was running in is disabled. If that condition isn't met, the default `dualStackIPv6` setting in the Region is enabled.

Fargate FIPS-140 compliance

Resource name: `fargateFIPSMODE`

Fargate supports the Federal Information Processing Standard (FIPS-140) which specifies the security requirements for cryptographic modules that protect sensitive information. It is the current United States and Canadian government standard, and is applicable to systems that are required to be compliant with Federal Information Security Management Act (FISMA) or Federal Risk and Authorization Management Program (FedRAMP).

You must turn on FIPS-140 compliance. For more information, see [the section called "AWS Fargate FIPS-140 compliance" \(p. 657\)](#).

Tag Resource Authorization

Resource name: `tagResourceAuthorization`

Some Amazon ECS API actions allow you to specify tags when you create the resource.

Amazon ECS is introducing tagging authorization for resource creation. Users must have permissions for action that creates the resource, such as `ecsCreateCluster`. If tags are specified in the

resource-creating action, AWS performs additional authorization on the `ecs:TagResource` action to verify if users or roles have permissions to create tags. Therefore, you must grant explicit permissions to use the `ecs:TagResource` action. For more information, see [the section called "Tag resources during creation" \(p. 651\)](#).

Topics

- [Amazon Resource Names \(ARNs\) and IDs \(p. 246\)](#)
- [ARN and resource ID format timeline \(p. 247\)](#)
- [AWS Fargate Federal Information Processing Standard \(FIPS-140\) compliance \(p. 247\)](#)
- [Tagging authorization \(p. 248\)](#)
- [Tagging authorization timeline \(p. 248\)](#)
- [Viewing account settings using the console \(p. 249\)](#)
- [Modifying account settings \(p. 249\)](#)
- [Reverting to the default Amazon ECS account settings \(p. 249\)](#)
- [Account setting management using the AWS CLI \(p. 249\)](#)

Amazon Resource Names (ARNs) and IDs

When Amazon ECS resources are created, each resource is assigned a unique Amazon Resource Name (ARN) and resource identifier (ID). If you use 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 use the [stop-task](#) AWS CLI command to stop a task, you must specify the task ARN or ID in the command.

You can opt in to and opt out of the new Amazon Resource Name (ARN) and resource ID format on a per-Region basis. Currently, any new account created is opted in by default.

You can opt in or opt out of the new Amazon Resource Name (ARN) and resource ID format at any time. After you opted in, any new resources that you create use the new format.

Note

A resource ID doesn't change after it's created. Therefore, opting in or out of the new format doesn't affect your existing resource IDs.

The following sections describe how ARN and resource ID formats are changing. For more information about the transition to the new formats, see [Amazon Elastic Container Service FAQ](#).

Amazon Resource Name (ARN) format

Some resources have a user-friendly name, such as a service named `production`. In other cases, you must specify a resource using the Amazon Resource Name (ARN) format. The new ARN format for Amazon ECS tasks, services, and container instances includes the cluster name. For information about opting in to the new ARN format, see [Modifying account settings \(p. 249\)](#).

The following table shows both the current format and the new format for each resource type.

Resource type	ARN
Container instance	Current: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:container-instance/<i>container-instance-id</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:container-instance/<i>cluster-name</i>/<i>container-instance-id</i></code>

Resource type	ARN
Amazon ECS service	Current: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:service/<i>service-name</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:service/<i>cluster-name</i>/<i>service-name</i></code>
Amazon ECS task	Current: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:task/<i>task-id</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:task/<i>cluster-name</i>/<i>task-id</i></code>

Resource ID length

A resource ID takes the form of a unique combination of letters and numbers. New resource ID formats include shorter IDs for Amazon ECS tasks and container instances. The current resource ID format is 36 characters long. The new IDs are in a 32-character format that doesn't include any hyphens. For information about opting in to the new resource ID format, see [Modifying account settings \(p. 249\)](#).

ARN and resource ID format timeline

The timeline for the opt-in and opt-out periods for the new Amazon Resource Name (ARN) and resource ID format for Amazon ECS resources ended on April 1, 2021. By default, all accounts are opted in to the new format. All new resources created receive the new format, and you can no longer opt out.

AWS Fargate Federal Information Processing Standard (FIPS-140) compliance

You must turn on Federal Information Processing Standard (FIPS-140) compliance on Fargate. For more information, see [the section called "AWS Fargate FIPS-140 compliance" \(p. 657\)](#).

Run `put-account-setting-default` with the `fargateFIPSMode` option set to enabled. For more information, see, [put-account-setting-default](#) in the *Amazon Elastic Container Service API Reference*.

- Example to turn on FIPS-140 compliance

```
aws ecs put-account-setting-default --name fargateFIPSMode --value enabled
```

Output

```
{
    "setting": {
        "name": "fargateFIPSMode",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:user"
    }
}
```

You can run `list-account-settings` to view the current FIPS-140 compliance status. Use the `effective-settings` option to view the account level settings.

```
aws ecs list-account-settings --effective-settings
```

Tagging authorization

Amazon ECS is introducing tagging authorization for resource creation. Users must have permissions for actions that create the resource, such as `ecsCreateCluster`. When you create a resource and specify tags for that resource, AWS performs additional authorization to verify that there are permissions to create tags. Therefore, you must grant explicit permissions to use the `ecs:TagResource` action. For more information, see [the section called “Tag resources during creation” \(p. 651\)](#).

In order to opt in to tagging authorization, run `put-account-setting-default` with the `tagResourceAuthorization` option set to enable. For more information, see, [put-account-setting-default](#) in the *Amazon Elastic Container Service API Reference*. You can run `list-account-settings` to view the current tagging authorization status.

- Example to turn on tagging authorization

```
aws ecs put-account-setting-default --name tagResourceAuthorization --value on --region region
```

Output

```
{  
  "setting": {  
    "name": "tagResourceAuthorization",  
    "value": "on",  
    "principalArn": "arn:aws:iam::123456789012:user"  
  }  
}
```

After you opt in, you must configure the appropriate permissions to allow users to tag resources on creation. For more information, see [the section called “Tag resources during creation” \(p. 651\)](#).

You can run `list-account-settings` to view the current tagging authorization status. Use the `effective-settings` option to view the account level settings.

```
aws ecs list-account-settings --effective-settings
```

Tagging authorization timeline

You can confirm whether tagging authorization is active by running `list-account-settings` to view the `tagResourceAuthorization` value. When the value is on, it means that the tagging authorization is in use. For more information, see, [list-account-settings](#) in the *Amazon Elastic Container Service API Reference*.

The following are the important dates related to tagging authorization.

- April 18, 2023 – Tagging authorization is introduced. All new and existing accounts must opt in to use the feature. You can opt in to using tagging authorization. By opting in, you must grant the appropriate permissions.

Viewing account settings using the console

You can use the AWS Management Console to view your account settings.

Important

The `dualStackIPv6` and the `fargateFIPSMODE` account settings can only be viewed or changed using the AWS CLI.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.

Modifying account settings

You can use the AWS Management Console to modify your account settings.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. To increase or decrease the number of tasks that you can run in the `awsvpc` network mode for each EC2 instance, under **AWSVPC Trunking**, select **AWSVPC Trunking**.
6. To use or stop using CloudWatch Container Insights by default for clusters, under **CloudWatch Container Insights**, select or clear **CloudWatch Container Insights**.
7. To opt in or out of tagging authorization, under **Resource Tagging Authorization**, select or clear **Resource Tagging Authorization**.
8. Choose **Save changes**.
9. On the confirmation screen, choose **Confirm** to save the selection.

Reverting to the default Amazon ECS account settings

You can use the AWS Management Console to revert your Amazon ECS account settings to the default.

The **Revert to account default** option is only available when your account settings are no longer the default settings.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. Choose **Revert to account default**.
6. On the confirmation screen, choose **Confirm** to save the selection.

Account setting management using the AWS CLI

You can manage your account settings using the Amazon ECS API, AWS CLI or SDKs.

For information about the available API actions for task definitions see [Account setting actions](#) in the *Amazon Elastic Container Service API Reference*.

Use one of the following commands to modify the default account setting for all users or roles on your account. These changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves.

- [put-account-setting-default](#) (AWS CLI)

```
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --region us-east-2
```

You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Region us-east-1 -Force
```

To modify the account settings for your user account (AWS CLI)

Use one of the following commands to modify the account settings for your user. If you're using these commands as the root user, changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves.

- [put-account-setting](#) (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-east-1
```

You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
```

To modify the account settings for a specific user or role (AWS CLI)

Use one of the following commands and specify the ARN of a user, role, or root user in the request to modify the account settings for a specific user or role.

- [put-account-setting](#) (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --principal-arn arn:aws:iam::aws_account_id:user/principalName --region us-east-1
```

You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -PrincipalArn arn:aws:iam::aws_account_id:user/principalName -Region us-east-1 -Force
```

Amazon ECS container instances

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.

Note

Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

Amazon ECS supports the following container instance types.

- Linux
- Windows
- External, such as an on-premises VM

Container instance concepts

- Your container instance must be running the Amazon ECS container agent. The container agent is able to register the instance into one of your clusters. If you are using an Amazon ECS-optimized AMI, the agent is already installed. To use a different operating system, install the agent. For more information, see [Amazon ECS container agent \(p. 356\)](#).
- Because the Amazon ECS container agent makes calls to Amazon ECS on your behalf, you must launch container instances with an IAM role that authenticates to your account and provides the required resource permissions. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
- Beginning with Linux Amazon ECS-optimized AMI version 20200430 and later, the Amazon EC2 Instance Metadata Service Version 2 (IMDSv2) is supported on your container instances. For Amazon ECS-optimized AMIs prior to version 20200430, Amazon EC2 Instance Metadata Service Version 1 (IMDSv1) is supported. For more information, see [Configuring the instance metadata service](#) in the *Amazon EC2 User Guide for Linux Instances*.
- If any of the containers associated with your tasks require external connectivity, you can map their network ports to ports on the host Amazon ECS container instance so they are reachable from the internet. Your container instance security group must allow inbound access to the ports you want to expose. For more information, see [Create a Security Group](#) in the [Amazon VPC Getting Started Guide](#).
- We strongly recommend launching your container instances inside a VPC, because Amazon VPC delivers more control over your network and offers more extensive configuration capabilities. For more information, see [Amazon EC2 and Amazon Virtual Private Cloud](#) in the *Amazon EC2 User Guide for Linux Instances*.
- Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 659\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 412\)](#) in this guide. For more information, see [the section called “Create a virtual private cloud” \(p. 12\)](#).

- The type of Amazon EC2 instance that you choose for your container instances determines the resources available in your cluster. Amazon EC2 provides different instance types, each with different

CPU, memory, storage, and networking capacity that you can use to run your tasks. For more information, see [Amazon EC2 Instances](#).

- Because each container instance has unique state information that is stored locally on the container instance and within Amazon ECS:
 - You should not deregister an instance from one cluster and re-register it into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances with the latest Amazon ECS-optimized Amazon Linux 2 AMI in the new cluster. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS Linux container instance \(p. 272\)](#).
 - You cannot stop a container instance and change its instance type. Instead, we recommend that you terminate the container instance and launch a new container instance with the desired instance size and the latest Amazon ECS-optimized Amazon Linux 2 AMI in your desired cluster. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS Linux container instance \(p. 272\)](#) in this guide.

Container instance lifecycle

When the Amazon ECS container agent registers an Amazon EC2 instance into your cluster, the Amazon EC2 instance reports its status as ACTIVE and its agent connection status as TRUE. This container instance can accept RunTask requests.

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

Important

If you stop and start a container instance, or reboot that instance, some older versions of the Amazon ECS container agent register the instance again without deregistering the original container instance ID. In this case, Amazon ECS lists more container instances in your cluster than you actually have. (If you have duplicate container instance IDs for the same Amazon EC2 instance ID, you can safely deregister the duplicates that are listed as ACTIVE with an agent connection status of FALSE.) This issue is fixed in the current version of the Amazon ECS container agent. For more information about updating to the current version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

If you change the status of a container instance to DRAINING, new tasks are not placed on the container instance. Any service tasks running on the container instance are removed, if possible, so that you can perform system updates. For more information, see [Container instance draining \(p. 354\)](#).

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

Important

You can drain the instances manually, or build an Auto Scaling group lifecycle hook to set the instance status to DRAINING. See [Amazon EC2 Auto Scaling lifecycle hooks](#) for more information about Auto Scaling lifecycle hooks.

Choosing the Amazon EC2 instance type

If you use Amazon EC2 to provide capacity for your cluster, you can choose from a large selection of instance types. All Amazon EC2 instance types and families are compatible with Amazon ECS.

To determine which instance types you can use, start by eliminating the instance types or instance families that don't meet the specific requirements of your application. For example, if your application requires a GPU, you can exclude any instance types that don't have a GPU. However, you should also consider other requirements, too. For example, consider the CPU architecture, network throughput, and if instance storage is a requirement. Next, examine the amount of CPU and memory provided by each instance type. As a general rule, the CPU and memory must be large enough to hold at least one replica of the task that you want to run.

You can choose from the instance types that are compatible with your application. With larger instances, you can launch more tasks at the same time. And, with smaller instances, you can scale out in a more fine-grained way to save costs. You don't need to choose a single Amazon EC2 instance type that to fit all the applications in your cluster. Instead, you can create multiple Auto Scaling Groups,. Each group can have a different instance type. Then, you can create an Amazon EC2 Capacity Provider for each one of these groups. Last, in the Capacity Provider strategy of your service and task, you can select the Capacity Provider that best suits its needs. For more information, see [Instance types](#) in the *Amazon EC2 User Guide for Linux Instances*.

Using Amazon EC2 Spot

Spot capacity can provide significant cost savings over on-demand instances. Spot capacity is excess capacity that's priced significantly lower than on-demand or reserved capacity. Spot capacity is suitable for batch processing and machine-learning workloads, and development and staging environments. More generally, it's suitable for any workload that tolerates temporary downtime.

Understand that the following consequences because Spot capacity might not be available all the time.

- During periods of extremely high demand, Spot capacity might be unavailable. This can cause Amazon EC2 Spot instance launches to be delayed. In these events, Amazon ECS services retry launching tasks, and Amazon EC2 Auto Scaling groups also retry launching instances, until the required capacity becomes available. Amazon EC2 doesn't replace Spot capacity with on-demand capacity.
- When the overall demand for capacity increases, Spot instances and tasks might be terminated with only a two-minute warning. After the warning is sent, tasks should begin an orderly shutdown if necessary before the instance is fully terminated. This helps minimize the possibility of errors. For more information about a graceful shutdown, see [Graceful shutdowns with ECS](#) .

To help minimize Spot capacity shortages, consider the following recommendations:

- Use multiple Regions and Availability Zones - Spot capacity varies by Region and Availability Zone. You can improve Spot availability by running your workloads in multiple Regions and Availability Zones. If possible, specify subnets in all the Availability Zones in the Regions where you run your tasks and instances.
- Use multiple Amazon EC2 instance types - When you use Mixed Instance Policies with Amazon EC2 Auto Scaling, multiple instance types are launched into your Auto Scaling Group. This ensures that a request for Spot capacity can be fulfilled when needed. To maximize reliability and minimize complexity, use instance types with roughly the same amount of CPU and memory in your Mixed Instances Policy. These instances can be from a different generation, or variants of the same base instance type. Note that they might come with additional features that you might not require. An example of such a list could include m4.large, m5.large, m5a.large, m5d.large, m5n.large, m5dn.large, and m5ad.large. For more information, see [Auto Scaling groups with multiple instance types and purchase options](#) in the *Amazon EC2 Auto Scaling User Guide*.
- Use the capacity-optimized Spot allocation strategy - With Amazon EC2 Spot, you can choose between the capacity- and cost-optimized allocation strategies. If you choose the capacity-optimized strategy when launching a new instance, Amazon EC2 Spot selects the instance type with the greatest availability in the selected Availability Zone. This helps reduce the possibility that the instance is terminated soon after it launches.

Linux instances

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.

Note

Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

The following Linux container instance operating systems are available:

- Amazon Linux: This is a general purpose operating system.
- Bottlerocket: This is an operating system that is optimized for container workloads and that has a focus on security. It does not include a package manager and is immutable by default. For information about the security features and guidance, see [Security Features](#) and [Security Guidance](#) on the GitHub website.

An Amazon ECS container instance specification consists of the following components:

Required

- A modern Linux distribution running at least version 3.10 of the Linux kernel.
- The Amazon ECS container agent (preferably the latest version). For more information, see [Amazon ECS container agent \(p. 356\)](#).
- A Docker daemon running at least version 1.9.0, and any Docker runtime dependencies. For more information, see [Check runtime dependencies](#) in the Docker documentation.

Note

For the best experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS container agent version that you are using.

Recommended

- An initialization and nanny process to run and monitor the Amazon ECS container agent. The Amazon ECS-optimized AMIs use the `ecs-init` RPM to manage the agent. For more information, see the [ecs-init project](#) on GitHub.

Topics

- [Amazon ECS-optimized AMI \(p. 255\)](#)
- [Amazon ECS-optimized Bottlerocket AMIs \(p. 268\)](#)
- [Launching an Amazon ECS Linux container instance \(p. 272\)](#)
- [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#)
- [Starting a task at container instance launch time \(p. 281\)](#)
- [Elastic network interface trunking \(p. 283\)](#)
- [Container Instance Memory Management \(p. 304\)](#)
- [Manage container instances remotely using AWS Systems Manager \(p. 306\)](#)

Amazon ECS-optimized AMI

Amazon ECS provides the Amazon ECS-optimized AMIs that are preconfigured with the requirements and recommendations to run your container workloads on Amazon ECS Linux instances. We recommend that you use the Amazon ECS-optimized Amazon Linux 2 AMI for your Amazon EC2 instances unless your application requires a specific operating system or a Docker version that is not yet available in that AMI.

Although you can create your own Amazon EC2 instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly.

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and Amazon ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 259\)](#).

The following variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances.

Operating system	AMI	Description
Amazon Linux 2	Amazon ECS-optimized Amazon Linux 2 AMI	For most cases, recommended for launching your Amazon EC2 instances for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 AMI does not come with the AWS CLI preinstalled.
	Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 instances and you want to use Linux kernel 5.10 instead of kernel 4.14 for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI does not come with the AWS CLI preinstalled.
Amazon Linux 2 (arm64)	Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2 Processors, for your Amazon ECS workloads. For more information, see General Purpose Instances in the Amazon EC2 User Guide for Linux Instances . The Amazon ECS-optimized Amazon Linux 2 (arm64) AMI does not come with the AWS CLI preinstalled.

Operating system	AMI	Description
	Amazon ECS-optimized Amazon Linux 2 kernel 5.10 (arm64) AMI	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2 Processors, and you want to use Linux kernel 5.10 instead of Linux kernel 4.14 for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 kernel 5.10 (arm64) AMI does not come with the AWS CLI preinstalled.
Amazon Linux 2 (GPU)	Amazon ECS GPU-optimized AMI	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 GPU-based instances for your Amazon ECS workloads. It comes pre-configured with NVIDIA kernel drivers and a Docker GPU runtime which makes running workloads that take advantage of GPUs on Amazon ECS. For more information, see Working with GPUs on Amazon ECS (p. 142) .
Amazon Linux 2 (Neuron)	Amazon ECS optimized Amazon Linux 2 (Neuron) AMI	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 Inf1, Trn1 or Inf2 instances. It comes pre-configured with AWS Inferentia and AWS Trainium drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see Using AWS Neuron on Amazon Linux 2 on Amazon ECS (p. 154) . The Amazon ECS optimized Amazon Linux 2 (Neuron) AMI does not come with the AWS CLI preinstalled.

Operating system	AMI	Description
Amazon Linux 2023	Amazon ECS-optimized Amazon Linux 2023 AMI	Amazon Linux 2023 is the next generation of Amazon Linux from AWS. For most cases, recommended for launching your Amazon EC2 instances for your Amazon ECS workloads. For more information, see What is Amazon Linux 2023 in the Amazon Linux 2023 User Guide .
Amazon Linux 2023 (arm64)	Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI	<p>Based on Amazon Linux 2023 this AMI is recommended for use when launching your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2 Processors, for your Amazon ECS workloads. For more information, see General Purpose Instances in the Amazon EC2 User Guide for Linux Instances.</p> <p>The Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI does not come with the AWS CLI preinstalled.</p>
Amazon Linux 2023 (Neuron)	Amazon ECS optimized Amazon Linux 2023 (Neuron) AMI	<p>Based on Amazon Linux 2023, this AMI is recommended for use when launching your Amazon EC2 Inf1 instances. It comes pre-configured with AWS Inferentia drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see Using AWS Neuron on Amazon Linux 2 on Amazon ECS (p. 154). The Amazon ECS optimized Amazon Linux 2023 (Neuron) AMI does not come with the AWS CLI preinstalled.</p>

Operating system	AMI	Description
Amazon Linux	Amazon ECS-optimized Amazon Linux AMI	<p>This AMI is based off of Amazon Linux. We recommend that you migrate your workloads to the Amazon ECS-optimized Amazon Linux 2 AMI. Support for the Amazon ECS-optimized Amazon Linux AMI is the same as the Amazon Linux AMI. For more information, see Amazon Linux AMI.</p> <p>Important On April 15, 2021, the Amazon ECS-optimized Amazon Linux AMI ended its standard support phase and entered a maintenance support phase. In the maintenance support phase, Amazon ECS will continue providing critical and important security updates for a reduced list of packages. During this period, Amazon ECS will no longer add support for new EC2 instance types, new services and features, and new packages. Instead, Amazon ECS will provide updates only for critical and important security fixes that apply to a reduced set of packages. Maintenance support period will end on June 30, 2023.</p>

Amazon ECS-optimized AMI changelog

Amazon ECS provides a changelog for the Linux variant of the Amazon ECS-optimized AMI on GitHub. For more information, see [Changelog](#).

The Linux variants of the Amazon ECS-optimized AMI use the Amazon Linux 2 AMI or Amazon Linux 2023 AMI as their base. You can retrieve the Amazon Linux 2 source AMI name or the Amazon Linux 2023 AMI name for each variant by querying the Systems Manager Parameter Store API. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 259\)](#). The Amazon Linux 2 AMI release notes are available as well. For more information, see [Amazon Linux 2 release notes](#). The Amazon Linux 2023 release notes are available as well. For more information see, [Amazon Linux 2023 release notes](#).

The following pages provide additional information about the changes:

- [Source AMI release notes on GitHub](#)
- [Docker Engine release notes](#) in the Docker documentation
- [NVIDIA Driver Documentation](#) in the NVIDIA documentation

Retrieving Amazon ECS-Optimized AMI metadata

The AMI ID, image name, operating system, container agent version, source image name, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see [GetParameters](#) and [GetParametersByPath](#).

Note

Your administrative user must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.

- `ssm:GetParameters`
- `ssm:GetParameter`
- `ssm:GetParametersByPath`

Systems Manager Parameter Store parameter format

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

Linux Amazon ECS-optimized AMIs

- Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<version>
```

- Amazon Linux 2 kernel 5.10 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/<version>
```

- Amazon Linux 2 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>
```

- Amazon Linux 2 kernel 5.10 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/arm64/<version>
```

- Amazon Linux 2 (GPU) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>
```

- Amazon Linux 2 (Neuron) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/inf/<version>
```

- Amazon Linux 2023 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/<version>
```

- Amazon Linux 2023 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/arm64/<version>
```

- Amazon Linux 2023 (Neuron) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/inf/<version>
```

- Amazon Linux AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux/<version>
```

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

The following parameter name format retrieves the image ID of the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by using the sub-parameter `image_id`.

```
/aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

The following parameter name format retrieves the metadata of a specific Amazon ECS-optimized AMI version by specifying the AMI name.

- Amazon ECS-optimized Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20181112-x86_64-ebs
```

Note

All versions of the Amazon ECS-optimized Amazon Linux 2 AMI are available for retrieval. Only Amazon ECS-optimized AMI versions `amzn-ami-2017.09.1-amazon-ecs-optimized` (Linux) and later can be retrieved.

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest stable Amazon ECS-optimized AMI

You can retrieve the latest stable Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

Linux Amazon ECS-optimized AMIs

- **For the Amazon ECS-optimized Amazon Linux 2 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/recommended --region us-east-1
```

- For the Amazon ECS-optimized Amazon Linux 2 kernel 5.10 (arm64) AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/arm64/recommended --region us-east-1
```

- For the Amazon ECS GPU-optimized AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended --region us-east-1
```

- For the Amazon ECS optimized Amazon Linux 2 (Neuron) AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/recommended --region us-east-1
```

- For the Amazon ECS-optimized Amazon Linux 2023 AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/recommended --region us-east-1
```

- For the Amazon ECS-optimized Amazon Linux 2023 (arm64) AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/arm64/recommended --region us-east-1
```

- For the Amazon ECS-optimized Amazon Linux AMIs:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux/recommended --region us-east-1
```

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

[Retrieving the metadata of a specific Amazon ECS-optimized Amazon Linux 2 AMI version](#)

Retrieve the metadata of a specific Amazon ECS-optimized Amazon Linux AMI version using the AWS CLI with the following AWS CLI command. Replace the AMI name with the name of the Amazon ECS-optimized Amazon Linux AMI to retrieve.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20200928-x86_64-ebs --region us-east-1
```

[Retrieving the Amazon ECS-optimized Amazon Linux 2 AMI metadata using the Systems Manager GetParametersByPath API](#)

Retrieve the Amazon ECS-optimized Amazon Linux 2 AMI metadata with the Systems Manager GetParametersByPath API using the AWS CLI with the following command.

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/ --region us-east-1
```

Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 AMI ID by using the sub-parameter `image_id`.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended/  
image_id --region us-east-1
```

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/  
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2023 AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2023 AMI ID by using the sub-parameter `image_id`.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-  
linux-2023/recommended/image_id --region us-east-1
```

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/  
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

Linux example

```
Parameters:  
LatestECSSOptimizedAMI:  
  Description: AMI ID  
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>  
  Default: /aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

AMI storage configuration

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

The following describes the storage configuration for each of the Amazon ECS-optimized AMIs.

Topics

- [Amazon Linux 2 storage configuration \(p. 263\)](#)
- [Amazon Linux 2023 storage configuration \(p. 263\)](#)
- [Amazon ECS-optimized Amazon Linux AMI storage configuration \(p. 263\)](#)

Amazon Linux 2 storage configuration

By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata.

The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the OverlayFS storage driver](#) in the Docker documentation.

Amazon Linux 2023 storage configuration

By default, the Amazon ECS-optimized Amazon Linux 2023 AMI ships with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata.

The default filesystem for the Amazon ECS-optimized Amazon Linux 2023 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the OverlayFS storage driver](#) in the Docker documentation.

Amazon ECS-optimized Amazon Linux AMI storage configuration

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

By default, the Amazon ECS-optimized Amazon Linux AMI ships with 30 GiB of total storage. You can modify this value at launch time to increase the available storage on your Amazon EC2 instance. This storage is used for the operating system and for Docker images and metadata. The sections below describe the storage configuration of the Amazon ECS-optimized Amazon Linux AMI, based on the AMI version.

Version 2015.09.d and later

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

Note

You can increase these default volume sizes by changing the block device mapping settings for your instances when you launch them; however, you cannot specify a smaller volume size than the default. For more information, see [Block Device Mapping](#) in the *Amazon EC2 User Guide for Linux Instances*.

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

Note

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

To determine the available storage for Docker

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

Note

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

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

```
[ec2-user ~]$ sudo vgs
```

Output:

VG	#PV	#LV	#SN	Attr	VSize	VFree
docker	1	1	0	wz--n-	22.00g	204.00m

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

```
[ec2-user@ ~]$ sudo lvs
```

Output:

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync
Convert											
docker-pool	docker	twi-aot---	21.75g			7.63	4.96				

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

Note

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

```
[ec2-user ~]$ docker info | grep "Data Space"
```

Output:

Data Space Used: 1.782 GB
Data Space Total: 23.35 GB
Data Space Available: 21.57 GB

To extend the Docker logical volume

The easiest way to add storage to your Amazon EC2 instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you are unable to do this, you can add storage to the volume group that Docker uses and extend its logical volume by following these steps.

Note

If your Amazon EC2 instance storage is filling up too quickly, there are a few actions that you can take to reduce this effect:

- (Amazon ECS container agent 1.8.0 and later) Reduce the amount of time that stopped or exited containers remain on your container instances. The `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent configuration variable sets the time duration to wait from when a task is stopped until the Docker container is removed (by default, this value is 3 hours). This removes the Docker container data. If this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).
- Remove non-running containers and unused images from your Amazon EC2 instances. You can use the following example commands to manually remove stopped containers and unused images. Deleted containers cannot be inspected later, and deleted images must be pulled again before starting new containers from them.

To remove non-running containers, execute the following command on your Amazon EC2 instance:

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

To remove unused images, execute the following command on your Amazon EC2 instance:

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

- Remove unused data blocks within containers. You can use the following command to run `fstrim` on any running container and discard any data blocks that are unused by the container file system.

```
$ sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ fstrim /proc/Z/root/"
```

1. Create a new Amazon EBS volume in the same Availability Zone as your Amazon EC2 instance. For more information, see [Creating an Amazon EBS Volume](#) in the *Amazon EC2 User Guide for Linux Instances*.
2. Attach the volume to your Amazon EC2 instance. The default location for the Docker data volume is `/dev/xvdcz`. For consistency, attach additional volumes in reverse alphabetical order from that device name (for example, `/dev/xvdcy`). For more information, see [Attaching an Amazon EBS Volume to an Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
3. Connect to your Amazon EC2 instance using SSH..
4. Check the size of your `docker-pool` logical volume. The example below shows a logical volume of 409.19 GiB.

```
[ec2-user ~]$ sudo lvs
```

Output:

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%Sync
Convert	docker-pool	docker twi-aot---	409.19g			0.16	0.08			

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

```
[ec2-user ~]$ sudo vgs
```

Output:

VG	#PV	#LV	#SN	Attr	VSize	VFree
docker	1	1	0	wz--n-	1024.00g	612.75g

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

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

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

```
[ec2-user ~]$ sudo vgs
```

Output:

VG	#PV	#LV	#SN	Attr	VSize	VFree
docker	2	1	0	wz--n-	2.00t	1.60t

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

```
[ec2-user ~]$ sudo lvextend -L+1024G /dev/docker/docker-pool
```

Output:

Size of logical volume docker/docker-pool_tdata changed from 409.19 GiB (104752 extents) to 1.40 TiB (366896 extents).
Logical volume docker-pool successfully resized

- Verify that your logical volume has increased in size.

```
[ec2-user ~]$ sudo lvs
```

Output:

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%Sync
Convert	docker-pool	docker twi-aot---	1.40t			0.04	0.12			

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

Note

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

```
[ec2-user ~]$ docker info | grep "Data Space"
```

Output:

```
Data Space Used: 109.6 MB
Data Space Total: 1.539 TB
Data Space Available: 1.539 TB
```

Version 2015.09.c and earlier

Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.c and earlier launch with a single 30-GiB volume that is attached at /dev/xvda and mounted as the root of the file system. This volume shares the operating system and all Docker images and metadata. You can determine the available storage on your Amazon EC2 instance with standard storage information commands (such as **df -h**).

There is no practical way to add storage (that Docker can use) to instances launched from these AMIs without stopping them. If you find that your Amazon EC2 instances need more storage than the default 30 GiB, you should terminate each instance. Then, launch another in its place with the latest Amazon ECS-optimized Amazon Linux AMI and a large enough data storage volume.

Amazon ECS-optimized Linux AMI build script

Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. These build scripts are now available on GitHub. For more information, see [amazon-ecs-ami](#) on GitHub.

The build scripts repository includes a [HashiCorp packer](#) template and build scripts to generate each of the Linux variants of the Amazon ECS-optimized AMI. These scripts are the source of truth for Amazon ECS-optimized AMI builds, so you can follow the GitHub repository to monitor changes to our AMIs. For example, perhaps you want your own AMI to use the same version of Docker that the Amazon ECS team uses for the official AMI.

For more information, see the Amazon ECS AMI repository at [aws/amazon-ecs-ami](#) on GitHub.

To build an Amazon ECS-optimized Linux AMI

1. Clone the [aws/amazon-ecs-ami](#) GitHub repo.

```
git clone https://github.com/aws/amazon-ecs-ami.git
```

2. Add an environment variable for the AWS Region to use when creating the AMI. Replace the `us-west-2` value with the Region to use.

```
export REGION=us-west-2
```

3. A Makefile is provided to build the AMI. From the root directory of the cloned repository, use one of the following commands, corresponding to the Linux variant of the Amazon ECS-optimized AMI you want to build.

- Amazon ECS-optimized Amazon Linux 2 AMI

```
make al2
```

- Amazon ECS-optimized Amazon Linux 2 (arm64) AMI

```
make al2arm
```

- Amazon ECS GPU-optimized AMI

```
make al2gpu
```

- Amazon ECS optimized Amazon Linux 2 (Neuron) AMI

```
make al2inf
```

- Amazon ECS-optimized Amazon Linux 2023 AMI

```
make al2023
```

- Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI

```
make al2023arm
```

- Amazon ECS optimized Amazon Linux 2023 (Neuron) AMI

```
make al2023neu
```

- Amazon ECS-optimized Amazon Linux AMI

```
make al1
```

Important

On April 15, 2021, the Amazon ECS-optimized Amazon Linux AMI ended its standard support phase and entered a maintenance support phase. In the maintenance support phase, Amazon ECS will continue providing critical and important security updates for a reduced list of packages. During this period, Amazon ECS will no longer add support for new EC2 instance types, new services and features, and new packages. Instead, Amazon ECS will provide updates only for critical and important security fixes that apply to a reduced set of packages. Maintenance support period will end on June 30, 2023.

Amazon ECS-optimized Bottlerocket AMIs

The Amazon ECS-optimized Bottlerocket AMI is built on top of [Bottlerocket](#). Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. The Amazon ECS-optimized Bottlerocket AMI is secure and only includes the minimum number of packages that's required to run containers. This improves resource usage, reduces security attack surface, and helps lower management overhead. The Bottlerocket AMI is also integrated with Amazon ECS to help reduce the operational overhead involved in updating container instances in a cluster.

Bottlerocket differs from Amazon Linux in the following ways:

- Bottlerocket doesn't include a package manager, and its software can only be run as containers. Updates to Bottlerocket are both applied and can be rolled back in a single step, which reduces the likelihood of update errors.

- The primary mechanism to manage Bottlerocket hosts is with a container scheduler. Unlike Amazon Linux, logging into individual Bottlerocket instances is intended to be an infrequent operation for advanced debugging and troubleshooting purposes only.

For more information about Bottlerocket, see the [documentation](#) and [releases](#) on GitHub.

An Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI that you can use when launching Amazon ECS container instances.

Amazon ECS-optimized Bottlerocket AMI variants

You can use the following Amazon ECS-optimized Bottlerocket AMI variants for your Amazon EC2 instances:

- `aws-ecs-1`
- `aws-ecs-1-nvidia`

For more information about the `aws-ecs-1-nvidia` variant, see [Announcing NVIDIA GPU support for Bottlerocket on Amazon ECS](#).

Considerations

Consider the following when using a Bottlerocket AMI with Amazon ECS.

- You can deploy to Amazon EC2 instances with x86 or Arm64 processors, but not to instances that have Inferentia chips.
- Bottlerocket images don't include an SSH server or a shell. However, you can use out-of-band management tools to gain SSH administrator access and perform bootstrapping. For more information, see these sections in the [bottlerocket README.md](#) on GitHub:
 - [Exploration](#)
 - [Admin container](#)
- By default, Bottlerocket has a [control container](#) that's enabled. This container runs the [AWS Systems Manager agent](#) that you can use to run commands or start shell sessions on Amazon EC2 Bottlerocket instances. For more information, see [Setting up Session Manager](#) in the [AWS Systems Manager User Guide](#).
- Bottlerocket is optimized for container workloads and has a focus on security. Bottlerocket doesn't include a package manager and is immutable. For information about the security features and guidance, see [Security Features](#) and [Security Guidance](#) on GitHub.
- The `awsvpc` network mode is supported for Bottlerocket AMI version 1.1.0 or later.
- The Bottlerocket AMIs don't support the `initProcessEnabled` task definition parameter.
- The Bottlerocket AMIs also don't support the following services and features:
 - App Mesh in task definitions
 - ECS Anywhere
 - ECS Service Connect
 - Amazon EFS in encrypted mode and `awsvpc` network mode
 - Elastic Inference Accelerator

Retrieving an Amazon ECS-optimized Bottlerocket AMI

You can retrieve the Amazon Machine Image (AMI) ID for Amazon ECS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. Using this parameter, you don't need to manually look up Amazon ECS-optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see [GetParameter](#). The user that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon ECS-optimized AMI metadata.

Retrieving the aws-ecs-1 Bottlerocket AMI variant

You can retrieve the latest stable aws-ecs-1 Bottlerocket AMI variant by AWS Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter `image_id`. Replace the `region` with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub. To retrieve a version other than the latest, replace `latest` with the version number.

- For the 64-bit (x86_64) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1/x86_64/latest/image_id" --query Parameter.Value --output text
```

- For the 64-bit Arm (arm64) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1/arm64/latest/image_id" --query Parameter.Value --output text
```

- **AWS Management Console** – You can query for the recommended Amazon ECS-optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace `region` with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub.

- For the 64-bit (x86_64) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1/x86\_64/latest/image\_id/description?region=region#
```

- For the 64-bit Arm (arm64) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1/arm64/latest/image\_id/description?region=region#
```

Retrieving the aws-ecs-1-nvidia Bottlerocket AMI variant

You can retrieve the latest stable aws-ecs-1-nvidia Bottlerocket AMI variant by Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter `image_id`. Replace the `region` with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub. To retrieve a version other than the latest, replace `latest` with the version number.

- For the 64-bit (x86_64) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
```

- For the 64 bit Arm (`arm64`) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
```

- **AWS Management Console** – You can query for the recommended Amazon ECS optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace `region` with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub.

- For the 64 bit (`x86_64`) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1-nvidia/x86\_64/latest/image\_id/description?region=region#
```

- For the 64 bit Arm (`arm64`) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1-nvidia/x86\_64/latest/image\_id/description?region=region#
```

Launch a Bottlerocket container instance

You can use the AWS CLI to launch the container instance.

1. Create a file that's called `userdata.toml`. This file is used for the instance user data. Replace `cluster-name` with the name of your cluster.

```
[settings.ecs]
cluster = "cluster-name"
```

2. Use one of the commands that are included in [the section called "Retrieving an Amazon ECS-optimized Bottlerocket AMI" \(p. 270\)](#) to get the Bottlerocket AMI ID. You use this in the following step.
3. Run the following command to launch the Bottlerocket instance. Remember to replace the following parameters:
 - Replace `subnet` with the ID of the private or public subnet that your instance will launch in.
 - Replace `bottlerocket_ami` with the AMI ID from the previous step.
 - Replace `t3.large` with the instance type that you want to use.
 - Replace `region` with the Region code.

```
aws ec2 run-instances --key-name ecs-bottlerocket-example \
--subnet-id subnet \
--image-id bottlerocket_ami \
--instance-type t3.large \
--region region \
--tag-specifications 'ResourceType=instance,Tags=[{Key=bottlerocket,Value=example}]' \
 \
--user-data file://userdata.toml \
--iam-instance-profile Name=ecsInstanceRole
```

4. Run the following command to verify that the container instance is registered to the cluster. When you run this command, remember to replace the following parameters:
 - Replace *cluster* with your cluster name.
 - Replace *region* with your Region code.

```
aws ecs list-container-instances --cluster cluster-name --region region
```

For a detailed walkthrough of how to get started with the Bottlerocket operating system on Amazon ECS, see [Using a Bottlerocket AMI with Amazon ECS](#) on GitHub and [Getting started with Bottlerocket and Amazon ECS](#) on the AWS blog site.

Launching an Amazon ECS Linux container instance

Your Amazon ECS container instances are created using the Amazon EC2 console. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#).

You can launch an instance by various methods including the Amazon EC2 console, AWS CLI, and SDK. The procedure on this page covers the launch wizard in the Amazon EC2 console. For information about the other methods for launching an instance, see [Launch your instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

For more information about the launch wizard, see [Launch an instance using the new launch instance wizard](#) in the *Amazon EC2 User Guide for Linux Instances*.

New Amazon EC2 launch instance wizard

You can use the new Amazon EC2 wizard to launch an instance. The launch instance wizard specifies the launch parameters that are required for launching an instance. You can use the following list for the parameters and leave the parameters not listed as the default. The following instructions take you through each parameter group.

Parameters for instance configuration

- [Initiate instance launch \(p. 272\)](#)
- [Name and tags \(p. 273\)](#)
- [Application and OS Images \(Amazon Machine Image\) \(p. 273\)](#)
- [Instance type \(p. 273\)](#)
- [Key pair \(login\) \(p. 273\)](#)
- [Network settings \(p. 274\)](#)
- [Configure storage \(p. 274\)](#)
- [Advanced details \(p. 275\)](#)

Initiate instance launch

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation bar at the top of the screen, the current AWS Region is displayed (for example, US East (Ohio)). Select a Region in which to launch the instance.
3. From the Amazon EC2 console dashboard, choose **Launch instance**.

If you see the old launch wizard, the new launch instance wizard is not yet the default view in the currently selected Region. To open the new launch instance wizard, choose **Opt-in to the new experience** at the top right of the screen. If you do not see **Opt-in to the new experience** at the top

right, the wizard is not available in your Region. You can launch an instance with the old wizard. For more information, see [the section called "Old Amazon EC2 launch instance wizard" \(p. 276\)](#).

Name and tags

The instance name is a tag, where the key is **Name**, and the value is the name that you specify. You can tag the instance, the volumes, and elastic graphics. For Spot Instances, you can tag the Spot Instance request only.

Specifying an instance name and additional tags is optional.

- For **Name**, enter a descriptive name for the instance. If you don't specify a name, the instance can be identified by its ID, which is automatically generated when you launch the instance.
- To add additional tags, choose **Add additional tags**. Choose **Add tag**, and then enter a key and value, and select the resource type to tag. Choose **Add tag** again for each additional tag to add.

Application and OS Images (Amazon Machine Image)

An Amazon Machine Image (AMI) contains the information required to create an instance. For example, an AMI might contain the software that's required to act as a web server, such as Apache, and your website.

Use the **Search** bar to find a suitable Amazon ECS-optimized AMI published by AWS.

1. Enter one of the following terms in the **Search** bar.
 - **ami-ecs**
 - The **Value** of an Amazon ECS-optimized AMI.

For the latest Amazon ECS-optimized AMIs and their values, see [Linux Amazon ECS-optimized AMI](#).

2. Press **Enter**.
3. On the **Choose an Amazon Machine Image (AMI)** page, select the **AWS Marketplace AMIs** tab.
4. From the left **Refine results** pane, select **Amazon Web Services** as the **Publisher**.
5. Choose **Select** on the row of the AMI that you want to use.

Alternatively, choose **Cancel** (at top right) to return to the launch instance wizard without choosing an AMI. A default AMI will be selected. Ensure that the AMI meets the requirements outlined in [Linux instances](#).

Instance type

The instance type defines the hardware configuration and size of the instance. Larger instance types have more CPU and memory. For more information, see [Instance types](#).

- For **Instance type**, select the instance type for the instance.

The instance type that you select determines the resources available for your tasks to run on.

Key pair (login)

For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one.

Important

If you choose the **Proceed without key pair (Not recommended)** option, you won't be able to connect to the instance unless you choose an AMI that is configured to allow users another way to log in.

Network settings

Configure the network settings, as necessary.

- **Networking platform:** Choose **Virtual Private Cloud (VPC)**, and then specify the subnet in the **Network interfaces** section.
- **VPC:** Select an existing VPC in which to create the security group.
- **Subnet:** You can launch an instance in a subnet associated with an Availability Zone, Local Zone, Wavelength Zone, or Outpost.

To launch the instance in an Availability Zone, select the subnet in which to launch your instance. To create a new subnet, choose **Create new subnet** to go to the Amazon VPC console. When you are done, return to the launch instance wizard and choose the Refresh icon to load your subnet in the list.

To launch the instance in a Local Zone, select a subnet that you created in the Local Zone.

To launch an instance in an Outpost, select a subnet in a VPC that you associated with the Outpost.

- **Auto-assign Public IP:** If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 659\)](#)

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 412\)](#) in this guide.

- **Firewall (security groups):** Use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored.
 - To select an existing security group, choose **Select existing security group**, and select the security group that you created in [Set up to use Amazon ECS \(p. 10\)](#).

Configure storage

The AMI you selected includes one or more volumes of storage, including the root volume. You can specify additional volumes to attach to the instance.

You can use the **Simple** view.

- **Storage type:** Configure the storage for your container instance.

If you are using the Amazon ECS-optimized Amazon Linux 2 AMI, your instance has a single 30 GiB volume configured, which is shared between the operating system and Docker.

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

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

Advanced details

For **Advanced details**, expand the section to view the fields and specify any additional parameters for the instance.

- **Purchasing option:** Choose **Request Spot Instances** to request Spot Instances. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a Not available message, you may need to choose a different instance type.

- **IAM instance profile:** Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- **(Optional) User data:** Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 370\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:

- By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

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

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

Note

For more information about this configuration, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#).

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

- Specify tags for your container instance using the `ECS_CONTAINER_INSTANCE_TAGS` configuration parameter. This creates tags that are associated with Amazon ECS only, they cannot be listed using the Amazon EC2 API.

Important

If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the `ECS_CONTAINER_INSTANCE_TAGS` agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_TAGS=[{"tag_key": "tag_value"}]
```

EOF

- Specify tags for your container instance and then use the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` configuration parameter to propagate them from Amazon EC2 to Amazon ECS

The following is an example of a user data script that would propagate the tags associated with a container instance, as well as register the container instance with a cluster named `your_cluster_name`:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#).

Old Amazon EC2 launch instance wizard

To launch a container instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select the Region to use.
3. From the **EC2 Dashboard**, choose **Launch instance**.
4. On the **Choose an Amazon Machine Image (AMI)** page, complete the following steps:
 - a. Choose **AWS Marketplace**.
 - b. Choose an AMI for your container instance. You can search for one of the Amazon ECS-optimized AMIs, for example the **Amazon ECS-Optimized Amazon Linux 2 AMI**. If you do not choose an Amazon ECS-optimized AMI, you must follow the procedures in [Installing the Amazon ECS container agent \(p. 356\)](#).

For more information on the latest Amazon ECS-optimized AMIs, see [Amazon ECS-optimized AMI \(p. 255\)](#).

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

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

Choose **Next: Configure Instance Details** when you are done.

6. On the **Configure Instance Details** page, complete the following steps:
 - a. Set the **Number of instances** field depending on how many container instances you want to add to your cluster.
 - b. (Optional) To use Spot Instances, for **Purchasing option**, select the check box next to **Request Spot Instances**. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

- c. For **Network**, choose the VPC into which to launch your container instance.
- d. For **Subnet**, choose a subnet to use, or keep the default option to choose the default subnet in any Availability Zone.
- e. Set the **Auto-assign Public IP** field depending on whether you want your instance to be accessible from the public internet. If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 659\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 412\)](#) in this guide. For more information, see [the section called “Create a virtual private cloud” \(p. 12\)](#).

- f. Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- g. (Optional) Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 370\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:

- By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

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

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

Note

For more information about this configuration, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#).

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

- Specify tags for your container instance using the `ECS_CONTAINER_INSTANCE_TAGS` configuration parameter. This creates tags that are associated with Amazon ECS only, they cannot be listed using the Amazon EC2 API.

Important

If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the `ECS_CONTAINER_INSTANCE_TAGS` agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Specify tags for your container instance and then use the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` configuration parameter to propagate them from Amazon EC2 to Amazon ECS

The following is an example of a user data script that would propagate the tags associated with a container instance, as well as register the container instance with a cluster named `your_cluster_name`:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#).

- h. Choose **Next: Add Storage**.
7. On the **Add Storage** page, configure the storage for your container instance.

If you are using the Amazon ECS-optimized Amazon Linux 2 AMI, your instance has a single 30 GiB volume configured, which is shared between the operating system and Docker.

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

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

When done configuring your volumes, choose **Next: Add Tags**.

8. On the **Add Tags** page, specify tags by providing key and value combinations for the container instance. Choose **Add another tag** to add more than one tag to your container instance. For more information resource tags, see [Resources and tags \(p. 529\)](#).

Choose **Next: Configure Security Group** when you are done.

9. On the **Configure Security Group** page, use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored. Select or create a security group as follows, and then choose **Review and Launch**.
10. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. Instead, select the security group that you created in [Set up to use Amazon ECS \(p. 10\)](#) using the following steps:
 - a. Choose **Edit security groups**.
 - b. On the **Configure Security Group** page, select the **Select an existing security group** option.
 - c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.
11. On the **Review Instance Launch** page, choose **Launch**.

12. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.
When you are ready, select the acknowledgment field, and then choose **Launch Instances**.
13. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
14. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. If the **Public DNS** column is hidden, choose **Show/Hide, Public DNS**.

Using Spot Instances

A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. Because Spot Instances enable you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly. The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and adjusted gradually based on the long-term supply of and demand for Spot Instances. For more information, see [Spot Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

You can register Spot Instances to your Amazon ECS clusters. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Spot Instance Draining

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance two-minute interruption notice for terminate and stop actions. It does not provide the two-minute notice for the hibernate action. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in DRAINING status.

Important

Amazon ECS does not receive a notice from Amazon EC2 when instances are removed by Auto Scaling Capacity Rebalancing. For more information, see [Amazon EC2 Auto Scaling Capacity Rebalancing](#).

When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the PENDING state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

Spot Instance draining is turned off by default and must be manually enabled. To enable Spot Instance draining for a new container instance, when launching the container instance add the following script into the **User data** field, replacing *MyCluster* with the name of the cluster to register the container instance to.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

To turn on Spot Instance draining for an existing container instance

1. Connect to the Spot Instance over SSH.
2. Edit the `/etc/ecs/ecs.config` file and add the following:

```
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
```

3. Restart the ecs service.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

4. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see [the section called "Container agent introspection" \(p. 410\)](#).

```
curl http://localhost:51678/v1/metadata
```

Bootstrapping container instances with Amazon EC2 user data

When you launch an Amazon EC2 instance, you have the option of passing user data to the instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud booothooks, shell scripts, and cloud-init directives. For more information about these and other format types, see the [Cloud-Init documentation](#).

You can pass this user data when using the Amazon EC2 launch wizard. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Topics

- [Amazon ECS container agent \(p. 280\)](#)
- [Docker daemon \(p. 281\)](#)

Amazon ECS container agent

The Linux variants of the Amazon ECS-optimized AMI look for agent configuration data in the /etc/ecs/ecs.config file when the container agent starts. You can specify this configuration data at launch with Amazon EC2 user data. For more information about available Amazon ECS container agent configuration variables, see [Amazon ECS container agent configuration \(p. 370\)](#).

To set only a single agent configuration variable, such as the cluster name, use **echo** to copy the variable to the configuration file:

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

If you have multiple variables to write to /etc/ecs/ecs.config, use the following heredoc format. This format writes everything between the lines beginning with **cat** and EOF to the configuration file.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username": "my_name", "password": "my_password", "email": "email@example.com"}}
ECS_LOGLEVEL=debug
EOF
```

To set custom instance attributes, set the ECS_INSTANCE_ATTRIBUTES environment variable.

```
#!/bin/bash cat <<'EOF' >> ecs.config
ECS_INSTANCE_ATTRIBUTES='{"envtype": "prod"}'
EOF
```

Docker daemon

You can specify Docker daemon configuration information with Amazon EC2 user data. For more information about configuration options, see [the Docker daemon documentation](#).

In the example below, the custom options are added to the Docker daemon configuration file, /etc/docker/daemon.json which is then specified in the user data when the instance is launched.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"debug": true}
EOF
systemctl restart docker --no-block
```

In the example below, the custom options are added to the Docker daemon configuration file, /etc/docker/daemon.json which is then specified in the user data when the instance is launched. This example shows how to turn off to turn off the docker-proxy in the Docker daemon config file.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"userland-proxy": false}
EOF
systemctl restart docker --no-block
```

Starting a task at container instance launch time

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

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

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

To start a task at container instance launch time

1. If you have not done so already, create a task definition with the container you want to run on your container instance at launch by following the procedures in [Creating a task definition using the console \(p. 125\)](#).
2. Modify your `ecsInstanceRole` IAM role to add permissions for the `StartTask` API operation. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
 - a. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Roles**.
 - c. Choose the `ecsInstanceRole`. If the role does not exist, use the procedure in [Amazon ECS container instance IAM role \(p. 638\)](#) to create the role and return to this procedure. If the role does exist, select the role to view the attached policies.
 - d. In the **Permissions** tab, choose **Add inline policy**.
 - e. For **Service**, choose **Choose a service, Elastic Container Service**.
 - f. For **Actions**, type **StartTask** in the search field, and then select **StartTask**.
 - g. For **Resources**, select **All resources**, and then choose **Review policy**.
 - h. On the **Review policy** page, enter a name for your policy, such as `ecs-start-task` and choose **Create policy**.
3. Launch one or more container instances using the Amazon ECS-optimized Amazon Linux 2 AMI by following the procedure in [Launching an Amazon ECS Linux container instance \(p. 272\)](#), but in [Step 6.g \(p. 277\)](#) copy and paste the MIME multi-part user data script below into the **User data** field. Substitute `your_cluster_name` with the cluster for the container instance to register into and `my_task_def` with the task definition to run on the instance at launch.

Note

The MIME multi-part content below uses a shell script to set configuration values and install packages. It also uses a systemd job to start the task after the `ecs` service is running and the introspection API is available.

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

----BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

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

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

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
exec 2>>/var/log/ecs/ecs-start-task.log
set -x

# Install prerequisite tools
yum install -y jq aws-cli

# Wait for the ECS service to be responsive
until curl -s http://localhost:51678/v1/metadata
do
    sleep 1
done

# Grab the container instance ARN and AWS Region from instance metadata
```

```

instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.\n  .ContainerInstanceArn' | awk -F/ '{print $NF}' )
cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk -F/
'{print $NF}' )
region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.\n  .ContainerInstanceArn' | awk -F: '{print $4}' )

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

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

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
    [Unit]
    Description=ECS Start Task
    Requires=ecs.service
    After=ecs.service

    [Service]
    Restart=on-failure
    RestartSec=30
    ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

    [Install]
    WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
# deadlock
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
----BOUNDARY----
```

4. Verify that your container instances launch into the correct cluster and that your tasks have started.
 - a. Open the console at <https://console.aws.amazon.com/ecs/v2>.
 - b. From the navigation bar, choose the Region that your cluster is in.
 - c. In the navigation pane, choose **Clusters** and select the cluster that hosts your container instances.
 - d. On the **Cluster** page, choose **Tasks**, and then choose your tasks.

Each container instance you launched should have your task running on it.

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

Elastic network interface trunking

Note

This feature is not available on Fargate.

Each Amazon ECS task that uses the awsvpc network mode receives its own elastic network interface (ENI), which is attached to the container instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a c5.large instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs

to the instance. Because each task using the awsvpc network mode requires an ENI, you can typically only run two such tasks on this instance type.

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and enable the awsvpcTrunking account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the awsvpc network mode on each container instance. Using this feature, a c5.1large instance with awsvpcTrunking enabled has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks.

The trunk network interface is fully managed by Amazon ECS and is deleted when you either terminate or deregister your container instance from the cluster. For more information, see [Task networking for tasks that are hosted on Amazon EC2 instances \(p. 90\)](#).

ENI trunking considerations

There are several things to consider when using the ENI trunking feature.

- Only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with version 1.28.1 or later of the container agent and version 1.28.1-2 or later of the ecs-init package, support the increased ENI limits. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. Windows containers are not supported at this time.
- Only new Amazon EC2 instances launched after enabling awsvpcTrunking receive the increased ENI limits and the trunk network interface. Previously launched instances do not receive these features regardless of the actions taken.
- Amazon EC2 instances must have resource-based IPv4 DNS requests turned off. To disable this option, ensure the **Enable resource-based IPV4 (A record) DNS requests** option is deselected when creating a new instance using the Amazon EC2 console. To disable this option using the AWS CLI, use the following command.

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
```

- Amazon EC2 instances in shared subnets are not supported. They will fail to register to a cluster if they are used.
- Your Amazon ECS tasks must use the awsvpc network mode and the EC2 launch type. Tasks using the Fargate launch type always receive a dedicated ENI regardless of how many are launched, so this feature is not needed.
- Your Amazon ECS tasks must be launched in the same Amazon VPC as your container instance. Your tasks will fail to start with an attribute error if they are not within the same VPC.
- When launching a new container instance, the instance transitions to a REGISTERING status while the trunk elastic network interface is provisioned for the instance. If the registration fails, the instance transitions to a REGISTRATION_FAILED status. You can troubleshoot a failed registration by describing the container instance to view the statusReason field which describes the reason for the failure. The container instance then can be manually deregistered or terminated. Once the container instance is successfully deregistered or terminated, Amazon ECS will delete the trunk ENI.

Note

Amazon ECS emits container instance state change events which you can monitor for instances that transition to a REGISTRATION_FAILED state. For more information, see [Container instance state change events \(p. 560\)](#).

- Once the container instance is terminated, the instance transitions to a Deregistering status while the trunk elastic network interface is deprovisioned. The instance then transitions to an INACTIVE status.

- If a container instance in a public subnet with the increased ENI limits is stopped and then restarted, the instance loses its public IP address, and the container agent loses its connection.

Working with container instances with increased ENI limits

Before you launch a container instance with the increased ENI limits, the following prerequisites must be completed.

- The service-linked role for Amazon ECS must be created. The Amazon ECS service-linked role provides 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 \(p. 624\)](#). 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
```

- Your account or container instance IAM role must enable the awsVpcTrunking account setting. This can be done in the following ways:
 - Any user can use the PutAccountSettingDefault API to enable all IAM users and roles on an account
 - A root user can use the PutAccountSetting API to enable the user or container instance role that will register the instance with the cluster
 - A container instance role can enable itself when the PutAccountSetting API is run on an instance prior to it being registered with a cluster

For more information, see [Account settings \(p. 244\)](#).

After the prerequisites are met, you can launch a new container instance using one of the supported Amazon EC2 instance types, and the instance will have the increased ENI limits. For a list of supported instance types, see [Supported Amazon EC2 instance types \(p. 287\)](#). The container instance must have version 1.28.1 or later of the container agent and version 1.28.1-2 or later of the ecs-init package. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Important

Amazon EC2 instances must have resource-based IPv4 DNS requests turned off. To disable this option, ensure the **Enable resource-based IPV4 (A record) DNS requests** option is deselected when creating a new instance using the Amazon EC2 console. To disable this option using the AWS CLI, use the following command.

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
```

To enable all IAM users or roles on your account to the increased ENI limits (AWS Management Console)

1. As the account owner, open the Amazon ECS classic 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 enable to the increased ENI limits.
3. Choose **Account settings**.
4. For **IAM user or role**, ensure your root user or container instance IAM role is selected.
5. For **AWSVPC Trunking**, select the check box. Choose **Save**.

Important

IAM users and IAM roles need the `ecs:PutAccountSetting` permission to perform this action.

6. On the confirmation screen, choose **Confirm** to save the selection.

To enable all user or roles on your account to the increased ENI limits using the command line

Any user on an account can use one of the following commands to modify the default account setting for all IAM users or roles on your account. These changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves.

- [put-account-setting-default](#) (AWS CLI)

```
aws ecs put-account-setting-default \
    --name awsvpcTrunking \
    --value enabled \
    --region us-east-1
```

- [Write-ECSAccountSettingDefault](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name awsvpcTrunking -Value enabled -Region us-east-1 -Force
```

To enable a user or container instance IAM role to the increased ENI limits as the account owner using the command line

The account owner can use one of the following commands and specify the ARN of the principal user or container instance IAM role in the request to modify the account settings.

- [put-account-setting](#) (AWS CLI)

The following example is for modifying the account setting of a specific user:

```
aws ecs put-account-setting \
    --name awsvpcTrunking \
    --value enabled \
    --principal-arn arn:aws:iam::aws_account_id:user/userName \
    --region us-east-1
```

The following example is for modifying the account setting of a specific container instance IAM role:

```
aws ecs put-account-setting \
    --name awsvpcTrunking \
    --value enabled \
    --principal-arn arn:aws:iam::aws_account_id:role/ecsInstanceRole \
    --region us-east-1
```

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

The following example is for modifying the account setting of a specific user:

```
Write-ECSAccountSetting -Name awsvpcTrunking -Value enabled -PrincipalArn
    arn:aws:iam::aws_account_id:user/userName -Region us-east-1 -Force
```

The following example is for modifying the account setting of a specific container instance IAM role:

```
Write-ECSAccountSetting -Name awsVpcTrunking -Value enabled -PrincipalArn arn:aws:iam::aws_account_id:role/ecsInstanceRole -Region us-east-1 -Force
```

To view your container instances with increased ENI limits with the AWS CLI

Each container instance has a default network interface, referred to as a trunk network interface. Use the following command to list your container instances with increased ENI limits by querying for the ecs.awsVpcTrunkId attribute, which indicates it has a trunk network interface.

- [list-attributes](#) (AWS CLI)

```
aws ecs list-attributes \
    --target-type container-instance \
    --attribute-name ecs.awsVpcTrunkId \
    --cluster cluster_name \
    --region us-east-1
```

- [Get-ECSAttributeList](#) (AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsVpcTrunkId -Region us-east-1
```

Supported Amazon EC2 instance types

The following shows the supported Amazon EC2 instance types and how many tasks using the awsVpc network mode can be launched on each instance type before and after enabling the awsVpcTrunking account setting. For the elastic network interface (ENI) limits on each instance type, add one to the current task limit, as the primary network interface counts against the limit, and add two to the new task limit, as both the primary network interface and the trunk network instance count again the limit.

Important

Although other instance types are supported in the same instance family, the a1.metal, c5.metal, c5a.8xlarge, c5ad.8xlarge, c5d.metal, m5.metal, p3dn.24xlarge, r5.metal, r5.8xlarge, and r5d.metal instance types are not supported.

The c5n, d3, d3en, g3, g3s, g4dn, i3, i3en, inf1, m5dn, m5n, m5zn, mac1, r5b, r5n, r5dn, u-12tb1, u-6tb1, u-9tb1, and z1d instance families are not supported.

Topics

- [General purpose \(p. 287\)](#)
- [Compute optimized \(p. 293\)](#)
- [Memory optimized \(p. 297\)](#)
- [Storage optimized \(p. 302\)](#)
- [Accelerated computing \(p. 303\)](#)

General purpose

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
a1.medium	1	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60
m5.large	2	10
m5.xlarge	3	20
m5.2xlarge	3	40
m5.4xlarge	7	60
m5.8xlarge	7	60
m5.12xlarge	7	60
m5.16xlarge	14	120
m5.24xlarge	14	120
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.8xlarge	7	60
m5a.12xlarge	7	60
m5a.16xlarge	14	120
m5a.24xlarge	14	120
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60
m5ad.8xlarge	7	60
m5ad.12xlarge	7	60
m5ad.16xlarge	14	120
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m5d.2xlarge	3	40
m5d.4xlarge	7	60
m5d.8xlarge	7	60
m5d.12xlarge	7	60
m5d.16xlarge	14	120
m5d.24xlarge	14	120
m5d.metal	14	120
m5n.large	2	10
m5n.xlarge	3	20
m5n.2xlarge	3	40
m5n.4xlarge	7	60
m5n.8xlarge	7	60
m5n.12xlarge	7	60
m5n.16xlarge	14	120
m5zn.large	2	14
m5zn.xlarge	3	31
m5zn.2xlarge	3	64
m5zn.3xlarge	7	98
m5zn.6xlarge	7	120
m6a.large	2	10
m6a.xlarge	3	20
m6a.2xlarge	3	40
m6a.4xlarge	7	60
m6a.8xlarge	7	90
m6a.12xlarge	7	120
m6a.16xlarge	14	120
m6a.24xlarge	14	120
m6a.32xlarge	14	120
m6a.48xlarge	14	120
m6a.metal	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6g.medium	1	4
m6g.large	2	10
m6g.xlarge	3	20
m6g.2xlarge	3	40
m6g.4xlarge	7	60
m6g.8xlarge	7	60
m6g.12xlarge	7	60
m6g.16xlarge	14	120
m6gd.metal	14	120
m6gd.medium	1	4
m6gd.large	2	10
m6gd.xlarge	3	20
m6gd.2xlarge	3	40
m6gd.4xlarge	7	60
m6gd.8xlarge	7	60
m6gd.12xlarge	7	60
m6gd.16xlarge	14	120
m6gd.metal	14	120
m6i.large	2	10
m6i.xlarge	3	20
m6i.2xlarge	3	40
m6i.4xlarge	7	60
m6i.8xlarge	7	90
m6i.12xlarge	7	120
m6i.16xlarge	14	120
m6i.24xlarge	14	120
m6i.32xlarge	14	120
m6i.metal	14	120
m6id.large	2	10
m6id.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6id.2xlarge	3	40
m6id.4xlarge	7	60
m6id.8xlarge	7	90
m6id.12xlarge	7	120
m6id.16xlarge	14	120
m6id.24xlarge	14	120
m6id.32xlarge	14	120
m6id.metal	14	120
m6idn.large	2	10
m6idn.xlarge	3	20
m6idn.2xlarge	3	40
m6idn.4xlarge	7	60
m6idn.8xlarge	7	90
m6idn.12xlarge	7	120
m6idn.16xlarge	14	120
m6idn.24xlarge	14	120
m6idn.32xlarge	13	120
m6idn.metal	13	120
m6in.large	2	10
m6in.xlarge	3	20
m6in.2xlarge	3	40
m6in.4xlarge	7	60
m6in.8xlarge	7	90
m6in.12xlarge	7	120
m6in.16xlarge	14	120
m6in.24xlarge	14	120
m6in.32xlarge	13	120
m6in.metal	13	120
m7g.medium	1	4
m7g.large	2	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m7g.xlarge	3	20
m7g.2xlarge	3	40
m7g.4xlarge	7	60
m7g.8xlarge	7	60
m7g.12xlarge	7	60
m7g.16xlarge	14	120
m7g.metal	14	120
m7gd.medium	1	4
m7gd.large	2	10
m7gd.xlarge	3	20
m7gd.2xlarge	3	40
m7gd.4xlarge	7	60
m7gd.8xlarge	7	60
m7gd.12xlarge	7	60
m7gd.16xlarge	14	120
m7i.large	2	10
m7i.xlarge	3	20
m7i.2xlarge	3	40
m7i.4xlarge	7	60
m7i.8xlarge	7	90
m7i.12xlarge	7	120
m7i.16xlarge	14	120
m7i.24xlarge	14	120
m7i.48xlarge	14	120
m7i-flex.large	2	4
m7i-flex.xlarge	3	10
m7i-flex.2xlarge	3	20
m7i-flex.4xlarge	7	40
m7i-flex.8xlarge	7	60
mac2.metal	7	12

Compute optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c5.large	2	10
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60
c5.9xlarge	7	60
c5.12xlarge	7	60
c5.18xlarge	14	120
c5.24xlarge	14	120
c5a.large	2	10
c5a.xlarge	3	20
c5a.2xlarge	3	40
c5a.4xlarge	7	60
c5a.8xlarge	7	60
c5a.12xlarge	7	60
c5a.16xlarge	14	120
c5a.24xlarge	14	120
c5ad.large	2	10
c5ad.xlarge	3	20
c5ad.2xlarge	3	40
c5ad.4xlarge	7	60
c5ad.12xlarge	7	60
c5ad.16xlarge	14	120
c5ad.24xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.12xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c5d.18xlarge	14	120
c5d.24xlarge	14	120
c6a.large	2	10
c6a.xlarge	3	20
c6a.2xlarge	3	40
c6a.4xlarge	7	60
c6a.8xlarge	7	90
c6a.12xlarge	7	120
c6a.16xlarge	14	120
c6a.24xlarge	14	120
c6a.32xlarge	14	120
c6a.48xlarge	14	120
c6a.metal	14	120
c6g.medium	1	4
c6g.large	2	10
c6g.xlarge	3	20
c6g.2xlarge	3	40
c6g.4xlarge	7	60
c6g.8xlarge	7	60
c6g.12xlarge	7	60
c6g.16xlarge	14	120
c6g.metal	14	120
c6gd.medium	1	4
c6gd.large	2	10
c6gd.xlarge	3	20
c6gd.2xlarge	3	40
c6gd.4xlarge	7	60
c6gd.8xlarge	7	60
c6gd.12xlarge	7	60
c6gd.16xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c6gd.metal	14	120
c6gn.medium	1	4
c6gn.large	2	10
c6gn.xlarge	3	20
c6gn.2xlarge	3	40
c6gn.4xlarge	7	60
c6gn.8xlarge	7	60
c6gn.12xlarge	7	60
c6gn.16xlarge	14	120
c6i.large	2	10
c6i.xlarge	3	20
c6i.2xlarge	3	40
c6i.4xlarge	7	60
c6i.8xlarge	7	90
c6i.12xlarge	7	120
c6i.16xlarge	14	120
c6i.24xlarge	14	120
c6i.32xlarge	14	120
c6i.metal	14	120
c6id.large	2	10
c6id.xlarge	3	20
c6id.2xlarge	3	40
c6id.4xlarge	7	60
c6id.8xlarge	7	90
c6id.12xlarge	7	120
c6id.16xlarge	14	120
c6id.24xlarge	14	120
c6id.32xlarge	14	120
c6id.metal	14	120
c6in.large	2	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c6in.xlarge	3	20
c6in.2xlarge	3	40
c6in.4xlarge	7	60
c6in.8xlarge	7	90
c6in.12xlarge	7	120
c6in.16xlarge	14	120
c6in.24xlarge	14	120
c6in.32xlarge	13	120
c6in.metal	13	120
c7g.medium	1	4
c7g.large	2	10
c7g.xlarge	3	20
c7g.2xlarge	3	40
c7g.4xlarge	7	60
c7g.8xlarge	7	60
c7g.12xlarge	7	60
c7g.16xlarge	14	120
c7g.metal	14	120
c7gd.medium	1	4
c7gd.large	2	10
c7gd.xlarge	3	20
c7gd.2xlarge	3	40
c7gd.4xlarge	7	60
c7gd.8xlarge	7	60
c7gd.12xlarge	7	60
c7gd.16xlarge	14	120
c7gn.medium	1	4
c7gn.large	2	10
c7gn.xlarge	3	20
c7gn.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c7gn.4xlarge	7	60
c7gn.8xlarge	7	60
c7gn.12xlarge	7	60
c7gn.16xlarge	14	120
hpc6a.48xlarge	1	120
hpc7g.4xlarge	3	120
hpc7g.8xlarge	3	120
hpc7g.16xlarge	3	120

Memory optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
hpc6id.32xlarge	1	120
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40
r5.4xlarge	7	60
r5.12xlarge	7	60
r5.16xlarge	14	120
r5.24xlarge	14	120
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.8xlarge	7	60
r5a.12xlarge	7	60
r5a.16xlarge	14	120
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r5ad.4xlarge	7	60
r5ad.8xlarge	7	60
r5ad.12xlarge	7	60
r5ad.16xlarge	14	120
r5ad.24xlarge	14	120
r5b.16xlarge	14	120
r5d.large	2	10
r5d.xlarge	3	20
r5d.2xlarge	3	40
r5d.4xlarge	7	60
r5d.8xlarge	7	60
r5d.12xlarge	7	60
r5d.16xlarge	14	120
r5d.24xlarge	14	120
r5dn.16xlarge	14	120
r6a.large	2	10
r6a.xlarge	3	20
r6a.2xlarge	3	40
r6a.4xlarge	7	60
r6a.8xlarge	7	90
r6a.12xlarge	7	120
r6a.16xlarge	14	120
r6a.24xlarge	14	120
r6a.32xlarge	14	120
r6a.48xlarge	14	120
r6a.metal	14	120
r6g.medium	1	4
r6g.large	2	10
r6g.xlarge	3	20
r6g.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r6g.4xlarge	7	60
r6g.8xlarge	7	60
r6g.12xlarge	7	60
r6g.16xlarge	14	120
r6g.metal	14	120
r6gd.medium	1	4
r6gd.large	2	10
r6gd.xlarge	3	20
r6gd.2xlarge	3	40
r6gd.4xlarge	7	60
r6gd.8xlarge	7	60
r6gd.12xlarge	7	60
r6gd.16xlarge	14	120
r6gd.metal	14	120
r6i.large	2	10
r6i.xlarge	3	20
r6i.2xlarge	3	40
r6i.4xlarge	7	60
r6i.8xlarge	7	90
r6i.12xlarge	7	120
r6i.16xlarge	14	120
r6i.24xlarge	14	120
r6i.32xlarge	14	120
r6i.metal	14	120
r6idn.large	2	10
r6idn.xlarge	3	20
r6idn.2xlarge	3	40
r6idn.4xlarge	7	60
r6idn.8xlarge	7	90
r6idn.12xlarge	7	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r6idn.16xlarge	14	120
r6idn.24xlarge	14	120
r6idn.32xlarge	13	120
r6idn.metal	13	120
r6in.large	2	10
r6in.xlarge	3	20
r6in.2xlarge	3	40
r6in.4xlarge	7	60
r6in.8xlarge	7	90
r6in.12xlarge	7	120
r6in.16xlarge	14	120
r6in.24xlarge	14	120
r6in.32xlarge	13	120
r6in.metal	13	120
r6id.large	2	10
r6id.xlarge	3	20
r6id.2xlarge	3	40
r6id.4xlarge	7	60
r6id.8xlarge	7	90
r6id.12xlarge	7	120
r6id.16xlarge	14	120
r6id.24xlarge	14	120
r6id.32xlarge	14	120
r6id.metal	14	120
r7g.medium	1	4
r7g.large	2	10
r7g.xlarge	3	20
r7g.2xlarge	3	40
r7g.4xlarge	7	60
r7g.8xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r7g.12xlarge	7	60
r7g.16xlarge	14	120
r7g.metal	14	120
r7gd.medium	1	4
r7gd.large	2	10
r7gd.xlarge	3	20
r7gd.2xlarge	3	40
r7gd.4xlarge	7	60
r7gd.8xlarge	7	60
r7gd.12xlarge	7	60
r7gd.16xlarge	14	120
u-3tb1.56xlarge	7	12
u-6tb1.56xlarge	14	12
u-18tb1.metal	14	12
u-24tb1.metal	14	12
x2gd.medium	1	10
x2gd.large	2	10
x2gd.xlarge	3	20
x2gd.2xlarge	3	40
x2gd.4xlarge	7	60
x2gd.8xlarge	7	60
x2gd.12xlarge	7	60
x2gd.16xlarge	14	120
x2gd.metal	14	120
x2idn.16xlarge	14	120
x2idn.24xlarge	14	120
x2idn.32xlarge	14	120
x2idn.metal	14	120
x2iedn.xlarge	3	13
x2iedn.2xlarge	3	29

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
x2iedn.4xlarge	7	60
x2iedn.8xlarge	7	120
x2iedn.16xlarge	14	120
x2iedn.24xlarge	14	120
x2iedn.32xlarge	14	120
x2iedn.metal	14	120
x2iezn.2xlarge	3	64
x2iezn.4xlarge	7	120
x2iezn.6xlarge	7	120
x2iezn.8xlarge	7	120
x2iezn.12xlarge	14	120
x2iezn.metal	14	120

Storage optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
i4g.large	2	10
i4g.xlarge	3	20
i4g.2xlarge	3	40
i4g.4xlarge	7	60
i4g.8xlarge	7	60
i4g.16xlarge	14	120
i4i.xlarge	3	8
i4i.2xlarge	3	28
i4i.4xlarge	7	58
i4i.8xlarge	7	118
i4i.16xlarge	14	248
i4i.32xlarge	14	498
i4i.metal	14	498
im4gn.large	2	10
im4gn.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
im4gn.2xlarge	3	40
im4gn.4xlarge	7	60
im4gn.8xlarge	7	60
im4gn.16xlarge	14	120
is4gen.medium	1	4
is4gen.large	2	10
is4gen.xlarge	3	20
is4gen.2xlarge	3	40
is4gen.4xlarge	7	60
is4gen.8xlarge	7	60

Accelerated computing

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
dl1.24xlarge	59	120
g4ad.xlarge	1	12
g4ad.2xlarge	1	12
g4ad.4xlarge	2	12
g4ad.8xlarge	3	12
g4ad.16xlarge	7	12
g5.xlarge	3	6
g5.2xlarge	3	19
g5.4xlarge	7	40
g5.8xlarge	7	90
g5.12xlarge	14	120
g5.16xlarge	7	120
g5.24xlarge	14	120
g5.48xlarge	6	120
g5g.xlarge	3	20
g5g.2xlarge	3	40
g5g.4xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
g5g.8xlarge	7	60
g5g.16xlarge	14	120
g5g.metal	14	120
inf2.xlarge	3	20
inf2.8xlarge	7	90
inf2.24xlarge	14	120
inf2.48xlarge	14	120
p3.2xlarge	3	40
p3.8xlarge	7	60
p3.16xlarge	7	120
p4d.24xlarge	59	120
p4de.24xlarge	59	120
p5.48xlarge	63	242
trn1.2xlarge	3	19
trn1.32xlarge	39	120
trn1n.32xlarge	79	242
vt1.3xlarge	3	40
vt1.6xlarge	7	60
vt1.24xlarge	14	120

Container Instance Memory Management

When the Amazon ECS container agent registers a container instance into a cluster, the agent must determine how much memory the container instance has available to reserve for your tasks. Because of platform memory overhead and memory occupied by the system kernel, this number is different than the installed memory amount that is advertised for Amazon EC2 instances. For example, an m4.1.large instance has 8 GiB of installed memory. However, this does not always translate to exactly 8192 MiB of memory available for tasks when the container instance registers.

If you specify 8192 MiB for the task, and none of your container instances have 8192 MiB or greater of memory available to satisfy this requirement, then the task cannot be placed in your cluster.

You should also reserve some memory for the Amazon ECS container agent and other critical system processes on your container instances, so that your task's containers do not contend for the same memory and possibly starts a system failure. For more information, see [Reserving System Memory \(p. 305\)](#).

The Amazon ECS container agent uses the Docker `ReadMemInfo()` function to query the total memory available to the operating system. Both Linux and Windows provide command line utilities to determine the total memory.

Example - Determine Linux total memory

The `free` command returns the total memory that is recognized by the operating system.

```
$ free -b
```

Example output for an `m4.large` instance running the Amazon ECS-optimized Amazon Linux AMI.

	total	used	free	shared	buffers	cached
Mem:	8373026816	348180480	8024846336	90112	25534464	205418496
-/+ buffers/cache:	117227520	8255799296				

This instance has 8373026816 bytes of total memory, which translates to 7985 MiB available for tasks.

Example - Determine Windows total memory

The `wmic` command returns the total memory that is recognized by the operating system.

```
C:\> wmic ComputerSystem get TotalPhysicalMemory
```

Example output for an `m4.large` instance running the Amazon ECS-optimized Windows AMI.

TotalPhysicalMemory
8589524992

This instance has 8589524992 bytes of total memory, which translates to 8191 MiB available for tasks.

Reserving System Memory

If you occupy all of the memory on a container instance with your tasks, then it is possible that your tasks will contend with critical system processes for memory and possibly start a system failure. The Amazon ECS container agent provides a configuration variable called `ECS_RESERVED_MEMORY`, which you can use to remove a specified number of MiB of memory from the pool that is allocated to your tasks. This effectively reserves that memory for critical system processes.

For example, if you specify `ECS_RESERVED_MEMORY=256` in your container agent configuration file, then the agent registers the total memory minus 256 MiB for that instance, and 256 MiB of memory could not be allocated by ECS tasks. For more information about agent configuration variables and how to set them, see [Amazon ECS container agent configuration \(p. 370\)](#) and [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#).

Viewing Container Instance Memory

You can view how much memory a container instance registers with in the Amazon ECS console (or with the [DescribeContainerInstances](#) API operation). If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, you can observe the memory available for that container instance and then assign your tasks that much memory.

To view container instance memory

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**, and then choose the cluster that hosts your container instance.

3. Choose **Infrastructure**, and then under Container instances, choose a container instance.
4. The **Resources** section shows the registered and available memory for the container instance.

The **Registered** memory value is what the container instance registered with Amazon ECS when it was first launched, and the **Available** memory value is what has not already been allocated to tasks.

Manage container instances remotely using AWS Systems Manager

You can use the Run Command capability in AWS Systems Manager (Systems Manager) to securely and remotely manage the configuration of your Amazon ECS container instances. Run Command provides a simple way to perform common administrative tasks without logging on locally to the instance. You can manage configuration changes across your clusters by simultaneously executing commands on multiple container instances. Run Command reports the status and results of each command.

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

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

For more information about Run Command, see [AWS Systems Manager Run Command](#) in the *AWS Systems Manager User Guide*.

Topics

- [Run Command IAM policy \(p. 306\)](#)
- [Using Run Command \(p. 307\)](#)

Run Command IAM policy

Before you can send commands to your container instances with Run Command, you must attach an IAM policy that allows `ecsInstanceRole` to have access to the Systems Manager APIs. The following procedure describes how to attach the Systems Manager managed policies to your container instance role so that instances launched with this role can use Run Command.

To attach the Systems Manager policies to your `ecsInstanceRole`

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose `ecsInstanceRole`. If the role does not exist, follow the procedures in [Amazon ECS container instance IAM role \(p. 638\)](#) to create the role.
4. Choose the **Permissions** tab.
5. Choose **Attach policies**.
6. To narrow the available policies to attach, for **Filter**, type SSM.
7. In the list of policies, select the box next **AmazonSSMManagedInstanceCore**. Use this policy to provide the minimum permissions that are necessary to use Systems Manager.

For information about other policies you can provide for Systems Manager operations, see [Create an IAM Instance Profile for Systems Manager](#) in the *AWS Systems Manager User Guide*.

8. Choose **Attach Policy**.

Using Run Command

After you attach Systems Manager managed policies to your `ecsInstanceRole` and verify that AWS Systems Manager Agent (SSM Agent) is installed on your container instances, you can start using Run Command to send commands to your container instances. For information about running commands and shell scripts on your instances and viewing the resulting output, see [Running Commands Using Systems Manager Run Command](#) and [Run Command Walkthroughs](#) in the *AWS Systems Manager User Guide*.

Example: To update container instance software with Run Command

A common use case for Run Command is to update the instance software on your entire fleet of container instances at one time.

1. [Attach Systems Manager managed policies to your `ecsInstanceRole`. \(p. 306\)](#)
2. Verify that SSM Agent is installed on your container instances. For more information, see [Manually install SSM Agent on EC2 instances for Linux](#).
3. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager>.
4. In the left navigation pane, choose **Run Command**, and then choose **Run command**.
5. For **Command document**, choose AWS-RunShellScript.
6. In the **Commands** section, enter the command or commands to send to your container instances. In this example, the following command updates the instance software:

```
$ yum update -y
```

7. In the **Target instances** section, select the boxes next to the container instances where you want to run the update command.
8. Choose **Run** to send the command to the specified instances.
9. (Optional) Choose the refresh icon to monitor the command status.
10. (Optional) In **Targets and output**, choose the button next to the instance ID, and then choose **View output**.

Windows instances

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.

Note

Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

Topics

- [Amazon ECS-optimized AMI \(p. 308\)](#)
- [Launching an Amazon ECS Windows container instance \(p. 326\)](#)
- [Bootstrapping Windows container instances with Amazon EC2 user data \(p. 333\)](#)
- [Connect to your container Windows instance \(p. 335\)](#)
- [Deregister an Amazon EC2 backed container instance \(p. 335\)](#)

Amazon ECS-optimized AMI

The Amazon ECS-optimized AMIs are preconfigured with the necessary components that you need to run Amazon ECS workloads. Although you can create your own container instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly.

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and Amazon ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see [the section called "Retrieving Amazon ECS-Optimized AMI metadata" \(p. 308\)](#).

You can subscribe to the Windows AMI Amazon SNS topics to be notified when a new AMI is released or an AMI version is marked private. For more information, see [Subscribing to Amazon ECS-optimized AMI update notifications \(p. 311\)](#).

Important

All ECS-optimized AMI variants produced after August will be migrating from Docker EE (Mirantis) to Docker CE (Moby project).

To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.

Amazon ECS-optimized AMI variants

The following Windows Server variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances. For more information, see [Amazon EC2 Windows containers \(p. 363\)](#).

Important

All ECS-optimized AMI variants produced after August will be migrating from Docker EE (Mirantis) to Docker CE (Moby project).

- [Amazon ECS-optimized Windows Server 2022 Full AMI](#)
- [Amazon ECS-optimized Windows Server 2022 Core AMI](#)
- [Amazon ECS-optimized Windows Server 2019 Full AMI](#)
- [Amazon ECS-optimized Windows Server 2019 Core AMI](#)
- [Amazon ECS-optimized Windows Server 2016 Full AMI](#)
- [Amazon ECS-optimized Windows Server 20H2 Core AMI](#)

Important

On August 9, 2022, the Amazon ECS-optimized Windows Server 20H2 Core AMI reached its end of support date. No new versions of this AMI will be released. For more information, see [Windows Server release information](#).

Windows Server 2022, Windows Server 2019, and Windows Server 2016 are Long-Term Servicing Channel (LTSC) releases. Windows Server 20H2 is a Semi-Annual Channel (SAC) release. For more information, see [Windows Server release information](#).

Retrieving Amazon ECS-Optimized AMI metadata

The AMI ID, image name, operating system, container agent version, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see [GetParameters](#) and [GetParametersByPath](#).

Note

Your administrative user must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.

- `ssm:GetParameters`
- `ssm:GetParameter`
- `ssm:GetParametersByPath`

Systems Manager Parameter Store parameter format

Note

The following Systems Manager Parameter Store API parameters are deprecated and should not be used to retrieve the latest Windows AMIs:

- `/aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended/image_id`
- `/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id`

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

- Windows Server 2022 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

- Windows Server 2022 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

- Windows Server 2019 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Windows Server 2019 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

- Windows Server 2016 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

The following parameter name format retrieves the metadata of the latest stable Windows Server 2019 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

The following is an example of the JSON object that is returned for the parameter value.

```
{  
  "Parameters": [  
    {
```

```
        "Name": "/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized",
        "Type": "String",
        "Value": "{\"image_name\":\"Windows_Server-2019-English-Full-ECS_Optimized-2023.06.13\",\"image_id\":\"ami-0debc1fb48e4aee16\",\"ecs_runtime_version\":\"Docker (CE) version 20.10.21\",\"ecs_agent_version\":\"1.72.0\"}",
        "Version": 58,
        "LastModifiedDate": "2023-06-22T19:37:37.841000-04:00",
        "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized",
        "dataType": "text"
    },
    "InvalidParameters": []
}
```

Each of the fields in the output above are available to be queried as sub-parameters. Construct the parameter path for a sub-parameter by appending the sub-parameter name to the path for the selected AMI. The following sub-parameters are available:

- schema_version
- image_id
- image_name
- os
- ecs_agent_version
- ecs_runtime_version

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest stable Amazon ECS-optimized AMI

You can retrieve the latest stable Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

- **For the Amazon ECS-optimized Windows Server 2022 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2022 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized --region us-east-1
```

- For the Amazon ECS-optimized Windows Server 2016 Full AMI:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

```
Parameters:  
LatestECSOptimizedAMI:  
  Description: AMI ID  
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>  
  Default: /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized/image_id
```

Subscribing to Amazon ECS-optimized AMI update notifications

AWS provides two Amazon SNS topic ARNs for notifications related to the Windows Server Amazon Machine Image (AMIs). One topic sends update notifications when new Windows Server AMIs are released. The other topic sends notifications when previously released Windows Server AMIs are made private. While these topics are not specific to the Amazon ECS-optimized Windows AMIs, because the Amazon ECS-optimized Windows AMIs follow the same release schedule, you can use these notifications for an indication for when new Amazon ECS-optimized Windows AMIs are updated. For more information on subscribing to Windows AMI notifications, see [Subscribing to Windows AMI notifications](#) in the *Amazon EC2 User Guide for Windows Instances*.

Note

Your user, or the role attached to your user must have the sns::subscribe IAM permission to subscribe to an Amazon SNS topic.

Amazon ECS-optimized AMI versions

This topic lists the current and previous versions of the Amazon ECS-optimized AMIs and their corresponding versions of the Amazon ECS container agent, Docker, and the ecs-init package.

The Amazon ECS-optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see [the section called “Retrieving Amazon ECS-Optimized AMI metadata” \(p. 308\)](#).

Windows Amazon ECS-optimized AMIs versions

The following tabs display a list of Windows Amazon ECS-optimized AMIs versions.

Note

For details on referencing the Systems Manager Parameter Store parameter in an AWS CloudFormation template, see [Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template \(p. 262\)](#).

Important

To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.

Windows Server 2022 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Full-ECS_Optimized-2023.07.11	1.73.1	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2023.06.13	1.72.0	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2023.05.18	1.71.1	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2023.04.18	1.70.2	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2023.03.21	1.69.0	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Full-ECS_Optimized-2023.02.21	1.68.2	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2023.01.11	1.68.0	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.12.14	1.67.2	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.11.09	1.65.1	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.10.12	1.64.0	20.10.17 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.09.22	1.63.1	20.10.17 (Docker CE)	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.09.14	1.62.2	20.10.17 (Docker CE)	Private

Amazon ECS-optimized Windows Server 2022 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Full-ECS_Optimized-2022.08.15	1.62.1	20.10.9	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.07.13	1.61.3	20.10.9	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.06.15	1.61.2	20.10.9	Private
Windows_Server-2022-English-Full-ECS_Optimized-2022.01.18	1.57.1	20.10.9	Private
Windows_Server-2022-English-Full-ECS_Optimized-2021.12.16	1.57.1	20.10.7	Private
Windows_Server-2022-English-Full-ECS_Optimized-2021.11.11	1.57.0	20.10.7	Private
Windows_Server-2022-English-Full-ECS_Optimized-2021.009.23	1.55.3	20.10.7	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

Windows Server 2022 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Core-ECS_Optimized-2023.07.11	1.73.1	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2023.06.13	1.72.0	20.10.21 (Docker CE)	Public

Amazon ECS-optimized Windows Server 2022 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Core-ECS_Optimized-2023.05.18	1.71.1	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2023.04.18	1.70.2	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2023.03.21	1.69.0	20.10.21 (Docker CE)	Public
Windows_Server-2022-English-Core-ECS_Optimized-2023.02.21	1.68.2	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2023.01.11	1.68.0	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.12.14	1.67.2	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.11.09	1.65.1	20.10.21 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.10.12	1.64.0	20.10.17 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.09.22	1.63.1	20.10.17 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.09.14	1.62.2	20.10.17 (Docker CE)	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.08.15	1.62.1	20.10.9	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.07.13	1.61.3	20.10.9	Private
Windows_Server-2022-English-Core-ECS_Optimized-2022.06.15	1.61.2	20.10.9	Private
Windows_Server-2022-English-Core-ECS_Optimized-2021.12.16	1.57.1	20.10.7	Private

Amazon ECS-optimized Windows Server 2022 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Core-ECS_Optimized-2021.11.11	1.57.0	20.10.7	Private
Windows_Server-2022-English-Core-ECS_Optimized-2021.009.23	1.55.3	20.10.7	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

Windows Server 2019 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2019 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Full-ECS_Optimized-2023.07.11	1.73.1	20.10.21 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2023.06.13	1.72.0	20.10.21 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2023.05.18	1.71.1	20.10.21 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2023.04.18	1.70.2	20.10.21 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2023.03.21	1.69.0	20.10.21 (Docker CE)	Public
Windows_Server-2019-English-Full-ECS_Optimized-2023.02.21	1.68.2	20.10.21 (Docker CE)	Private
Windows_Server-2019-English-Full-ECS_Optimized-2023.01.11	1.68.0	20.10.21 (Docker CE)	Private

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-1.67.2_English-Full-ECS_Optimized-2022.12.14	1.67.2	20.10.21 (Docker CE)	Private
Windows_Server-2019-1.65.1_English-Full-ECS_Optimized-2022.11.09	1.65.1	20.10.21 (Docker CE)	Private
Windows_Server-2019-1.64.0_English-Full-ECS_Optimized-2022.10.12	1.64.0	20.10.17 (Docker CE)	Private
Windows_Server-2019-1.63.1_English-Full-ECS_Optimized-2022.09.22	1.63.1	20.10.17 (Docker CE)	Private
Windows_Server-2019-1.62.2_English-Full-ECS_Optimized-2022.09.14	1.62.2	20.10.17 (Docker CE)	Private
Windows_Server-2019-1.62.1_English-Full-ECS_Optimized-2022.08.15	1.62.1	20.10.9	Private
Windows_Server-2019-1.61.3_English-Full-ECS_Optimized-2022.07.13	1.61.3	20.10.9	Private
Windows_Server-2019-1.61.2_English-Full-ECS_Optimized-2022.06.15	1.61.2	20.10.9	Private
Windows_Server-2019-1.57.1_English-Full-ECS_Optimized-2022.01.18	1.57.1	20.10.9	Private
Windows_Server-2019-1.57.1_English-Full-ECS_Optimized-2021.12.16	1.57.1	20.10.7	Private
Windows_Server-2019-1.57.0_English-Full-ECS_Optimized-2021.11.11	1.57.0	20.10.7	Private
Windows_Server-2019-1.55.3_English-Full-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Private
Windows_Server-2019-1.55.0_English-Full-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-2019-1.54.02_English-Full-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Private

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019- English-Full-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.10.14	1.45.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Full-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.09.11	1.30.0	19.03.1	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.08.16	1.29.1	19.03.1	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.07.19	1.29.0	18.09.8	Private
Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10	1.27.0	18.09.4	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Windows Server 2019 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2019 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019- English-Core-ECS_Optimized-2023.07.11	1.73.1	20.10.21 (Docker CE)	Public
Windows_Server-2019- English-Core-ECS_Optimized-2023.06.13	1.72.0	20.10.21 (Docker CE)	Public
Windows_Server-2019- English-Core-ECS_Optimized-2023.05.18	1.71.1	20.10.21 (Docker CE)	Public
Windows_Server-2019- English-Core-ECS_Optimized-2023.04.18	1.70.2	20.10.21 (Docker CE)	Public
Windows_Server-2019- English-Core-ECS_Optimized-2023.03.21	1.69.0	20.10.21 (Docker CE)	Public
Windows_Server-2019- English-Core-ECS_Optimized-2023.02.21	1.68.2	20.10.21 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2023.01.11	1.68.0	20.10.21 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.12.14	1.67.2	20.10.21 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.11.09	1.65.1	20.10.21 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.10.12	1.64.0	20.10.17 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.09.22	1.63.1	20.10.17 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.09.14	1.62.2	20.10.17 (Docker CE)	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.08.15	1.62.1	20.10.9	Private
Windows_Server-2019- English-Core-ECS_Optimized-2022.07.13	1.61.3	20.10.9	Private

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-1.61.2_English-Core-ECS_Optimized-2022.06.15	1.61.2	20.10.9	Private
Windows_Server-2019-1.57.1_English-Core-ECS_Optimized-2022.01.18	1.57.1	20.10.9	Private
Windows_Server-2019-1.57.1_English-Core-ECS_Optimized-2021.12.16	1.57.1	20.10.7	Private
Windows_Server-2019-1.57.0_English-Core-ECS_Optimized-2021.11.11	1.57.0	20.10.7	Private
Windows_Server-2019-1.55.3_English-Core-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Private
Windows_Server-2019-1.55.0_English-Core-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Private
Windows_Server-2019-1.54.02_English-Core-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Private
Windows_Server-2019-1.54.0_English-Core-ECS_Optimized-2021.07.08	1.54.0	20.10.6	Private
Windows_Server-2019-1.53.0_English-Core-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Private
Windows_Server-2019-1.52.2_English-Core-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Private
Windows_Server-2019-1.51.0_English-Core-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Private
Windows_Server-2019-1.50.2_English-Core-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2019-1.50.0_English-Core-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2019-1.49.0_English-Core-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-1.48.0_English-Core-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2019-1.47.0_English-Core-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private
Windows_Server-2019-1.45.0_English-Core-ECS_Optimized-2020.10.14	1.45.0	19.03.11	Private
Windows_Server-2019-1.44.3_English-Core-ECS_Optimized-2020.09.09	1.44.3	19.03.11	Private
Windows_Server-2019-1.43.0_English-Core-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2019-1.41.1_English-Core-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private
Windows_Server-2019-1.40.0_English-Core-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2019-1.39.0_English-Core-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private
Windows_Server-2019-1.35.0_English-Core-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2019-1.34.0_English-Core-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2019-1.34.0_English-Core-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private
Windows_Server-2019-1.32.1_English-Core-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2019-1.32.0_English-Core-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Windows Server 2016 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2016 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2023.07.11	1.73.1	20.10.21 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2023.06.13	1.72.0	20.10.21 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2023.05.18	1.71.1	20.10.21 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2023.04.18	1.70.2	20.10.21 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2023.03.21	1.69.0	20.10.21 (Docker CE)	Public
Windows_Server-2016-English-Full-ECS_Optimized-2023.02.21	1.68.2	20.10.21 (Docker CE)	Private
Windows_Server-2016-English-Full-ECS_Optimized-2023.01.11	1.68.0	20.10.21 (Docker CE)	Private
Windows_Server-2016-English-Full-ECS_Optimized-2022.12.14	1.67.2	20.10.21 (Docker CE)	Private
Windows_Server-2016-English-Full-ECS_Optimized-2022.11.09	1.65.1	20.10.21 (Docker CE)	Private
Windows_Server-2016-English-Full-ECS_Optimized-2022.10.12	1.64.0	20.10.17 (Docker CE)	Private
Windows_Server-2016-English-Full-ECS_Optimized-2022.09.22	1.63.1	20.10.17 (Docker CE)	Private

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016- English-Full-ECS_Optimized-2022.09.14	1.62.2	20.10.17 (Docker CE)	Private
Windows_Server-2016- English-Full-ECS_Optimized-2022.08.15	1.62.1	20.10.9	Private
Windows_Server-2016- English-Full-ECS_Optimized-2022.07.13	1.61.3	20.10.9	Private
Windows_Server-2016- English-Full-ECS_Optimized-2022.06.15	1.61.2	20.10.9	Private
Windows_Server-2016- English-Full-ECS_Optimized-2022.01.18	1.57.1	20.10.9	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.12.16	1.57.1	20.10.7	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.11.11	1.57.0	20.10.7	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Private
Windows_Server-2016- English-Full-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Private

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-1.50.2_English-Full-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2016-1.50.0_English-Full-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2016-1.49.0_English-Full-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2016-1.48.0_English-Full-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2016-1.47.0_English-Full-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private
Windows_Server-2016-1.45.0_English-Full-ECS_Optimized-2020.10.14	1.45.0	19.03.12	Private
Windows_Server-2016-1.44.3_English-Full-ECS_Optimized-2020.09.09	1.44.3	19.03.11	Private
Windows_Server-2016-1.43.0_English-Full-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2016-1.41.1_English-Full-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private
Windows_Server-2016-1.40.0_English-Full-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2016-1.39.0_English-Full-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private
Windows_Server-2016-1.35.0_English-Full-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2016-1.34.0_English-Full-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2016-1.34.0_English-Full-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.09.11	1.30.0	19.03.1	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.08.16	1.29.1	19.03.1	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.07.19	1.29.0	18.09.8	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07	1.26.0	18.03.1	Private

Use the following AWS CLI Amazon ECS-optimized Windows Server 2016 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

Building your own Amazon ECS-optimized Windows AMI

EC2 Image Builder can be used to build your own custom Amazon ECS-optimized Windows AMI. This makes it easy to use a Windows AMI with your own license on Amazon ECS. Amazon ECS provides a managed Image Builder component which provides the system configuration needed to run Windows instances to host your containers. Each Amazon ECS managed component includes a specific container agent and Docker version. You can customize your image to use either the latest Amazon ECS managed component, or if an older container agent or Docker version is needed you can specify a different component.

For a full walkthrough of using EC2 Image Builder, see [Getting started with EC2 Image Builder](#) in the *EC2 Image Builder User Guide*.

When building your own Amazon ECS-optimized Windows AMI using EC2 Image Builder, you create an image recipe. Your image recipe must meet the following requirements:

- The **Source image** should be based on Windows Server 2004 Core, Windows Server 2016 Full, Windows Server 2019 Core, Windows Server 2019 Full, Windows Server 2022 Core, or Windows Server 2022 Full. Any other Windows operating system is not supported and may not be compatible with the component.
- When specifying the **Build components**, the `ecs-optimized-ami-windows` component is required. The `update-windows` component is recommended, which ensures the image contains the latest security updates.

Selected components (2)

Expand the component to view versioning options. To sort the build sequence, drag the components up and down.

Sequence	Component (drag the component up or down to change the sequence)	Expand versioning
1	 ecs-optimized-ami-windows ▼ Versioning options <input checked="" type="radio"/> Use latest available component version <input type="radio"/> Specify component version	Owner: Amazon 
2	 update-windows ► Versioning options	Owner: Amazon 

To specify a different component version, expand the **Versioning options** menu and specify the component version you want to use. For more information, see [Listing the ecs-optimized-ami-windows component versions \(p. 326\)](#).

[Listing the ecs-optimized-ami-windows component versions](#)

When creating an EC2 Image Builder recipe and specifying the `ecs-optimized-ami-windows` component, you can either use the default option or you can specify a specific component version. To determine what component versions are available, along with the Amazon ECS container agent and Docker versions contained within the component, you can use the AWS Management Console.

To list the available `ecs-optimized-ami-windows` component versions

1. Open the EC2 Image Builder console at <https://console.aws.amazon.com/imagebuilder/>.
2. On the navigation bar, select the Region that are building your image in.
3. In the navigation pane, under the **Saved configurations** menu, choose **Components**.
4. On the **Components** page, in the search bar type `ecs-optimized-ami-windows` and pull down the qualification menu and select **Quick start (Amazon-managed)**.
5. Use the **Description** column to determine the component version with the Amazon ECS container agent and Docker version your image requires.

[Launching an Amazon ECS Windows container instance](#)

Your Amazon ECS container instances are created using the Amazon EC2 console. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#).

For more information about the launch wizard, see [Launch an instance using the new launch instance wizard](#) in the *Amazon EC2 User Guide for Windows Instances*.

[New Amazon EC2 launch instance wizard](#)

You can use the new Amazon EC2 wizard to launch an instance. You can use the following list for the parameters and leave the parameters not listed as the default. The following instructions take you through each parameter group.

Parameters for instance configuration

- [Initiate instance launch \(p. 327\)](#)

- [Name and tags \(p. 327\)](#)
- [Application and OS Images \(Amazon Machine Image\) \(p. 327\)](#)
- [Instance type \(p. 328\)](#)
- [Key pair \(login\) \(p. 328\)](#)
- [Network settings \(p. 328\)](#)
- [Configure storage \(p. 329\)](#)
- [Advanced details \(p. 329\)](#)

Initiate instance launch

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation bar at the top of the screen, the current AWS Region is displayed (for example, US East (Ohio)). Select a Region in which to launch the instance. This choice is important because some Amazon EC2 resources can be shared between Regions, while others can't.
3. From the Amazon EC2 console dashboard, choose **Launch instance**.

If you see the old launch wizard, the new launch instance wizard is not yet the default view in the currently selected Region. To open the new launch instance wizard, choose **Opt-in to the new experience** at the top right of the screen. If you do not see **Opt-in to the new experience** at the top right, the wizard is not available in your Region. You can launch an instance with the old wizard. For more information, see [the section called "Old Amazon EC2 launch instance wizard" \(p. 330\)](#).

Name and tags

The instance name is a tag, where the key is **Name**, and the value is the name that you specify. You can tag the instance, the volumes, and elastic graphics. For Spot Instances, you can tag the Spot Instance request only.

Specifying an instance name and additional tags is optional.

- For **Name**, enter a descriptive name for the instance. If you don't specify a name, the instance can be identified by its ID, which is automatically generated when you launch the instance.
- To add additional tags, choose **Add additional tags**. Choose **Add tag**, and then enter a key and value, and select the resource type to tag. Choose **Add tag** again for each additional tag to add.

Application and OS Images (Amazon Machine Image)

An Amazon Machine Image (AMI) contains the information required to create an instance. For example, an AMI might contain the software that's required to act as a web server, such as Apache, and your website.

For the latest Amazon ECS-optimized AMIs and their values, see [Windows Amazon ECS-optimized AMI](#).

Use the **Search** bar to find a suitable Amazon ECS-optimized AMI published by AWS.

1. Based on your requirements, enter one of the following AMIs in the **Search** bar and press **Enter**.
 - Windows_Server-2022-English-Full-ECS_Optimized
 - Windows_Server-2022-English-Core-ECS_Optimized
 - Windows_Server-2019-English-Full-ECS_Optimized
 - Windows_Server-2019-English-Core-ECS_Optimized
 - Windows_Server-2016-English-Full-ECS_Optimized

2. On the **Choose an Amazon Machine Image (AMI)** page, select the **Community AMIs** tab.
3. From the list that appears, choose a Microsoft-verified AMI with the most recent publish date and click **Select**.

Instance type

The instance type defines the hardware configuration and size of the instance. Larger instance types have more CPU and memory. For more information, see [Instance types](#).

- For **Instance type**, select the instance type for the instance.

The instance type that you select determines the resources available for your tasks to run on.

Key pair (login)

For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one.

Important

If you choose the **Proceed without key pair (Not recommended)** option, you won't be able to connect to the instance unless you choose an AMI that is configured to allow users another way to log in.

Network settings

Configure the network settings, as necessary.

- **Networking platform:** Choose **Virtual Private Cloud (VPC)**, and then specify the subnet in the **Network interfaces** section.
- **VPC:** Select an existing VPC in which to create the security group.
- **Subnet:** You can launch an instance in a subnet associated with an Availability Zone, Local Zone, Wavelength Zone, or Outpost.

To launch the instance in an Availability Zone, select the subnet in which to launch your instance. To create a new subnet, choose **Create new subnet** to go to the Amazon VPC console. When you are done, return to the launch instance wizard and choose the Refresh icon to load your subnet in the list.

To launch the instance in a Local Zone, select a subnet that you created in the Local Zone.

To launch an instance in an Outpost, select a subnet in a VPC that you associated with the Outpost.

- **Auto-assign Public IP:** If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 659\)](#)

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 412\)](#) in this guide.

- **Firewall (security groups):** Use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored.

- To select an existing security group, choose **Select existing security group**, and select the security group that you created in [Set up to use Amazon ECS \(p. 10\)](#)

Configure storage

The AMI you selected includes one or more volumes of storage, including the root volume. You can specify additional volumes to attach to the instance.

You can use the **Simple** view.

- **Storage type:** Configure the storage for your container instance.

If you are using the Amazon ECS-optimized Amazon Linux 2 AMI, your instance has a single 30 GiB volume configured, which is shared between the operating system and Docker.

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

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

Advanced details

For **Advanced details**, expand the section to view the fields and specify any additional parameters for the instance.

- **Purchasing option:** Choose **Request Spot Instances** to request Spot Instances. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

- **IAM instance profile:** Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- (Optional) **User data:** Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 370\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:

- By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks.

In addition, the following options are available when you use the `awsvpc` network mode.

- `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.
- `AwsVpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in the `awsvpc` network mode.

- `AwsVpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes in the task namespace.

Replace `ip-address` with the IP Address for the additional routes, for example `172.31.42.23/32`.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI -
AwsVpcBlockIMDS -AwsVpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

Old Amazon EC2 launch instance wizard

To launch a container instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select the Region to use.
3. From the **EC2 Dashboard**, choose **Launch instance**.
4. On the **Choose an Amazon Machine Image (AMI)** page, complete the following steps:
 - a. Choose **AWS Marketplace**.
 - b. Choose an AMI for your container instance. You can search for one of the Amazon ECS-optimized AMIs, for example **Windows_2019_Full_ECS_Optimized**. If you do not choose an Amazon ECS-optimized AMI, you must follow the procedures in [Installing the Amazon ECS container agent \(p. 356\)](#).

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

Choose **Next: Configure Instance Details** when you are done.

6. On the **Configure Instance Details** page, complete the following steps:
 - a. Set the **Number of instances** field depending on how many container instances you want to add to your cluster.
 - b. (Optional) To use Spot Instances, for **Purchasing option**, select the check box next to **Request Spot Instances**. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

- c. For **Network**, choose the VPC into which to launch your container instance.
- d. For **Subnet**, choose a subnet to use, or keep the default option to choose the default subnet in any Availability Zone.
- e. Set the **Auto-assign Public IP** field depending on whether you want your instance to be accessible from the public internet. If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 659\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 412\)](#) in this guide. For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).

- f. Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- g. Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 370\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:

- By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks.

In addition, the following options are available when you use the `awsvpc` network mode.

- `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.
- `AwsVpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in the `awsvpc` network mode.
- `AwsVpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes in the task namespace.

Replace `ip-address` with the IP Address for the additional routes, for example `172.31.42.23/32`.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI
-AwsVpcBlockIMDS -AwsVpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

- h. Choose **Next: Add Storage**.

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

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

When done configuring your volumes, choose **Next: Add Tags**.

8. On the **Add Tags** page, specify tags by providing key and value combinations for the container instance. Choose **Add another tag** to add more than one tag to your container instance. For more information resource tags, see [Resources and tags \(p. 529\)](#).

Choose **Next: Configure Security Group** when you are done.

9. On the **Configure Security Group** page, use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored. Select or create a security group as follows, and then choose **Review and Launch**.

10. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. Instead, select the security group that you created in [Set up to use Amazon ECS \(p. 10\)](#) using the following steps:
 - a. Choose **Edit security groups**.
 - b. On the **Configure Security Group** page, select the **Select an existing security group** option.
 - c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.
11. On the **Review Instance Launch** page, choose **Launch**.
12. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

When you are ready, select the acknowledgment field, and then choose **Launch Instances**.
13. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
14. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. If the **Public DNS** column is hidden, choose **Show/Hide, Public DNS**.

Using Spot Instances

A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. Because Spot Instances allow you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly. The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and adjusted gradually based on the long-term supply of and demand for Spot Instances. For more information, see [Spot Instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

You can register Spot Instances to your Amazon ECS clusters. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Spot Instance Draining

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance interruption notice, which gives the instance a two-minute warning before it is interrupted. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in DRAINING status.

Important

Amazon ECS monitors for the Spot Instance interruption notices that have the terminate and stop instance-actions. If you specified either the hibernate instance interruption behavior when requesting your Spot Instances or Spot Fleet, then Amazon ECS Spot Instance draining is not supported for those instances.

When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the PENDING state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

You must set the `ECS_ENABLE_SPOT_INSTANCE_DRAINING` parameter before you start the container agent. Use the following commands to manually turn on Spot Instance draining. Substitute `my-cluster` with the name of your cluster.

```
[Environment]::SetEnvironmentVariable("ECS_ENABLE_SPOT_INSTANCE_DRAINING", "true",  
"Machine")
```

```
# Initialize the agent
Initialize-ECSAgent -Cluster my-cluster
```

For more information, see [the section called “Launching a container instance” \(p. 326\)](#).

Bootstrapping Windows container instances with Amazon EC2 user data

When you launch an Amazon ECS container instance, you have the option of passing user data to the instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud booothooks, shell scripts, and cloud-init directives. For more information about these and other format types, see the [Cloud-Init documentation](#).

You can pass this user data when using the Amazon EC2 launch wizard. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Topics

- [Default Windows user data \(p. 333\)](#)
- [Windows agent installation user data \(p. 334\)](#)
- [Windows IAM roles for tasks \(p. 335\)](#)

Default Windows user data

This example user data script shows the default user data that your Windows container instances receive if you use the [console \(p. 879\)](#). The script below does the following:

- Sets the cluster name to the name you entered.
- Sets the IAM roles for tasks.
- Sets json-file and awslogs as the available logging drivers.

In addition, the following options are available when you use the awsvpc network mode.

- **EnableTaskENI**: This flag turns on task networking and is required when you use the awsvpc network mode.
- **AwsVpcBlockIMDS**: This optional flag blocks IMDS access for the task containers running in awsvpc network mode.
- **AwsVpcAdditionalLocalRoutes**: This optional flag allows you to have additional routes.

Replace ip-address with the IP Address for the additional routes, for example 172.31.42.23/32.

You can use this script for your own container instances (provided that they are launched from the Amazon ECS-optimized Windows Server AMI).

Replace the **-Cluster *cluster-name*** line to specify your own cluster name.

```
<powershell>
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '[{"json-file", "awslogs"}]' -EnableTaskENI -AwsVpcBlockIMDS -AwsVpcAdditionalLocalRoutes
```

```
'["ip-address"]'  
</powershell>
```

For Windows tasks that are configured to use the awslogs logging driver, you must also set the ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE environment variable on your container instance. Use the following syntax.

Replace the -Cluster *cluster-name* line to specify your own cluster name.

```
<powershell>  
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE", $TRUE,  
    "Machine")  
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '['json-  
file',"awslogs"]'  
</powershell>
```

Windows agent installation user data

This example user data script installs the Amazon ECS container agent on an instance launched with a **Windows_Server-2016-English-Full-Containers** AMI. It has been adapted from the agent installation instructions on the [Amazon ECS Container Agent GitHub repository](#) README page.

Note

This script is shared for example purposes. It is much easier to get started with Windows containers by using the Amazon ECS-optimized Windows Server AMI. For more information, see [Creating a cluster using the classic console \(p. 879\)](#).

You can use this script for your own container instances (provided that they are launched with a version of the **Windows_Server-2016-English-Full-Containers** AMI). Be sure to replace the *windows* line to specify your own cluster name (if you are not using a cluster called windows).

```
<powershell>  
# Set up directories the agent uses  
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force  
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force  
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force  
# Set up configuration  
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"  
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")  
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log\  
\ecs-agent.log", "Machine")  
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS\data",  
    "Machine")  
# Download the agent  
$agentVersion = "latest"  
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-  
$agentVersion.zip"  
$zipFile = "${env:TEMP}\ecs-agent.zip"  
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri  
# Put the executables in the executable directory.  
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force  
Set-Location ${ecsExeDir}  
# Set $EnableTaskIAMRoles to $true to enable task IAM roles  
# Note that enabling IAM roles will make port 80 unavailable for tasks.  
[bool]$EnableTaskIAMRoles = $false  
if ($EnableTaskIAMRoles) {  
    $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-ecs-  
    agent/master/misc/windows-deploy/hostsetup.ps1  
    Invoke-Expression $($HostSetupScript.Content)  
}  
# Install the agent service
```

```
New-Service -Name "AmazonECS" `  
    -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe" -windows-service`  
    -DisplayName "Amazon ECS" `  
    -Description "Amazon ECS service runs the Amazon ECS agent" `  
    -DependsOn Docker `  
    -StartupType Manual  
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000  
sc.exe failureflag AmazonECS 1  
Start-Service AmazonECS  
</powershell>
```

Windows IAM roles for tasks

See the following Windows examples regarding bootstrapping IAM task roles.

- [Additional configuration for Windows IAM roles for tasks \(p. 637\)](#)
- [IAM roles for task container bootstrap script \(p. 638\)](#)

Connect to your container Windows instance

You can connect to your Windows instances to perform basic administrative tasks, such as installing or updating software or accessing diagnostic logs. To connect to your instance using Remote Desktop Protocol (RDP), your Windows instance must meet the following prerequisites.

- Amazon EC2 instances created from most Windows AMIs allow you to connect using Remote Desktop Protocol (RDP). RDP allows you to connect to and use your instance in the same way you use a computer sitting in front of you. It is available on most editions of Windows and available for Mac OS.
- Your Windows instance must have been launched with a valid Amazon EC2 key pair. Amazon EC2 instances have no password, you use a key pair for access over RDP. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance. For more information, see [the section called “Launching a container instance” \(p. 326\)](#).
- Ensure that the security group associated with your instance allows incoming RDP traffic (port 3389) from your IP address. The default security group doesn't allow incoming RDP traffic by default. For more information, see [Authorize inbound traffic for your Windows instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

1. Find the public IP or DNS address for your container instance.
 - a. Open the console at <https://console.aws.amazon.com/ecs/v2>.
 - b. In the navigation pane, choose **Clusters** and select the cluster that hosts the instance.
 - c. On the **Cluster : name** page, choose the **Infrastructure** tab.
 - d. Under **Container instances**, select the instance ID.
 - e. On the **Instances** page, record the **Public IP** or **Public DNS** for your instance.
2. Find the default username for your container instance AMI.
3. You can connect to your instance by using RDP. For more information, see [Connect to your Windows instance using RDP](#) in the *Amazon EC2 User Guide for Windows Instances*.

Deregister an Amazon EC2 backed container instance

Important

This topic is for container instances created in Amazon EC2 only. For more information about deregistering external instances, see [Deregistering an external instance \(p. 346\)](#).

When you are finished with an Amazon EC2 backed container instance, you should deregister it from your cluster. Following deregistration, the container instance is no longer able to accept new tasks.

If you have tasks running on the container instance when you deregister it, these tasks remain running until you terminate the instance or the tasks stop through some other means. However, these tasks are orphaned which means they are no longer monitored or accounted for by Amazon ECS. If an orphaned task on your container instance is part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different container instance, if possible. Any containers in orphaned service tasks that are registered with an Application Load Balancer target group are deregistered. They begin connection draining according to the settings on the load balancer or target group. If an orphaned tasks is using the aws vpc network mode, their elastic network interfaces are deleted.

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

When deregistering a container instance, be aware of the following considerations.

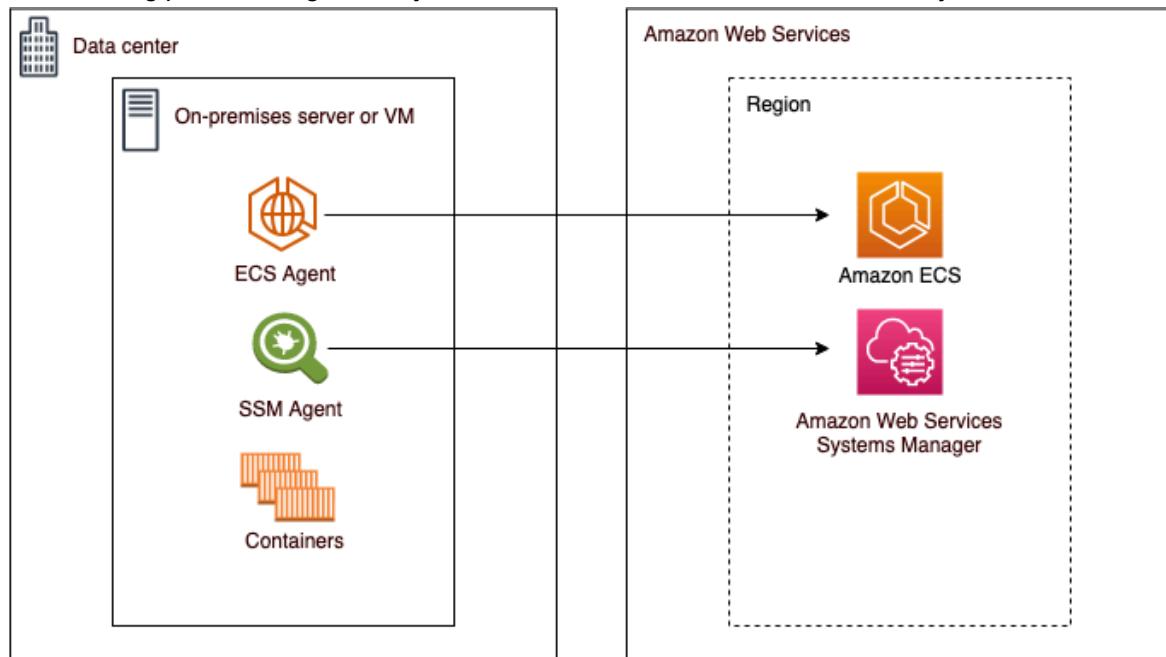
- Because each container instance has unique state information, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances in the new cluster. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS container instance \(p. 272\)](#).
- If the container instance is managed by an Auto Scaling group or a AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack. Otherwise, the Auto Scaling group or AWS CloudFormation will create a new instance after you terminate it.
- If you terminate a running container instance with a connected Amazon ECS container agent, the agent automatically deregisters the instance from your cluster. Stopped container instances or instances with disconnected agents are not automatically deregistered when terminated.
- Deregistering a container instance removes the instance from a cluster, but it does not terminate the Amazon EC2 instance. If you are finished using the instance, be sure to terminate it to stop billing. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the instance.
4. On the **Cluster : name** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance : id** page, choose **Deregister**.
7. On the confirmation screen, choose **Deregister**.
8. If you are finished with the container instance, terminate the underlying Amazon EC2 instance. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

External instances (Amazon ECS Anywhere)

Amazon ECS Anywhere provides support for registering an *external instance* such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster. External instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, the lack of Elastic Load Balancing support makes running these workloads less efficient. Amazon ECS added a new EXTERNAL launch type that you can use to create services or run tasks on your external instances.

The following provides a high-level system architecture overview of Amazon ECS Anywhere.



Topics

- [Supported operating systems and system architectures \(p. 337\)](#)
- [Considerations \(p. 338\)](#)
- [IAM permissions for Amazon ECS Anywhere \(p. 340\)](#)
- [Registering an external instance to a cluster \(p. 342\)](#)
- [Deregistering an external instance \(p. 346\)](#)
- [Running workloads on external instances \(p. 349\)](#)
- [Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance \(p. 350\)](#)

Supported operating systems and system architectures

The following is the list of supported operating systems and system architectures.

- Amazon Linux 2
- CentOS 7

Important

CentOS 8 has reached its End Of Life (EOL) on December 31, 2021 and is no longer supported by Amazon ECS Anywhere.

- CentOS Stream 8
- RHEL 7, RHEL 8 — Neither Docker or RHEL's open package repositories support installing Docker natively on RHEL. You must ensure that Docker is installed before you run the install script that's described in this document.
- Fedora 32, Fedora 33 — Fedora 32 and Fedora 33 default to using cgroups.v2, which isn't supported by Amazon ECS. As a result, the server's default grub configuration must be changed and the server rebooted. For instructions, see [Changing cgroup version](#) in the Docker documentation.

- openSUSE Tumbleweed
- Ubuntu 18, Ubuntu 20
- Debian 10

Important

Debian 9 Long Term Support (LTS support) ended on June 30, 2022 and is no longer supported by Amazon ECS Anywhere.

- SUSE Enterprise Server 15
- The x86_64 and ARM64 CPU architectures are supported.
- The following Windows operating system versions are supported:
 - Windows Server 2022
 - Windows Server 2019
 - Windows Server 2016
 - Windows Server 20H2

Considerations

Before you start using external instances, be aware of the following considerations.

- You can register an external instance to one cluster at a time. For instructions on how to register an external instance with a different cluster, see [Deregistering an external instance \(p. 346\)](#).
- Your external instances require an IAM role that allows them to communicate with AWS APIs. For more information, see [Required IAM permissions for external instances \(p. 340\)](#).
- Your external instances should not have a preconfigured instance credential chain defined locally as this will interfere with the registration script.
- To send container logs to CloudWatch Logs, make sure that you create and specify a task execution IAM role in your task definition. For more information, see [Conditional IAM permissions \(p. 342\)](#).
- When an external instance is registered to a cluster, the `ecs.capability.external` attribute is associated with the instance. This attribute identifies the instance as an external instance. You can add custom attributes to your external instances to use as a task placement constraint. For more information, see [Custom attributes \(p. 439\)](#).
- You can add resource tags to your external instance. For more information, see [Adding tags to an external container instance \(p. 535\)](#).
- ECS Exec is supported on external instances. For more information, see [Using Amazon ECS Exec for debugging \(p. 755\)](#).
- The following are additional considerations that are specific to networking with your external instances. For more information, see [Networking with ECS Anywhere \(p. 339\)](#).
 - Service load balancing isn't supported.
 - Service discovery isn't supported.
 - Tasks that run on external instances must use the bridge, host, or none network modes. The `awsvpc` network mode isn't supported.
 - There are Amazon ECS service domains in each AWS Region. These service domains must be allowed to send traffic to your external instances.
 - The SSM Agent installed on your external instance maintains IAM credentials that are rotated every 30 minutes using a hardware fingerprint. If your external instance loses connection to AWS, the SSM Agent automatically refreshes the credentials after the connection is re-established. For more information, see [Validating on-premises servers and virtual machines using a hardware fingerprint](#) in the *AWS Systems Manager User Guide*.
 - The `UpdateContainerAgent` API isn't supported. For instructions on how to update the SSM Agent or the Amazon ECS agent on your external instances, see [Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance \(p. 350\)](#).

- Amazon ECS capacity providers aren't supported. To create a service or run a standalone task on your external instances, use the EXTERNAL launch type.
- SELinux isn't supported.
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported.
- Integration with App Mesh isn't supported.
- When you run ECS Anywhere on Windows, you must use your own Windows license on the on-premises infrastructure.

Networking with ECS Anywhere

Amazon ECS external instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, such as a web service, the lack of Elastic Load Balancing support makes running these workloads less efficient because there isn't support for placing these workloads behind a load balancer.

The following are additional considerations that are specific to networking with your external instances.

- Service load balancing isn't supported.
- Service discovery isn't supported.
- Linux tasks that run on external instances must use the bridge, host, or none network modes. The `awsvpc` network mode isn't supported.

For more information about each network mode, see [Choosing a network mode](#) in the *Amazon ECS Best Practices Guide*.

- Windows tasks that run on external instances must use the default network mode.
- There are Amazon ECS service domains in each AWS Region. These service domains must be allowed to send traffic to your external instances.
- The SSM Agent installed on your external instance maintains IAM credentials that are rotated every 30 minutes using a hardware fingerprint. If your external instance loses connection to AWS, the SSM Agent automatically refreshes the credentials after the connection is re-established. For more information, see [Validating on-premises servers and virtual machines using a hardware fingerprint](#) in the *AWS Systems Manager User Guide*.

The following domains are used for communication between the Amazon ECS service and the Amazon ECS agent that's installed on your external instance. Make sure that traffic is allowed and that DNS resolution works. For each endpoint, `region` represents the Region identifier for an AWS Region that's supported by Amazon ECS, such as `us-east-2` for the US East (Ohio) Region. The endpoints for all Regions that you use should be allowed. For the `ecs-a` and `ecs-t` endpoints, you should include an asterisk (for example, `ecs-a-*`).

- `ecs-a-*.region.amazonaws.com` — This endpoint is used when managing tasks.
- `ecs-t-*.region.amazonaws.com` — This endpoint is used to manage task and container metrics.
- `ecs.region.amazonaws.com` — This is the service endpoint for Amazon ECS.
- `ssm.region.amazonaws.com` — This is the service endpoint for AWS Systems Manager.
- `ec2messages.region.amazonaws.com` — This is the service endpoint that AWS Systems Manager uses to communicate between the Systems Manager agent and the Systems Manager service in the cloud.
- `ssmmessages.region.amazonaws.com` — This is the service endpoint that is required to create and delete session channels with the Session Manager service in the cloud.
- If your tasks require communication with any other AWS services, make sure that those service endpoints are allowed. Example applications include using Amazon ECR to pull container images or

using CloudWatch for CloudWatch Logs. For more information, see [Service endpoints](#) in the [AWS General Reference](#).

Amazon FSx for Windows File Server with ECS Anywhere

In order to use the Amazon FSx for Windows File Server with Amazon ECS external instances you must establish a connection between your on-premises data center and the AWS Cloud. For information about the options for connecting your network to your VPC, see [Amazon Virtual Private Cloud Connectivity Options](#).

gMSA with ECS Anywhere

The following use cases are supported for ECS Anywhere.

- The Active Directory is in the AWS Cloud - For this configuration, you create a connection between your on-premises network and the AWS Cloud using an AWS Direct Connect connection. For information about how to create the connection, see [Amazon Virtual Private Cloud Connectivity Options](#). You create an Active Directory in the AWS Cloud. For information about how to get started with AWS Directory Service, see [Setting up AWS Directory Service](#) in the [AWS Directory Service Administration Guide](#). You can then join your external instances to the domain using the AWS Direct Connect connection. For information about working with gMSA with Amazon ECS, see [the section called "Using gMSAs for Windows Containers" \(p. 416\)](#).
- The Active Directory is in the on-premises data center. - For this configuration, you join your external instances to the on-premises Active Directory. You then use the locally available credentials when you run the Amazon ECS tasks.

IAM permissions for Amazon ECS Anywhere

There are several required and conditional IAM permissions that apply to Amazon ECS Anywhere. The following sections describe the IAM permissions in more detail.

Required IAM permissions for external instances

When registering an on-premises server or virtual machine (VM) to your cluster, the server or VM requires an IAM role to communicate with AWS APIs. You only need to create this IAM role once for each AWS account. However, this IAM role must be associated with each server or VM that you register to a cluster. This role is the `ECSAnywhereRole`. You can create this role manually. Alternatively, Amazon ECS can create the role on your behalf when you register an external instance in the AWS Management Console.

AWS provides two managed IAM policies that can be used when creating the ECS Anywhere IAM role, the `AmazonSSMManagedInstanceCore` and `AmazonEC2ContainerServiceforEC2Role` policies. The `AmazonEC2ContainerServiceforEC2Role` policy includes permissions that likely provide more access than you need. Therefore, depending on your specific use case, we recommend that you create a custom policy adding only the permissions from that policy that you require in it. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

To create the ECS Anywhere IAM role (AWS Management Console)

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. Choose the **AWS service** role type, and then choose **Systems Manager**, **Allows SSM to call AWS services on your behalf**.
4. Choose the **Systems Manager** use case and then **Next: Permissions**.

5. In the **Attached permissions policy** section, search for and select the **AmazonSSMManagedInstanceCore** and **AmazonEC2ContainerServiceforEC2Role** policies and then choose **Next: Review**.

Important

The **AmazonEC2ContainerServiceforEC2Role** managed policy provides permissions that are needed for your on-premises server or VM. However, the **AmazonEC2ContainerServiceforEC2Role** managed policy might grant permissions that aren't needed for your use case. Review the permissions granted by this policy and see if your use case doesn't require all of the permissions. Then, depending on your situation, optionally create a custom policy and add only the permissions you require. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

6. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
7. For **Role name**, enter **ECSAnywhereRole** and optionally you can edit the description.
8. Review your role information and then choose **Create role**.
9. Perform a search for the **ECSAnywhereRole** and then select it to view the role details.

To create the ECS Anywhere IAM role (AWS CLI)

1. Create a file named **ssm-trust-policy.json** that contains the trust policy to use for the IAM role. The file should contain the following:

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

2. Create an IAM role named **ecsAnywhereRole** using the trust policy that's created in the previous step.

```
aws iam create-role \
--role-name ecsAnywhereRole \
--assume-role-policy-document file://ssm-trust-policy.json
```

3. Attach the AWS managed **AmazonSSMManagedInstanceCore** policy to the **ecsAnywhereRole** role. This policy provides the Systems Manager API permissions that are needed for your on-premises server or VM.

```
aws iam attach-role-policy \
--role-name ecsAnywhereRole \
--policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

4. Attach the AWS managed **AmazonEC2ContainerServiceforEC2Role** policy to the **ecsAnywhereRole** role.

```
aws iam attach-role-policy \
--role-name ecsAnywhereRole \
--policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role
```

Important

The `AmazonEC2ContainerServiceforEC2Role` managed policy provides permissions that are needed for your on-premises server or VM. However, the `AmazonEC2ContainerServiceforEC2Role` managed policy might grant permissions that aren't needed for your use case. Review the permissions granted by this policy and see if your use case doesn't require all of the permissions. Then, depending on your situation, optionally create a custom policy and add only the permissions you require. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

Conditional IAM permissions

The task execution IAM role grants the Amazon ECS container agent permission to make AWS API calls on your behalf. When a task execution IAM role is used, it must be specified in your task definition. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

The task execution role is required if any of the following conditions apply:

- You're sending container logs to CloudWatch Logs using the `awslogs` log driver.
- Your task definition specifies a container image that's hosted in an Amazon ECR private repository. However, if the `ECSAnywhereRole` IAM role that's associated with your external instance also includes the permissions necessary to pull images from Amazon ECR then your task execution role doesn't need to include them.

Registering an external instance to a cluster

For each external instance you register with an Amazon ECS cluster, it must have the SSM Agent, the Amazon ECS container agent, and Docker installed. To register the external instance to an Amazon ECS cluster, it must first be registered as an AWS Systems Manager managed instance. You can create the installation script in a few clicks on the Amazon ECS console. The installation script includes an Systems Manager activation key and commands to install each of the required agents and Docker. The installation script must be run on your on-premises server or VM to complete the installation and registration steps.

Note

Before registering your Linux external instance with the cluster, create the `/etc/ecs/ecs.config` file on your external instance and add any container agent configuration parameters that you want. You can't do this after registering the external instance to a cluster. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

AWS Management Console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose a cluster to register your external instance to.
5. On the **Cluster : name** page, choose the **Infrastructure** tab.
6. On the **Register external instances** page, complete the following steps.
 - a. For **Activation key duration (in days)**, enter the number of days that the activation key remains active for. After the number of days you entered pass, the key no longer works when registering an external instance.
 - b. For **Number of instances**, enter the number of external instances that you want to register to your cluster with the activation key.

- c. For **Instance role**, choose the IAM role to associate with your external instances. If a role wasn't already created, choose **Create new role** to have Amazon ECS create a role on your behalf. For more information about what IAM permissions are required for your external instances, see [Required IAM permissions for external instances \(p. 340\)](#).
- d. Copy the registration command. This command should be run on each external instance you want to register to the cluster.

Important

The bash portion of the script must be run as root. If the command isn't run as root, an error is returned.

- e. Choose **Close**.

AWS CLI for Linux operating systems

1. Create an Systems Manager activation pair. This is used for Systems Manager managed instance activation. The output includes an ActivationId and ActivationCode. You use these in a later step. Make sure that you specify the ECS Anywhere IAM role that you created. For more information, see [Required IAM permissions for external instances \(p. 340\)](#).

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. On your on-premises server or virtual machine (VM), download the installation script.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh"
```

3. (Optional) On your on-premises server or virtual machine (VM), use the following steps to verify the installation script using the script signature file.
 - a. Download and install GnuPG. For more information about GNUpG, see the [GnuPG website](#). For Linux systems, install gpg using the package manager on your flavor of Linux.
 - b. Retrieve the Amazon ECS PGP public key.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

- c. Download the installation script signature. The signature is an ascii detached PGP signature stored in a file with the .asc extension.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh.asc" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh.asc"
```

- d. Verify the installation script file using the key.

```
gpg --verify /tmp/ecs-anywhere-install.sh.asc /tmp/ecs-anywhere-install.sh
```

The following is the expected output.

```
gpg: Signature made Tue 25 May 2021 07:16:29 PM UTC
gpg:                               using RSA key 50DECCC4710E61AF
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: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. On your on-premises server or virtual machine (VM), run the installation script. Specify the cluster name, Region, and the Systems Manager activation ID and activation code from the first step.

```
sudo bash /tmp/ecs-anywhere-install.sh \
--region $REGION \
--cluster $CLUSTER_NAME \
--activation-id $ACTIVATION_ID \
--activation-code $ACTIVATION_CODE
```

For an on-premises server or virtual machine (VM) that has the NVIDIA driver installed for GPU workloads, you must add the --enable-gpu flag to the installation script. When this flag is specified, the install script verifies that the NVIDIA driver is running and then adds the required configuration variables to run your Amazon ECS tasks. For more information about running GPU workloads and specifying GPU requirements in a task definition, see [Specifying GPUs in your task definition \(p. 145\)](#).

```
sudo bash /tmp/ecs-anywhere-install.sh \
--region $REGION \
--cluster $CLUSTER_NAME \
--activation-id $ACTIVATION_ID \
--activation-code $ACTIVATION_CODE \
--enable-gpu
```

Use the following steps to register an existing external instance with a different cluster.

To register an existing external instance with a different cluster

1. Stop the Amazon ECS container agent.

```
sudo systemctl stop ecs.service
```

2. Edit the /etc/ecs/ecs.config file and on the ECS_CLUSTER line, ensure the cluster name matches the name of the cluster to register the external instance with.
3. Remove the existing Amazon ECS agent data.

```
sudo rm /var/lib/ecs/data/agent.db
```

4. Start the Amazon ECS container agent.

```
sudo systemctl start ecs.service
```

AWS CLI for Windows operating systems

1. Create an Systems Manager activation pair. This is used for Systems Manager managed instance activation. The output includes an ActivationId and ActivationCode. You use these in a later step. Make sure that you specify the ECS Anywhere IAM role that you created. For more information, see [Required IAM permissions for external instances \(p. 340\)](#).

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. On your on-premises server or virtual machine (VM), download the installation script.

```
Invoke-RestMethod -URI "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-
install.ps1" -OutFile "ecs-anywhere-install.ps1"
```

3. (Optional) The Powershell script is signed by Amazon and therefore, Windows automatically performs the certificate validation on the same. You do not need to perform any manual validation.

To manually verify the certificate, right-click on the file, navigate to properties and use the Digital Signatures tab to obtain more details.

This option is only available when the host has the certificate in the certificate store.

The verification should return information similar to the following:

```
# Verification (PowerShell)
Get-AuthenticodeSignature -FilePath .\ecs-anywhere-install.ps1

SignerCertificate          Status      Path
-----                    -----      -----
EXAMPLECERTIFICATE          Valid       ecs-anywhere-install.ps1

...
Subject                   : CN="Amazon Web Services, Inc.",...
-----
```

4. On your on-premises server or virtual machine (VM), run the installation script. Specify the cluster name, Region, and the Systems Manager activation ID and activation code from the first step.

```
.\ecs-anywhere-install.ps1 -Region $Region -Cluster $Cluster -
ActivationID $ActivationID -ActivationCode $ActivationCode
```

5. Verify the Amazon ECS container agent is running.

```
Get-Service AmazonECS

Status     Name           DisplayName
-----     ----           -----
Running    AmazonECS     Amazon ECS
```

Use the following steps to register an existing external instance with a different cluster.

To register an existing external instance with a different cluster

1. Stop the Amazon ECS container agent.

```
Stop-Service AmazonECS
```

2. Modify the ECS_CLUSTER parameter so that the cluster name matches the name of the cluster to register the external instance with.

```
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", $ECSCluster,
[System.EnvironmentVariableTarget]::Machine)
```

3. Remove the existing Amazon ECS agent data.

```
Remove-Item -Recurse -Force $env:ProgramData\Amazon\ECS\data\*
```

4. Start the Amazon ECS container agent.

```
Start-Service AmazonECS
```

The AWS CLI can be used to create a Systems Manager activation before running the installation script to complete the external instance registration process.

Deregistering an external instance

We recommend that, after you finish using an external instance, you deregister the instance from both Amazon ECS and AWS Systems Manager. Following deregistration, the external instance is no longer able to accept new tasks.

If you have tasks that are running on the container instance when you deregister it, the tasks remain running until they stop through some other means. However, these tasks are no longer monitored or accounted for by Amazon ECS. If these tasks on your external instance are part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different instance, if possible.

To register an external instance to a new cluster, after the external instance has been deregistered from both Amazon ECS and Systems Manager, you can clean up the remaining AWS resources on the instance and register it with a new cluster.

AWS Management Console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.
4. On the **Cluster : name** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the external instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance : id** page, choose **Deregister**.
7. Review the deregistration message. Select **Deregister from AWS Systems Manager** to also deregister the external instance as an Systems Manager managed instance. Choose **Deregister**.

Note

You can deregister the external instance as an Systems Manager managed instance in the Systems Manager console. For instructions, see [Deregistering managed instances](#) in the *AWS Systems Manager User Guide*.

8. After you deregister the instance, clean up AWS resources on your on-premises server or VM .

Operating system	Steps
Linux	<p>a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.</p> <pre>sudo systemctl stop ecs amazon-ssm-agent</pre>

Operating system	Steps	
	<p>b. Remove the Amazon ECS and Systems Manager packages.</p> <p>For CentOS 7, CentOS 8, and RHEL 7</p> <pre style="border: 1px solid black; padding: 5px;"><code>sudo yum remove -y amazon-ecs-init amazon-ssm-agent</code></pre> <p>For SUSE Enterprise Server 15</p> <pre style="border: 1px solid black; padding: 5px;"><code>sudo zypper remove -y amazon-ecs-init amazon-ssm-agent</code></pre> <p>For Debian and Ubuntu</p> <pre style="border: 1px solid black; padding: 5px;"><code>sudo apt remove -y amazon-ecs-init amazon-ssm-agent</code></pre>	
Windows	<p>a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.</p> <pre style="border: 1px solid black; padding: 5px;"><code>Stop-Service AmazonECS</code></pre> <pre style="border: 1px solid black; padding: 5px;"><code>Stop-Service AmazonSSMAgent</code></pre> <p>b. Remove the Amazon ECS package.</p> <pre style="border: 1px solid black; padding: 5px;"><code>.\ecs-anywhere-install.ps1 -Uninstall</code></pre>	

AWS CLI

1. You need the instance ID and the container instance ARN to deregister the container instance. If you do not have these values, run the following commands

Run the following command to get the instance ID.

You use the instance ID (`instanceID`) to get the container instance ARN (`containerInstanceARN`).

```
instanceId=$(aws ssm describe-instance-information --region "{{ region }}" | jq ".InstanceInformationList[] | select(.IPAddress==\"{{ IPv4 Address }}\") | .InstanceId" | tr -d ''')
```

Run the following commands.

You use the `containerInstanceArn` as a parameter in the command to deregister the instance (`deregister-container-instance`).

```
instances=$(aws ecs list-container-instances --cluster "{{ cluster }}" --region "{{ region }}" | jq -c '.containerInstanceArns')
containerInstanceArn=$(aws ecs describe-container-instances --cluster "{{ cluster }}" --region "{{ region }}" --container-instances $instances | jq ".containerInstances[] | select(.ec2InstanceId==\"{{ instanceId }}\") | .containerInstanceArn" | tr -d '')
```

2. Run the following command to drain the instance.

```
aws ecs update-container-instances-state --cluster "{{ cluster }}" --region "{{ region }}" --container-instances "{{ containerInstanceArn }}" --status DRAINING
```

3. After the container instance finishes draining, run the following command to deregister the instance.

```
aws ecs deregister-container-instance --cluster "{{ cluster }}" --region "{{ region }}" --container-instance "{{ containerInstanceArn }}"
```

4. Run the following command to remove the container instance from SSM.

```
aws ssm deregister-managed-instance --region "{{ region }}" --instance-id "{{ instanceId }}"
```

5. After you deregister the instance, clean up AWS resources on your on-premises server or VM .

Operating system	Steps
Linux	<p>a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> sudo systemctl stop ecs amazon-ssm-agent </div> <p>b. Remove the Amazon ECS and Systems Manager packages.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> sudo (yum/apt/zypper) remove amazon-ecs-init amazon-ssm-agent </div> <p>c. Remove the leftover directories.</p>

Operating system	Steps
	<pre>sudo rm -rf /var/lib/ ecs /etc/ecs /var/lib/ amazon/ssm /var/log/ ecs /var/log/amazon/ssm</pre>
Windows	<p>a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Stop-Service AmazonECS </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Stop-Service AmazonSSMAgent </div> <p>b. Remove the Amazon ECS package.</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> .\ecs-anywhere- install.ps1 -Uninstall </div>

Running workloads on external instances

After an external instance is registered to your cluster, you can run your containerized workloads. The first step is to register a task definition. Amazon ECS provides the `requiresCompatibilities` parameter to validate the task definition is compatible with your external instances. When you deploy your workload, use the EXTERNAL launch type when creating your service or running your standalone task.

Creating a task definition that is compatible with your external instances

When registering an Amazon ECS task definition, use the `requiresCompatibilities` parameter and specify EXTERNAL which validates that the task definition is compatible to use when running Amazon ECS workloads on your external instances. If you use the console for registering a task definition, you must use the JSON editor. For more information, see [Creating a task definition using the console \(p. 125\)](#).

Important

If your tasks require a task execution IAM role, make sure that it's specified in the task definition. For more information, see [Conditional IAM permissions \(p. 342\)](#).

The following is an example task definition.

Linux

```
{
  "requiresCompatibilities": [
    "EXTERNAL"
  ],
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "public.ecr.aws/nginx/nginx:latest",
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ]
    }
  ]
}
```

```
"memory": 256,  
"cpu": 256,  
"essential": true,  
"portMappings": [  
    {"containerPort": 80,  
     "hostPort": 8080,  
     "protocol": "tcp"  
   }  
,  
  {"containerPort": 80,  
     "hostPort": 8080,  
     "protocol": "tcp"  
   }],  
  "networkMode": "bridge",  
  "family": "nginx"  
}
```

Windows

```
{  
  "requiresCompatibilities": [  
    "EXTERNAL"  
  ],  
  "containerDefinitions": [  
    {"name": "windows-container",  
     "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",  
     "memory": 256,  
     "cpu": 512,  
     "essential": true,  
     "portMappings": [  
       {"containerPort": 80,  
        "hostPort": 8080,  
        "protocol": "tcp"  
      }  
,  
      {"containerPort": 80,  
        "hostPort": 8080,  
        "protocol": "tcp"  
      }]  
    ],  
    "networkMode": "bridge",  
    "family": "windows-container"  
  }
```

Running a standalone task or creating a service

After you register your external instances to your cluster, grant the relevant IAM permissions, and register a valid task definition, you can start to run your workloads on Amazon ECS. When running your standalone tasks or creating a service, specify the EXTERNAL launch type, and the Amazon ECS scheduler places the tasks on your external instances.

For instructions on how to create services, see [Creating a service using the console \(p. 456\)](#).

For more information about running standalone tasks, see [Running a standalone task using the Amazon ECS console \(p. 430\)](#).

Verifying your running tasks on an external Windows instance

After you launch a task, you can verify the status by checking the Amazon ECS agent log (`ecs-agent.log`) which is located in the `\ProgramData\Amazon\ECS\log` directory.

Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance

Your on-premises server or VM must run both the AWS Systems Manager Agent (SSM Agent) and the Amazon ECS container agent when running Amazon ECS workloads. AWS releases new versions of these

agents when any capabilities are added or updated. If your external instances are using an earlier version of either agent, you can update them using the following procedures.

Updating the SSM Agent on an external instance

AWS Systems Manager recommends that you automate the process of updating the SSM Agent on your instances. They provide several methods to automate updates. For more information, see [Automating updates to SSM Agent](#) in the *AWS Systems Manager User Guide*.

Updating the Amazon ECS agent on an external instance

On your external instances, the Amazon ECS container agent is updated by upgrading the `ecs-init` package. Updating the Amazon ECS agent doesn't interrupt the running tasks or services. Amazon ECS provides the `ecs-init` package and signature file in an Amazon S3 bucket in each Region. Beginning with `ecs-init` version 1.52.1-1, Amazon ECS provides separate `ecs-init` packages for use depending on the operating system and system architecture your external instance uses.

Use the following table to determine the `ecs-init` package that you should download based on the operating system and system architecture your external instance uses.

Note

You can determine which operating system and system architecture that your external instance uses by using the following commands.

```
cat /etc/os-release  
uname -m
```

Operating systems (architecture)	ecs-init package
CentOS 7 (x86_64)	<code>amazon-ecs-init-latest.x86_64.rpm</code>
CentOS 8 (x86_64)	
SUSE Enterprise Server 15 (x86_64)	
RHEL 7 (x86_64)	
RHEL 8 (x86_64)	
CentOS 7 (aarch64)	<code>amazon-ecs-init-latest.aarch64.rpm</code>
CentOS 8 (aarch64)	
RHEL 7 (aarch64)	
Debian 9 (x86_64)	<code>amazon-ecs-init-latest.amd64.deb</code>
Debian 10 (x86_64)	
Ubuntu 18 (x86_64)	
Ubuntu 20 (x86_64)	
Debian 9 (aarch64)	<code>amazon-ecs-init-latest.arm64.deb</code>
Debian 10 (aarch64)	
Ubuntu 18 (aarch64)	

Operating systems (architecture)	ecs-init package
Ubuntu 20 (aarch64)	

Follow these steps to update the Amazon ECS agent.

To update the Amazon ECS agent

1. Confirm the Amazon ECS agent version that you're running.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

2. Download the ecs-init package for your operating system and system architecture. Amazon ECS provides the ecs-init package file in an Amazon S3 bucket in each Region. Make sure that you replace the <region> identifier in the command with the Region name (for example, us-west-2) that you're geographically closest to.

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb
```

3. (Optional) Verify the validity of the ecs-init package file using the PGP signature.

- a. Download and install GnuPG. For more information about GNUpG, see the [GnuPG website](#). For Linux systems, install gpg using the package manager on your flavor of Linux.
- b. Retrieve the Amazon ECS PGP public key.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

- c. Download the ecs-init package signature. The signature is an ASCII detached PGP signature that's stored in a file with the .asc extension. Amazon ECS provides the signature file in an Amazon S3 bucket in each Region. Make sure that you replace the <region> identifier in the command with the Region name (for example, us-west-2) that you're geographically closest to.

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm.asc
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm.asc
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb.asc
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb.asc
```

- d. Verify the ecs-init package file using the key.

For the rpm packages

```
gpg --verify amazon-ecs-init.rpm.asc ./amazon-ecs-init.rpm
```

For the deb packages

```
gpg --verify amazon-ecs-init.deb.asc ./amazon-ecs-init.deb
```

The following is the expected output.

```
gpg: Signature made Fri 14 May 2021 09:31:36 PM UTC
gpg:                               using RSA key 50DECC4710E61AF
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: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. Install the ecs-init package.

For the rpm package on CentOS 7, CentOS 8, and RHEL 7

```
sudo yum install -y ./amazon-ecs-init.rpm
```

For the rpm package on SUSE Enterprise Server 15

```
sudo zypper install -y --allow-unsigned-rpm ./amazon-ecs-init.rpm
```

For the deb package

```
sudo dpkg -i ./amazon-ecs-init.deb
```

5. Restart the ecs service.

```
sudo systemctl restart ecs
```

6. Verify the Amazon ECS agent version was updated.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

Container instance draining

There might be times when you need to remove a container instance from your cluster; for example, to perform system updates, update the Docker daemon, or to scale down the cluster capacity. Amazon ECS provides the ability to transition a container instance to a DRAINING status. This is referred to as *container instance draining*. When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance.

Draining behavior for services

Any tasks that are part of a service that are in a PENDING state are stopped immediately. If there is available container instance capacity in the cluster, the service scheduler will start replacement tasks. If there isn't enough container instance capacity, a service event message will be sent indicating the issue.

Tasks that are part of a service on the container instance that are in a RUNNING state are transitioned to a STOPPED state. The service scheduler attempts to replace the tasks according to the service's deployment type and deployment configuration parameters, minimumHealthyPercent and maximumPercent. For more information, see [Amazon ECS Deployment types \(p. 472\)](#) and [Service definition parameters \(p. 847\)](#).

- If minimumHealthyPercent is below 100%, the scheduler can ignore desiredCount temporarily during task replacement. For example, desiredCount is four tasks, a minimum of 50% allows the scheduler to stop two existing tasks before starting two new tasks. If the minimum is 100%, the service scheduler can't remove existing tasks until the replacement tasks are considered healthy. If tasks for services that do not use a load balancer are in the RUNNING state, they are considered healthy. Tasks for services that use a load balancer are considered healthy if they are in the RUNNING state and the container instance they are hosted on is reported as healthy by the load balancer.

Important

If you use Spot Instances and minimumHealthyPercent is greater than or equal to 100%, then the service will not have enough time to replace the task before the Spot Instance terminates.

- The maximumPercent parameter represents an upper limit on the number of running tasks during task replacement, which allows you to define the replacement batch size. For example, if desiredCount of four tasks, a maximum of 200% starts four new tasks before stopping the four tasks to be drained (provided that the cluster resources required to do this are available). If the maximum is 100%, then replacement tasks can't start until the draining tasks have stopped.

Important

If both minimumHealthyPercent and maximumPercent are 100%, then the service can't remove existing tasks, and also cannot start replacement tasks. This prevents successful container instance draining and prevents making new deployments.

Draining behavior for standalone tasks

Any standalone tasks in the PENDING or RUNNING state are unaffected; you must wait for them to stop on their own or stop them manually. The container instance will remain in DRAINING status.

A container instance has completed draining when all tasks running on the instance transition to a STOPPED state. The container instance remains in a DRAINING state until it is activated again or deleted.

You can verify the state of the tasks on the container instance by using the [ListTasks](#) operation with the `containerInstance` parameter to get a list of tasks on the instance followed by a [DescribeTasks](#) operation with the Amazon Resource Name (ARN) or ID of each task to verify the task state.

When you are ready for the container instance to start hosting tasks again, you change the state of the container instance from DRAINING to ACTIVE. The Amazon ECS service scheduler will then consider the container instance for task placement again.

Draining container instances

The following steps can be used to set a container instance to draining using the new AWS Management Console.

You can also use the [UpdateContainerInstancesState](#) API action or the [update-container-instances-state](#) command to change the status of a container instance to DRAINING.

AWS Management Console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose a cluster that hosts your instances.
4. On the **Cluster : name** page, choose the **Infrastructure** tab. Then, under **Container instances** select the check box for each container instance you want to drain.
5. Choose **Actions, Drain**.

Amazon ECS container agent

The Amazon ECS container agent allows container instances to connect to your cluster. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs, but you can also install it on any Amazon EC2 instance that supports the Amazon ECS specification. The Amazon ECS container agent is supported on Amazon EC2 instances and external instances (on-premises server or VM). For more information about external instances, see [the section called “Updating the Amazon ECS agent” \(p. 351\)](#).

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

Note

For tasks using the Fargate launch type and platform version 1.3.0 and prior, the Amazon ECS container agent is installed on the AWS managed infrastructure used for these tasks. If you are only using tasks with the Fargate launch type, no additional configuration is needed and the content in this topic does not apply. For tasks using the Fargate and platform version 1.4.0 and later (for Linux) or 1.0.0 or later (for Windows), the Fargate container agent is used. For more information, see [AWS Fargate platform versions](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Topics

- [Installing the Amazon ECS container agent \(p. 356\)](#)
- [Amazon ECS Linux container agent versions \(p. 362\)](#)
- [Amazon EC2 Windows containers \(p. 363\)](#)
- [Updating the Amazon ECS container agent \(p. 364\)](#)
- [Amazon ECS container agent configuration \(p. 370\)](#)
- [Private registry authentication for container instances \(p. 371\)](#)
- [Automated task and image cleanup \(p. 375\)](#)
- [Amazon ECS container metadata file \(p. 376\)](#)
- [Amazon ECS task metadata endpoint \(p. 380\)](#)
- [Amazon ECS container agent endpoint \(p. 407\)](#)
- [Amazon ECS container agent introspection \(p. 410\)](#)
- [HTTP proxy configuration \(p. 412\)](#)
- [Using gMSAs for Windows Containers \(p. 416\)](#)
- [Using gMSAs for Linux Containers \(p. 421\)](#)
- [Updating the Amazon ECS container agent with the console \(p. 428\)](#)

Installing the Amazon ECS container agent

If your container instance was not launched using an Amazon ECS-optimized AMI, you can install the Amazon ECS container agent manually using one of the following procedures. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs and does not require installation.

- For Amazon Linux 2 instances, you can install the agent using the `amazon-linux-extras` command. For more information, see [Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance \(p. 357\)](#).
- For Amazon Linux AMI instances, you can install the agent using the Amazon YUM repo. For more information, see [Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance \(p. 357\)](#).

- For non-Amazon Linux instances, you can either download the agent from one of the regional S3 buckets or from Amazon Elastic Container Registry Public. If you download from one of the regional S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature. For more information, see [Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance \(p. 358\)](#).

Note

The systemd units for both Amazon ECS and Docker services have a directive to wait for cloud-init to finish before starting both services. The cloud-init process is not considered finished until your Amazon EC2 user data has finished running. Therefore, starting Amazon ECS or Docker via Amazon EC2 user data may cause a deadlock. To start the container agent using Amazon EC2 user data you can use `systemctl enable --now --no-block ecs.service`.

Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance

To install the Amazon ECS container agent on an Amazon Linux 2 EC2 instance using the `amazon-linux-extras` command, use the following steps.

To install the Amazon ECS container agent on an Amazon Linux 2 EC2 instance

- Launch an Amazon Linux 2 EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
- Connect to your instance.
- Disable the `docker` Amazon Linux extra repository. The `ecs` Amazon Linux extra repository ships with its own version of Docker, so the `docker` extra must be turned off to avoid any potential future conflicts. This ensures that you are always using the Docker version that Amazon ECS intends for you to use with a particular version of the container agent.

```
[ec2-user ~]$ sudo amazon-linux-extras disable docker
```

- Install and enable the `ecs` Amazon Linux extra repository.

```
[ec2-user ~]$ sudo amazon-linux-extras install -y ecs; sudo systemctl enable --now ecs
```

- (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see [the section called "Container agent introspection" \(p. 410\)](#).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Note

If you get no response, ensure that you associated the Amazon ECS container instance IAM role when launching the instance. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance

To install the Amazon ECS container agent on an Amazon Linux AMI EC2 instance using the Amazon YUM repo, use the following steps.

To install the Amazon ECS container agent on an Amazon Linux AMI EC2 instance

1. Launch an Amazon Linux AMI EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
2. Connect to your instance.
3. Install the `ecs-init` package. For more information about `ecs-init`, see the [source code on GitHub](#).

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

4. Start the Docker daemon.

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

Output:

```
Starting cgconfig service: [ OK ]  
Starting docker: [ OK ]
```

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

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

Output:

```
ecs start/running, process 2804
```

6. (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see [the section called "Container agent introspection" \(p. 410\)](#).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

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

To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance, you can download the agent from one of the regional S3 buckets and install it.

Note

When using a non-Amazon Linux AMI, your Amazon EC2 instance requires `cgroupfs` support for the `cgroup` driver in order for the Amazon ECS agent to support task level resource limits. For more information, see [Amazon ECS agent on GitHub](#).

The latest Amazon ECS container agent files, by Region, for each system architecture are listed below for reference.

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
us-east-2	US East (Ohio)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-east-1	US East (N. Virginia)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-west-1	US West (N. California)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-west-2	US West (Oregon)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-east-1	Asia Pacific (Hong Kong)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-northeast-1	Asia Pacific (Tokyo)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-northeast-2	Asia Pacific (Seoul)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-south-1	Asia Pacific (Mumbai)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-southeast-1	Asia Pacific (Singapore)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
ap-southeast-2	Asia Pacific (Sydney)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
ca-central-1	Canada (Central)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
eu-central-1	Europe (Frankfurt)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
eu-west-1	Europe (Ireland)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
eu-west-2	Europe (London)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
eu-west-3	Europe (Paris)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
sa-east-1	South America (São Paulo)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-gov-east-1	AWS GovCloud (US-East)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-gov-west-1	AWS GovCloud (US-West)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)

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

1. Launch an Amazon EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
2. Connect to your instance.
3. Install the latest version of Docker on your instance.

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

Note

The minimum Docker version for reliable metrics is Docker version v20.10.13 and newer, which is included in Amazon ECS-optimized AMI 20220607 and newer.

Amazon ECS agent versions 1.20.0 and newer have deprecated support for Docker versions older than 1.9.0.

```
docker --version
```

5. Download the appropriate Amazon ECS agent file for your operating system and system architecture and install it.

For deb architectures:

```
ubuntu:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.amd64.deb
ubuntu:~$ sudo dpkg -i amazon-ecs-init-latest.amd64.deb
```

For rpm architectures:

```
fedoras:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.x86_64.rpm
fedoras:~$ sudo yum localinstall -y amazon-ecs-init-latest.x86_64.rpm
```

6. (Optional) To register the instance with a cluster other than the default cluster, edit the /etc/ecs/ecs.config file and add the following contents. The following example specifies the MyCluster cluster.

```
ECS_CLUSTER=MyCluster
```

For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 370\)](#).

Note

You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#) and [Private registry authentication for tasks \(p. 195\)](#).

7. Start the ecs service.

```
ubuntu:~$ sudo systemctl start ecs
```

Running the Amazon ECS agent with host network mode

When running the Amazon ECS container agent, ecs-init will create the container agent container with the host network mode. This is the only supported network mode for the container agent container.

This allows you to block access to the [Amazon EC2 instance metadata service endpoint](#) (<http://169.254.169.254>) for the containers started by the container agent. This ensures that

containers cannot access IAM role credentials from the container instance profile and enforces that tasks use only the IAM task role credentials. For more information, see [Task IAM role \(p. 631\)](#).

This also makes it so the container agent doesn't contend for connections and network traffic on the `docker0` bridge.

Amazon ECS Linux container agent versions

Each Amazon ECS container agent version supports a different feature set and provides bug fixes from previous versions. When possible, we always recommend using the latest version of the Amazon ECS container agent. To update your container agent to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

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

Important

The minimum Docker version for reliable metrics is Docker version v20.10.13 and newer, which is included in Amazon ECS-optimized AMI 20220607 and newer.

Amazon ECS agent versions 1.20.0 and newer have deprecated support for Docker versions older than 1.9.0.

Amazon ECS-optimized AMI

The Linux variants of the Amazon ECS-optimized AMI use the Amazon Linux 2 AMI as their base. The Amazon Linux 2 source AMI name for each variant can be retrieved by querying the Systems Manager Parameter Store API. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 259\)](#). The Amazon Linux 2 AMI release notes are available as well. For more information, see [Amazon Linux 2 release notes](#).

Launching your container instances from the most recent Amazon ECS-optimized Amazon Linux 2 AMI ensures that you receive the current container agent version. To launch a container instance with the latest Amazon ECS-optimized Amazon Linux 2 AMI, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Amazon ECS provides a changelog for the Linux variant of the Amazon ECS-optimized AMI on GitHub. For more information, see [Changelog](#).

Using other Linux Operating Systems

To install the latest version of the Amazon ECS container agent on another operating system, see [Installing the Amazon ECS container agent \(p. 356\)](#). The table in [Amazon ECS-optimized AMI changelog \(p. 258\)](#) shows the Docker version that is tested on Amazon Linux 2 for each agent version.

Additional information

The following pages provide additional information about the changes:

- [Amazon ECS Agent changelog](#) on GitHub
- The source code for the `ecs-init` application and the scripts and configuration for packaging the agent are now part of the agent repository. For older versions of `ecs-init` and packaging, see [Amazon ecs-init changelog](#) on GitHub

- [Docker Engine release notes](#) in the Docker documentation
- [NVIDIA Driver Documentation](#) in the NVIDIA documentation

Amazon EC2 Windows containers

Amazon ECS now supports Windows containers on container instances that are launched with the Amazon ECS-optimized Windows Server AMI and on AWS Fargate. For more information about Windows containers on AWS Fargate, see [Windows containers on AWS Fargate considerations](#).

Windows container instances use their own version of the Amazon ECS container agent. On the Amazon ECS-optimized Windows Server AMI, the Amazon ECS container agent runs as a service on the host. Unlike the Linux platform, the agent doesn't run inside a container because it uses the host's registry and the named pipe at `\.\pipe\docker_engine` to communicate with the Docker daemon.

The source code for the Amazon ECS container agent is [available on GitHub](#). We encourage you to submit pull requests for changes that you would like to have included. However, we do not currently provide support for running modified copies of this software. You can view open issues for Amazon ECS and Windows containers on our [GitHub issues page](#).

Amazon ECS vends AMIs that are optimized for Windows containers in the following variants. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

- **Amazon ECS-optimized Windows Server 2022 Full AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2022 Core AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2019 Full AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2019 Core AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2004 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Important

The Amazon ECS-optimized Windows Server 2004 Core AMI is being deprecated. No new versions of this AMI will be released.

- **Amazon ECS-optimized Windows Server 1909 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Important

The Amazon ECS-optimized Windows Server 1909 Core AMI is being deprecated. No new versions of this AMI will be released.

- **Amazon ECS-optimized Windows Server 2016 Full AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Windows Server 2022, Windows Server 2019, and Windows Server 2016 are Long-Term Servicing Channel (LTSC) releases. Windows Server 20H2 is a Semi-Annual Channel (SAC) release. For more information, see [Windows Server release information](#).

Topics

- [Windows container caveats \(p. 364\)](#)
- [Getting started with Windows containers \(p. 364\)](#)

Windows container caveats

Here are some things you should know about Amazon EC2 Windows containers and Amazon ECS.

- Windows containers can't run on Linux container instances, and the opposite is also the case. For better task placement for Windows and Linux tasks, keep Windows and Linux container instances in separate clusters and only place Windows tasks on Windows clusters. You can ensure that Windows task definitions are only placed on Windows instances by setting the following placement constraint: `memberOf(ecs.os-type=='windows')`.
- Windows containers are supported for tasks that use the EC2 and Fargate launch types.
- Windows containers and container instances can't support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they aren't supported at all, and others behave differently on Windows than they do on Linux. For more information, see [Amazon EC2 Windows task definition considerations \(p. 124\)](#).
- For the IAM roles for tasks feature, you need to configure your Windows container instances to allow the feature at launch. Your containers must run some provided PowerShell code when they use the feature. For more information, see [Additional configuration for Windows IAM roles for tasks \(p. 637\)](#).
- The IAM roles for tasks feature uses a credential proxy to provide credentials to the containers. This credential proxy occupies port 80 on the container instance, so if you use IAM roles for tasks, port 80 is not available for tasks. For web service containers, you can use an Application Load Balancer and dynamic port mapping to provide standard HTTP port 80 connections to your containers. For more information, see [Service load balancing \(p. 486\)](#).
- The Windows Server Docker images are large (9 GiB). So, your Windows container instances require more storage space than Linux container instances.
- To run a Windows container on a Windows Server, the container's base image OS version must match that of the host. For more information, see [Windows container version compatibility](#) on the Microsoft documentation website. If your cluster runs multiple Windows versions, you can ensure that a task is placed on an EC2 instance running on the same version by using the placement constraint: `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`. For more information, see [the section called "Retrieving Amazon ECS-Optimized AMI metadata" \(p. 308\)](#).

Getting started with Windows containers

Work through a tutorial that guides you through getting Windows containers running on Amazon ECS with the Amazon ECS-optimized Windows Server AMI in the AWS Management Console at [Getting started with the console using Windows on Amazon EC2 \(p. 26\)](#).

Updating the Amazon ECS container agent

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

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

Topics

- [Checking the Amazon ECS container agent version \(p. 365\)](#)
- [Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI \(p. 366\)](#)

- [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 368\)](#)

Checking the Amazon ECS container agent version

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

Amazon ECS console;

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.
4. On the **Cluster : name** page, choose the **Infrastructure** tab.
5. Under **Container instances**, note the **Agent version** column for your container instances. If the container instance does not contain the latest version of the container agent, the console alerts you with a message and flags the outdated agent version.

If your agent version is outdated, you can update your container agent with the following procedures:

- If your container instance is running an Amazon ECS-optimized AMI, see [Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI \(p. 366\)](#).
- If your container instance is not running an Amazon ECS-optimized AMI, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 368\)](#).

Important

To update the Amazon ECS agent version from versions before v1.0.0 on your Amazon ECS-optimized AMI, we recommend that you terminate your current container instance and launch a new instance with the most recent AMI version. Any container instances that use a preview version should be retired and replaced with the most recent AMI. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

Amazon ECS container agent introspection API

You can also use the to check the agent Amazon ECS container agent introspection API version from the container instance itself. For more information, see [Amazon ECS container agent introspection \(p. 410\)](#).

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

1. Log in to your container instance via SSH.
2. Query the introspection API.

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python -mjson.tool
```

Note

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

Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI

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

- Terminate the container instance and launch the latest version of the Amazon ECS-optimized Amazon Linux 2 AMI (either manually or by updating your Auto Scaling launch configuration with the latest AMI). This provides a fresh container instance with the most current tested and validated versions of Amazon Linux, Docker, `ecs-init`, and the Amazon ECS container agent. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).
- Connect to the instance with SSH and update the `ecs-init` package (and its dependencies) to the latest version. This operation provides the most current tested and validated versions of Docker and `ecs-init` that are available in the Amazon Linux repositories and the latest version of the Amazon ECS container agent. For more information, see [To update the ecs-init package on an Amazon ECS-optimized AMI \(p. 366\)](#).
- Update the container agent with the `UpdateContainerAgent` API operation, either through the console or with the AWS CLI or AWS SDKs. For more information, see [Updating the Amazon ECS container agent with the UpdateContainerAgent API operation \(p. 366\)](#).

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

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

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

```
sudo yum update -y ecs-init
```

Note

The `ecs-init` package and the Amazon ECS container agent are updated immediately. However, newer versions of Docker are not loaded until the Docker daemon is restarted. Restart either by rebooting the instance, or by running the following commands on your instance:

- Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart docker
```

- Amazon ECS-optimized Amazon Linux AMI:

```
sudo service docker restart && sudo start ecs
```

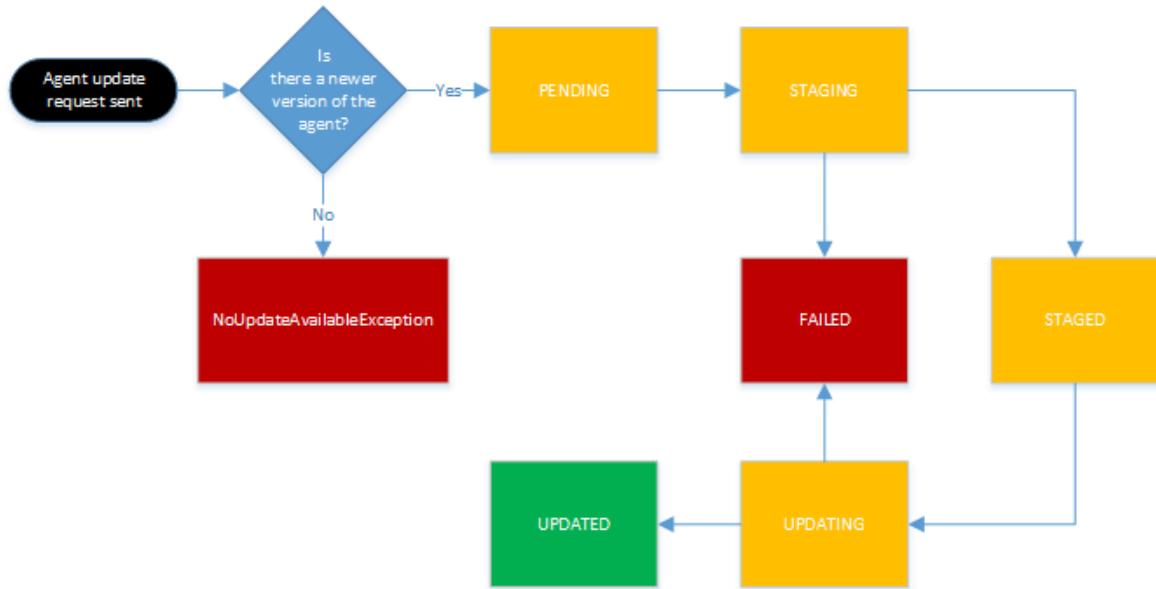
Updating the Amazon ECS container agent with the `UpdateContainerAgent` API operation

Important

The `UpdateContainerAgent` API is only supported on Linux variants of the Amazon ECS-optimized AMI, with the exception of the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI. For container instances using the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, update

the `ecs-init` package to update the agent. For container instances that are running other operating systems, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 368\)](#). If you are using Windows container instances, we recommend that you launch new container instances to update the agent version in your Windows clusters.

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



The stages in the update process shown above are as follows:

PENDING

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

STAGING

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

STAGED

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

UPDATING

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

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

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

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.

2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster.
4. On the **Cluster : name** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the instances to update, and then choose **Actions, Update agent**.

Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs)

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

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

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

```
ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
```

Output:

```
"ECS_DATADIR=/data",
```

Important

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

3. Stop the Amazon ECS container agent.

```
ubuntu:~$ docker stop ecs-agent
```

4. Delete the agent container.

```
ubuntu:~$ docker rm ecs-agent
```

5. Ensure that the /etc/ecs directory and the Amazon ECS container agent configuration file exist at /etc/ecs/ecs.config.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

6. Edit the /etc/ecs/ecs.config file and ensure that it contains at least the following variable declarations. If you do not want your container instance to register with the default cluster, specify your cluster name as the value for ECS_CLUSTER.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 370\)](#).

Note

You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#) and [Private registry authentication for tasks \(p. 195\)](#).

7. Pull the latest Amazon ECS container agent image from Amazon Elastic Container Registry Public.

```
ubuntu:~$ docker pull public.ecr.aws/ecs/amazon-ecs-agent:latest
```

Output:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

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

Note

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

The following example of the agent run command is broken into separate lines to show each option. For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 370\)](#).

Important

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

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

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

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

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
```

```
--volume=/etc/ecs:/etc/ecs \
--volume=/etc/ecs:/etc/ecs/pki \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

Note

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

Amazon ECS container agent configuration

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

If your container instance was launched with a Linux variant of the Amazon ECS-optimized AMI, you can set these environment variables in the `/etc/ecs/ecs.config` file and then restart the agent. You can also write these configuration variables to your container instances with Amazon EC2 user data at launch time. For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#).

If your container instance was launched with a Windows variant of the Amazon ECS-optimized AMI, you can set these environment variables with the PowerShell `SetEnvironmentVariable` command and then restart the agent. For more information, see [Run commands on your Windows instance at launch](#) in the [Amazon EC2 User Guide for Windows Instances](#) and [the section called "Bootstrap Container Instances" \(p. 333\)](#).

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

Available Parameters

For information about the available Amazon ECS container agent configuration parameters, see [Amazon ECS Container Agent](#) on GitHub.

Storing container instance configuration in Amazon S3

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

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance

configuration at launch. You can store a copy of your `ecs.config` file in a private bucket. You can then use Amazon EC2 user data to install the AWS CLI and copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

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

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and select the IAM role to use for your container instances. This role is likely titled `ecsInstanceRole`. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
3. Under **Managed Policies**, choose **Attach Policy**.
4. To narrow the policy results, on the **Attach Policy** page, for **Filter**, type `S3`.
5. Select the box to the left of the `AmazonS3ReadOnlyAccess` policy and choose **Attach Policy**.

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

1. Create an `ecs.config` file with valid Amazon ECS agent configuration variables using the following format. This example configures private registry authentication. For more information, see [Private registry authentication for tasks \(p. 195\)](#).

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

Note

For a full list of available Amazon ECS agent configuration variables, see [Amazon ECS Container Agent](#) on GitHub.

2. To store your configuration file, create a private bucket in Amazon S3. For more information, see [Create a Bucket](#) in the *Amazon Simple Storage Service User Guide*.
3. Upload the `ecs.config` file to your S3 bucket. For more information, see [Add an Object to a Bucket](#) in the *Amazon Simple Storage Service User Guide*.

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

1. Complete the earlier procedures in this section to allow read-only Amazon S3 access to your container instances and store an `ecs.config` file in a private S3 bucket.
2. Launch new container instances by following the steps in [Launching an Amazon ECS Linux container instance \(p. 272\)](#). In [Step 6.g \(p. 277\)](#), use the following example script that installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`.

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

Private registry authentication for container instances

The Amazon ECS container agent can authenticate with private registries, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions. This feature is only supported by tasks using the EC2 launch type.

Another method of enabling private registry authentication uses AWS Secrets Manager to store your private registry credentials securely and then reference them in your container definition. This allows your tasks to use images from private repositories. This method supports tasks using either the EC2 or Fargate launch types. For more information, see [Private registry authentication for tasks \(p. 195\)](#).

The Amazon ECS container agent looks for two environment variables when it launches:

- `ECS_ENGINE_AUTH_TYPE`, which specifies the type of authentication data that is being sent.
- `ECS_ENGINE_AUTH_DATA`, which contains the actual authentication credentials.

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

Important

We do not recommend that you inject these authentication environment variables at instance launch with Amazon EC2 user data or pass them with the `--env` option to the `docker run` command. These methods are not appropriate for sensitive data, such as authentication credentials. For information about safely adding authentication credentials to your container instances, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#).

Authentication formats

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

`dockercfg` authentication format

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

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

Output:

```
{  
  "auths": {  
    "https://index.docker.io/v1/": {  
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"  
    }  
  }  
}
```

Important

Newer versions of Docker create a configuration file as shown above with an outer `auths` object. The Amazon ECS agent only supports `dockercfg` authentication data that is in the below format, without the `auths` object. If you have the `jq` utility installed, you can extract this data with the following command: `cat ~/.docker/config.json | jq .auths`

```
cat ~/.docker/config.json | jq .auths
```

Output:

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

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

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

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"}, "repo.example-02.com":
{"auth":"fQ172MzEXAMPLEoF7225DU0j","email":"email@example-02.com"}}
```

docker authentication format

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

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

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

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

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
>{"username":"my_name","password":"my_password","email":"email@example-01.com"}, "repo.example-02.com":
>{"username":"another_name","password":"another_password","email":"email@example-02.com"}}
```

Turning on private registries

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

To enable private registries in the Amazon ECS-optimized AMI

1. Log in to your container instance using SSH.
2. Open the `/etc/ecs/ecs.config` file and add the `ECS_ENGINE_AUTH_TYPE` and `ECS_ENGINE_AUTH_DATA` values for your registry and account:

```
sudo vi /etc/ecs/ecs.config
```

This example authenticates a Docker Hub user account:

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

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

```
docker inspect ecs-agent | grep ECS_DATADIR
```

Output:

```
"ECS_DATADIR=/data",
```

Important

If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before stopping the agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can stop and start them while tasks are running without issues. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).

4. Stop the `ecs` service:

```
sudo stop ecs
```

Output:

```
ecs stop/waiting
```

5. Restart the `ecs` service.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

6. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see [the section called “Container agent introspection” \(p. 410\)](#).

```
curl http://localhost:51678/v1/metadata
```

Automated task and image cleanup

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

Note

The Amazon ECS agent image pull behavior can be customized using the `ECS_IMAGE_PULL_BEHAVIOR` parameter. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

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

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

Note

The automated image cleanup feature requires at least version 1.13.0 of the Amazon ECS container agent. To update your agent to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

Tunable parameters

The following agent configuration variables are available to tune your automated task and image cleanup experience. For more information about how to set these variables on your container instances, see [Amazon ECS container agent configuration \(p. 370\)](#).

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

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

`ECS_DISABLE_IMAGE_CLEANUP`

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

`ECS_IMAGE_CLEANUP_INTERVAL`

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

`ECS_IMAGE_MINIMUM_CLEANUP AGE`

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

ECS_NUM_IMAGES_DELETE_PER_CYCLE

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

Cleanup workflow

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

Amazon ECS container metadata file

Beginning with version 1.15.0 of the Amazon ECS container agent, various container metadata is available within your containers or the host container instance. By enabling this feature, you can query the information about a task, container, and container instance from within the container or the host container instance. The metadata file is created on the host instance and mounted in the container as a Docker volume and therefore is not available when a task is hosted on AWS Fargate.

The container metadata file is cleaned up on the host instance when the container is cleaned up. You can adjust when this happens with the ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION container agent variable. For more information, see [Automated task and image cleanup \(p. 375\)](#).

Topics

- [Turning on container metadata \(p. 376\)](#)
- [Container metadata file locations \(p. 377\)](#)
- [Container metadata file format \(p. 377\)](#)

Turning on container metadata

You can turn on container metadata at the container instance level by setting the ECS_ENABLE_CONTAINER_METADATA container agent variable to true. You can set this variable in the /etc/ecs/ecs.config configuration file and restart the agent. You can also set it as a Docker environment variable at runtime when the agent container is started. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

If the ECS_ENABLE_CONTAINER_METADATA is set to true when the agent starts, metadata files are created for any containers created from that point forward. The Amazon ECS container agent cannot create metadata files for containers that were created before the ECS_ENABLE_CONTAINER_METADATA container agent variable was set to true. To ensure that all containers receive metadata files, you should set this agent variable at container instance launch. The following is an example user data script that will set this variable as well as register your container instance with your cluster.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Container metadata file locations

By default, the container metadata file is written to the following host and container paths.

- **For Linux instances:**

- Host path: /var/lib/ecs/data/metadata/*cluster_name*/*task_id*/*container_name*/ecs-container-metadata.json

Note

The Linux host path assumes that the default data directory mount path (/var/lib/ecs/data) is used when the agent is started. If you are not using an Amazon ECS-optimized AMI (or the ecs-init package to start and maintain the container agent), be sure to set the ECS_HOST_DATA_DIR agent configuration variable to the host path where the container agent's state file is located. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

- Container path: /opt/ecs/metadata/*random_ID*/ecs-container-metadata.json

- **For Windows instances:**

- Host path: C:\ProgramData\Amazon\ECS\data\metadata\i*task_id*\i*container_name*\ecs-container-metadata.json
- Container path: C:\ProgramData\Amazon\ECS\metadata\i*random_ID*\ecs-container-metadata.json

However, for easy access, the container metadata file location is set to the ECS_CONTAINER_METADATA_FILE environment variable inside the container. You can read the file contents from inside the container with the following command:

- **For Linux instances:**

```
cat $ECS_CONTAINER_METADATA_FILE
```

- **For Windows instances (PowerShell):**

```
Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
```

Container metadata file format

The following information is stored in the container metadata JSON file.

Cluster

The name of the cluster that the container's task is running on.

ContainerInstanceARN

The full Amazon Resource Name (ARN) of the host container instance.

TaskARN

The full Amazon Resource Name (ARN) of the task that the container belongs to.

TaskDefinitionFamily

The name of the task definition family the container is using.

TaskDefinitionRevision

The task definition revision the container is using.

ContainerID

The Docker container ID (and not the Amazon ECS container ID) for the container.

ContainerName

The container name from the Amazon ECS task definition for the container.

DockerContainerName

The container name that the Docker daemon uses for the container (for example, the name that shows up in `docker ps` command output).

ImageID

The SHA digest for the Docker image used to start the container.

ImageName

The image name and tag for the Docker image used to start the container.

PortMappings

Any port mappings associated with the container.

ContainerPort

The port on the container that is exposed.

HostPort

The port on the host container instance that is exposed.

BindIp

The bind IP address that is assigned to the container by Docker. This IP address is only applied with the bridge network mode, and it is only accessible from the container instance.

Protocol

The network protocol used for the port mapping.

Networks

The network mode and IP address for the container.

NetworkMode

The network mode for the task to which the container belongs.

IPv4Addresses

The IP addresses associated with the container.

Important

If your task is using the `awsvpc` network mode, the IP address of the container will not be returned. In this case, you can retrieve the IP address by reading the `/etc/hosts` file with the following command:

```
tail -1 /etc/hosts | awk '{print $1}'
```

MetadataFileStatus

The status of the metadata file. When the status is READY, the metadata file is current and complete. If the file is not ready yet (for example, the moment the task is started), a truncated version of the file format is available. To avoid a likely race condition where the container has started, but the metadata has not yet been written, you can parse the metadata file and wait for this parameter to be set to READY before depending on the metadata. This is usually available in less than 1 second from when the container starts.

AvailabilityZone

The Availability Zone the host container instance resides in.

HostPrivateIPv4Address

The private IP address for the task the container belongs to.

HostPublicIPv4Address

The public IP address for the task the container belongs to.

Example Amazon ECS container metadata file (READY)

The following example shows a container metadata file in the READY status.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/2b88376d-aba3-4950-9ddf-bcb0f388a40c",
    "TaskDefinitionFamily": "console-sample-app-static",
    "TaskDefinitionRevision": "1",
    "ContainerID": "aec2557997f4eed9b280c2efd7afcccdedfda4ac399f7480cae870fcf7e163fd",
    "ContainerName": "simple-app",
    "DockerContainerName": "/ecs-console-sample-app-static-1-simple-app-e4e8e495e8baa5de1a00",
    "ImageID": "sha256:2ae34abc2ed0a22e280d17e13f9c01aa725688b09b7a1525d1a2750e2c0d1de",
    "ImageName": "httpd:2.4",
    "PortMappings": [
        {
            "ContainerPort": 80,
            "HostPort": 80,
            "BindIp": "0.0.0.0",
            "Protocol": "tcp"
        }
    ],
    "Networks": [
        {
            "NetworkMode": "bridge",
            "IPv4Addresses": [
                "192.0.2.0"
            ]
        }
    ],
    "MetadataFileStatus": "READY",
    "AvailabilityZone": "us-east-1b",
    "HostPrivateIPv4Address": "192.0.2.0",
    "HostPublicIPv4Address": "203.0.113.0"
}
```

Example Incomplete Amazon ECS container metadata file (not yet READY)

The following example shows a container metadata file that has not yet reached the READY status. The information in the file is limited to a few parameters that are known from the task definition. The container metadata file should be ready within 1 second after the container starts.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/default/1f73d099-b914-411c-a9ff-81633b7741dd",
```

```
"TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/  
d90675f8-1a98-444b-805b-3d9cabb6fc4",  
    "ContainerName": "metadata"  
}
```

Amazon ECS task metadata endpoint

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Amazon ECS task metadata endpoint](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The Amazon ECS container agent provides a method to retrieve various task metadata and [Docker stats](#). This is referred to as the task metadata endpoint. The following versions are available:

- Task metadata endpoint version 4 – Provides a variety of metadata and Docker stats to containers. Can also provide network rate data. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.39.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task metadata endpoint version 4 \(p. 380\)](#).
- Task metadata endpoint version 3 – Provides a variety of metadata and Docker stats to containers. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.21.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task Metadata Endpoint version 3 \(p. 397\)](#).
- Task metadata endpoint version 2 – Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.17.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task Metadata Endpoint version 2 \(p. 402\)](#).

If your Amazon ECS task is hosted on Amazon EC2, you can also access task host metadata using the [Instance Metadata Service \(IMDS\) endpoint](#). The following command, when run from within the instance hosting the task, lists the ID of the host instance.

```
curl http://169.254.169.254/latest/meta-data/instance-id
```

The information you can obtain from the endpoint is divided into categories such as *instance-id*. For more information on the different categories of host instance metadata you can obtain using the endpoint, see [Instance metadata categories](#).

Task metadata endpoint version 4

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Task metadata endpoint version 4](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The Amazon ECS container agent injects an environment variable into each container, referred to as the *task metadata endpoint* which provides various task metadata and [Docker stats](#) to the container.

The task metadata and network rate stats are sent to CloudWatch Container Insights and can be viewed in the AWS Management Console. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).

Note

Amazon ECS provides earlier versions of the task metadata endpoint. To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added

to the version 4 output. We will not remove any existing metadata or change the metadata field names.

Enabling the task metadata endpoint

The environment variable is injected by default into the containers of Amazon ECS tasks launched on Amazon EC2 Linux instances that are running at least version 1.39.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).

Note

You can add support for this feature on Amazon EC2 instances using older versions of the Amazon ECS container agent by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).

Task metadata endpoint version 4 paths

The following task metadata endpoint paths are available to containers.

`${ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata for the container.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

This path returns metadata 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 \(p. 382\)](#).

`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`

This path returns the metadata for the task included in the /task endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API. Any errors received when retrieving the tag metadata will be included in the `Errors` field in the response.

Note

The `Errors` field is only in the response for tasks hosted on Amazon EC2 Linux instances running at least version 1.50.0 of the container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0.

This endpoint requires the `ecs.ListTagsForResource` permission.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

This path returns Docker stats for the specific container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

For Amazon ECS tasks that use the awsvpc or bridge network modes hosted on Amazon EC2 Linux instances running at least version 1.43.0 of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

This path returns Docker stats for all of the containers associated with the task. This can be used by sidecar containers to extract network metrics. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

For Amazon ECS tasks that use the awsvpc or bridge network modes hosted on Amazon EC2 Linux instances running at least version 1.43.0 of the container agent, there will be additional network

rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

Task metadata JSON response

The following information is returned from the task metadata endpoint (`/${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response. This includes metadata associated with the task in addition to the metadata for each container within the task.

Cluster

The short name of the Amazon ECS cluster to which the task belongs.

ServiceName

The name of the service to which the task belongs. ServiceName will appear for Amazon EC2 and Amazon ECS Anywhere container instances if the task is associated with a service.

Note

The ServiceName metadata is only included when using Amazon ECS container agent version 1.63.1 or later.

VPCID

The VPC ID of the Amazon EC2 container instance. This field only appears for Amazon EC2 instances.

Note

The VPCID metadata is only included when using Amazon ECS container agent version 1.63.1 or later.

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.

Limits

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

PullStartedAt

The timestamp for when the first container image pull began.

PullStoppedAt

The timestamp for when the last container image pull finished.

AvailabilityZone

The Availability Zone the task is in.

Note

The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 (Windows).

LaunchType

The launch type the task is using. When using cluster capacity providers, this indicates whether the task is using Fargate or EC2 infrastructure.

Note

This LaunchType metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later (Linux) or 1.0.0 or later (Windows).

Containers

A list of container metadata for each container associated with the task.

DockerId

The Docker ID for the container.

When you use Fargate, the id is a 32-digit hex followed by a 10 digit number.

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 (expressed in CPU units) 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.

LogDriver

The log driver the container is using.

Note

This LogDriver metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

LogOptions

The log driver options defined for the container.

Note

This LogOptions metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

ContainerARN

The full Amazon Resource Name (ARN) of the container.

Note

This ContainerARN metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

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.

ExecutionStoppedAt

The time stamp for when the tasks DesiredStatus moved to STOPPED. This occurs when an essential container moves to STOPPED.

Examples

The following examples show example outputs from each of the task metadata endpoints.

Example container metadata response

When querying the \${ECS_CONTAINER_METADATA_URI_V4} endpoint you are returned only metadata about the container itself. The following is an example output.

```
{  
  "DockerId": "ea32192c8553fbff06c9340478a2ff089b2bb5646fb718b4ee206641c9086d66",  
  "Name": "curl",  
  "DockerName": "ecs-curltest-24-curl-cca48e8dcadd97805600",
```

```

    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/8f03e41243824aea923aca126495f665",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "24"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:15:07.620912337Z",
    "StartedAt": "2020-10-02T00:15:08.062559351Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/8f03e41243824aea923aca126495f665"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/0206b271-b33f-47ab-86c6-a0ba208a70a9",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.100"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:9e:32:c7:48:85",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-100.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}

```

Example task metadata response

When querying the \${ECS_CONTAINER_METADATA_URI_V4}/task endpoint you are returned metadata about the task the container is part of in addition to the metadata for each container within the task. The following is an example output.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "LaunchType": "EC2",
    "Containers": [

```

```
{
    "DockerId": "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
    "Name": "~internal~ecs~pause",
    "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
    "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-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RESOURCES_PROVISIONED",
    "KnownStatus": "RESOURCES_PROVISIONED",
    "Limits": {
        "CPU": 0,
        "Memory": 0
    },
    "CreatedAt": "2020-10-02T00:43:05.602352471Z",
    "StartedAt": "2020-10-02T00:43:06.076707576Z",
    "Type": "CNI_PAUSED",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
},
{
    "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
    "Name": "curl",
    "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:43:06.326590752Z",
    "StartedAt": "2020-10-02T00:43:06.767535449Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream-prefix": "curltest"
    }
}
```

```

        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIPv4Address": "10.0.2.1/24"
        }
    ]
}

```

Example task with tags metadata response

When querying the \${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags endpoint you are returned metadata about the task, including the task and container instance tags. The following is an example output.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "TaskTags": {
        "tag-use": "task-metadata-endpoint-test"
    },
    "ContainerInstanceTags": {
        "tag_key": "tag_value"
    },
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId": "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "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-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED"
        }
    ]
}
```

```

    "Limits": {
        "CPU": 0,
        "Memory": 0
    },
    "CreatedAt": "2020-10-02T00:43:05.602352471Z",
    "StartedAt": "2020-10-02T00:43:06.076707576Z",
    "Type": "CNI_PAUSE",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
},
{
    "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
    "Name": "curl",
    "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:43:06.326590752Z",
    "StartedAt": "2020-10-02T00:43:06.767535449Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/abb51bdd-11b4-467f-8f6c-adcf1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}

```

```

        ]
    }
}
```

Example task with tags with an error metadata response

When querying the `$(ECS_CONTAINER_METADATA_URI_V4)/taskWithTags` endpoint you are returned metadata about the task, including the task and container instance tags. If there is an error retrieving the tagging data, the error is returned in the response. The following is an example output for when the IAM role associated with the container instance doesn't have the `ecs>ListTagsForResource` permission allowed.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "Errors": [
        {
            "ErrorField": "ContainerInstanceTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform: ecs>ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:container-instance/default/2dd1b186f39845a584488d2ef155c131",
            "StatusCode": 400,
            "RequestId": "cd597ef0-272b-4643-9bd2-1de469870fa6",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:container-instance/default/2dd1b186f39845a584488d2ef155c131"
        },
        {
            "ErrorField": "TaskTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform: ecs>ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:task/default/9ef30e4b7aa44d0db562749cff4983f3",
            "StatusCode": 400,
            "RequestId": "862c5986-6cd2-4aa6-87cc-70be395531e1",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:task/default/9ef30e4b7aa44d0db562749cff4983f3"
        }
    ],
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId": "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "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-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",

```

```

        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RESOURCES_PROVISIONED",
    "KnownStatus": "RESOURCES_PROVISIONED",
    "Limits": {
        "CPU": 0,
        "Memory": 0
    },
    "CreatedAt": "2020-10-02T00:43:05.602352471Z",
    "StartedAt": "2020-10-02T00:43:06.076707576Z",
    "Type": "CNI_PAUSE",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
},
{
    "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
    "Name": "curl",
    "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:43:06.326590752Z",
    "StartedAt": "2020-10-02T00:43:06.767535449Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0
        }
    ]
}

```

```

        "MACAddress": "0e:10:e2:01:bd:91",
        "IPv4SubnetCIDRBlock": "10.0.2.0/24",
        "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
        "SubnetGatewayIpv4Address": "10.0.2.1/24"
    }
}
]
}
}

```

Example container stats response

When querying the ``${ECS_CONTAINER_METADATA_URI_V4}/stats` endpoint you are returned network metrics for the container. For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

The following is an example output from an Amazon ECS task on Amazon EC2 that uses the `bridge` network mode.

```
{
  "read": "2020-10-02T00:51:13.410254284Z",
  "preread": "2020-10-02T00:51:12.406202398Z",
  "pids_stats": {
    "current": 3
  },
  "blkio_stats": {
    "io_service_bytes_recursive": [
      ],
      "io_serviced_recursive": [
        ],
        "io_queue_recursive": [
          ],
          "io_service_time_recursive": [
            ],
            "io_wait_time_recursive": [
              ],
              "io_merged_recursive": [
                ],
                "io_time_recursive": [
                  ],
                  "sectors_recursive": [
                    ]
      },
      "num_procs": 0,
      "storage_stats": {
        },
        "cpu_stats": {
          "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
              182359190,
              178608875
            ]
          }
        }
}
```

```
        ],
        "usage_in_kernelmode": 40000000,
        "usage_in_usermode": 290000000
    },
    "system_cpu_usage": 13939680000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 360968065,
        "percpu_usage": [
            182359190,
            178608875
        ],
        "usage_in_kernelmode": 40000000,
        "usage_in_usermode": 290000000
    },
    "system_cpu_usage": 13937670000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 1806336,
    "max_usage": 6299648,
    "stats": {
        "active_anon": 606208,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 134217728,
        "hierarchical_memsw_limit": 268435456,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 4185,
        "pgmajfault": 0,
        "pgpgin": 2926,
        "pgpgout": 2778,
        "rss": 606208,
        "rss_huge": 0,
        "total_active_anon": 606208,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 4185,
        "total_pgmajfault": 0,
        "total_pgpgin": 2926,
        "total_pgpgout": 2778,
        "total_rss": 606208,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    }
}
```

```

        },
        "limit": 134217728
    },
    "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
    "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
    "networks": {
        "eth0": {
            "rx_bytes": 84,
            "rx_packets": 2,
            "rx_errors": 0,
            "rx_dropped": 0,
            "tx_bytes": 84,
            "tx_packets": 2,
            "tx_errors": 0,
            "tx_dropped": 0
        }
    },
    "network_rate_stats": {
        "rx_bytes_per_sec": 0,
        "tx_bytes_per_sec": 0
    }
}
}

```

Example task stats response

When querying the \${ECS_CONTAINER_METADATA_URI_V4}/task/stats endpoint you are returned network metrics about the task the container is part of. The following is an example output.

```
{
    "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854": {
        "read": "2020-10-02T00:51:32.51467703Z",
        "preread": "2020-10-02T00:51:31.50860463Z",
        "pids_stats": {
            "current": 1
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [
                ...
            ],
            "io_serviced_recursive": [
                ...
            ],
            "io_queue_recursive": [
                ...
            ],
            "io_service_time_recursive": [
                ...
            ],
            "io_wait_time_recursive": [
                ...
            ],
            "io_merged_recursive": [
                ...
            ],
            "io_time_recursive": [
                ...
            ],
            "sectors_recursive": [
                ...
            ]
        },
        "num_procs": 0,
        "storage_stats": {
            ...
        }
    }
}
```

```
},
"cpu_stats": {
    "cpu_usage": {
        "total_usage": 177232665,
        "percpu_usage": [
            13376224,
            163856441
        ],
        "usage_in_kernelmode": 0,
        "usage_in_usermode": 160000000
    },
    "system_cpu_usage": 13977820000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 177232665,
        "percpu_usage": [
            13376224,
            163856441
        ],
        "usage_in_kernelmode": 0,
        "usage_in_usermode": 160000000
    },
    "system_cpu_usage": 13975800000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 532480,
    "max_usage": 6279168,
    "stats": {
        "active_anon": 40960,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 9223372036854771712,
        "hierarchical_memsw_limit": 9223372036854771712,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 2033,
        "pgmajfault": 0,
        "ppggin": 1734,
        "ppgout": 1724,
        "rss": 40960,
        "rss_huge": 0,
        "total_active_anon": 40960,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 2033,
        "total_pgmajfault": 0,
        "total_ppggin": 1734,
        "total_ppgout": 1724
    }
}
```

```

        "total_pgpgout": 1724,
        "total_rss": 40960,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    },
    "limit": 4073377792
},
"name": "/ecs-curltest-26-internalecspause-a6bcc3dbadfacfe85300",
"id": "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854",
"networks": {
    "eth0": {
        "rx_bytes": 84,
        "rx_packets": 2,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 84,
        "tx_packets": 2,
        "tx_errors": 0,
        "tx_dropped": 0
    }
},
"network_rate_stats": {
    "rx_bytes_per_sec": 0,
    "tx_bytes_per_sec": 0
}
},
"5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af": {
    "read": "2020-10-02T00:51:32.512771349Z",
    "preread": "2020-10-02T00:51:31.510597736Z",
    "pids_stats": {
        "current": 3
    },
    "blkio_stats": {
        "io_service_bytes_recursive": [
            ],
        "io_serviced_recursive": [
            ],
        "io_queue_recursive": [
            ],
        "io_service_time_recursive": [
            ],
        "io_wait_time_recursive": [
            ],
        "io_merged_recursive": [
            ],
        "io_time_recursive": [
            ],
        "sectors_recursive": [
            ]
    },
    "num_procs": 0,
    "storage_stats": {
        },
    "cpu_stats": {
        }
}

```

```
"cpu_usage": {
    "total_usage": 379075681,
    "percpu_usage": [
        191355275,
        187720406
    ],
    "usage_in_kernelmode": 60000000,
    "usage_in_usermode": 310000000
},
"system_cpu_usage": 139778000000000,
"online_cpus": 2,
"throttling_data": {
    "periods": 0,
    "throttled_periods": 0,
    "throttled_time": 0
}
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 378825197,
        "percpu_usage": [
            191104791,
            187720406
        ],
        "usage_in_kernelmode": 60000000,
        "usage_in_usermode": 310000000
    },
    "system_cpu_usage": 139758000000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 1814528,
    "max_usage": 6299648,
    "stats": {
        "active_anon": 606208,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 134217728,
        "hierarchical_mems_w_limit": 268435456,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 5377,
        "pgmajfault": 0,
        "ppgpin": 3613,
        "ppgout": 3465,
        "rss": 606208,
        "rss_huge": 0,
        "total_active_anon": 606208,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 5377,
        "total_pgmajfault": 0,
        "total_ppgin": 3613,
        "total_ppgout": 3465,
        "total_rss": 606208,
    }
}
```

```
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writback": 0,
        "unevictable": 0,
        "writback": 0
    },
    "limit": 134217728
},
"name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
"id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
"networks": {
    "eth0": {
        "rx_bytes": 84,
        "rx_packets": 2,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 84,
        "tx_packets": 2,
        "tx_errors": 0,
        "tx_dropped": 0
    }
},
"network_rate_stats": {
    "rx_bytes_per_sec": 0,
    "tx_bytes_per_sec": 0
}
}
```

Task Metadata Endpoint version 3

Important

The task metadata version 3 endpoint is no longer being actively maintained. We recommend that you update the task metadata version 4 endpoint to get the latest metadata endpoint information. For more information, see [the section called “Task metadata endpoint version 4” \(p. 380\)](#).

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Task metadata endpoint version 3 in the Amazon Elastic Container Service User Guide for AWS Fargate](#).

Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called `ECS_CONTAINER_METADATA_URI` into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and [Docker stats](#) are available to tasks. For tasks that use the bridge network mode, network metrics are available when querying the `/stats` endpoints.

Enabling Task Metadata

The task metadata endpoint version 3 feature is enabled by default for tasks that use the Fargate launch type on platform version v1.3.0 or later and tasks that use the EC2 launch type and are launched on Amazon EC2 Linux infrastructure running at least version 1.21.0 of the Amazon ECS container agent or on Amazon EC2 Windows infrastructure running at least version 1.54.0 of the Amazon ECS container agent and use `awsvpc` network mode. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).

Important

For tasks using the Fargate launch type and platform versions prior to v1.3.0, the task metadata version 2 endpoint is supported. For more information, see [Task Metadata Endpoint version 2 \(p. 402\)](#).

Task Metadata Endpoint version 3 Paths

The following task metadata endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI}`

This path returns metadata JSON for the container.

`${ECS_CONTAINER_METADATA_URI}/task`

This path 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 \(p. 398\)](#).

`${ECS_CONTAINER_METADATA_URI}/taskWithTags`

This path returns the metadata for the task included in the /task endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`${ECS_CONTAINER_METADATA_URI}/stats`

This path returns Docker stats JSON for the specific Docker container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

This path 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.

Task Metadata JSON Response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI}/task`) JSON response.

Cluster

The Amazon Resource Name (ARN) or short name of 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.

Limits

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

PullStartedAt

The timestamp for when the first container image pull began.

PullStoppedAt

The timestamp for when the last container image pull finished.

AvailabilityZone

The Availability Zone the task is in.

Note

The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 or later (Windows).

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 (expressed in CPU units) 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.

ClockDrift

The information about the difference between the reference time and the system time. This applies to the Linux operating system.

ReferenceTime

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example 2021-09-07T16:57:44Z.

ClockErrorBound

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

ClockSynchronizationStatus

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are SYNCHRONIZED and NOT_SYNCHRONIZED.

ExecutionStoppedAt

The time stamp for when the tasks DesiredStatus moved to STOPPED. This occurs when an essential container moves to STOPPED.

Examples

The following examples show sample outputs from the task metadata endpoints.

Example Container Metadata Response

When querying the \${ECS_CONTAINER_METADATA_URI} endpoint you are returned only metadata about the container itself. The following is an example output.

```
{  
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",  
    "Name": "nginx-curl",  
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",  
    "Image": "nrdlngr/nginx-curl",  
    "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",  
    "Labels": {  
        "com.amazonaws.ecs.cluster": "default",  
        "com.amazonaws.ecs.container-name": "nginx-curl",  
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-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"
        ]
    }
]
}

```

Example Task Metadata Response

When querying the \${ECS_CONTAINER_METADATA_URI}/task endpoint you are returned metadata about the task the container is part of. The following is an example output.

The following JSON response is for a single-container task.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
    "Family": "nginx",
    "Revision": "5",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Containers": [
        {
            "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbcc4c",
            "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-east-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": "nrqlngr/nginx-curl",
    "ImageID": "sha256:2e00ae64383fc865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-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",
    "AvailabilityZone": "us-east-2b"
}
```

Task Metadata Endpoint version 2

Important

The task metadata version 2 endpoint is no longer being actively maintained. We recommend that you update the task metadata version 4 endpoint to get the latest metadata endpoint information. For more information, see [the section called “Task metadata endpoint version 4” \(p. 380\)](#).

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.

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.

Enabling task metadata

The task metadata version 2 feature is enabled by default for the following:

- Tasks using the Fargate launch type that use platform version v1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 77\)](#).
- Tasks using the EC2 launch type that also use the awsvpc network mode and are launched on Amazon EC2 Linux infrastructure running at least version 1.17.0 of the Amazon ECS container agent or on Amazon EC2 Windows infrastructure running at least version 1.54.0 of the Amazon ECS container agent. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).

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 \(p. 403\)](#).

`169.254.170.2/v2/metadata/<container-id>`

This endpoint returns metadata JSON for the specified Docker container ID.

`169.254.170.2/v2/metadata/taskWithTags`

This path returns the metadata for the task included in the /task endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`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 Resource Name (ARN) or short name of 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.

Limits

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

PullStartedAt

The timestamp for when the first container image pull began.

PullStoppedAt

The timestamp for when the last container image pull finished.

AvailabilityZone

The Availability Zone the task is in.

Note

The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 or later (Windows).

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 (expressed in CPU units) 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.

ClockDrift

The information about the difference between the reference time and the system time. This applies to the Linux operating system.

ReferenceTime

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example 2021-09-07T16:57:44Z.

ClockErrorBound

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

ClockSynchronizationStatus

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are SYNCHRONIZED and NOT_SYNCHRONIZED.

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-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",  
    "Family": "nginx",  
    "Revision": "5",  
    "DesiredStatus": "RUNNING",  
    "KnownStatus": "RUNNING",  
    "Containers": [  
        {
```

```

    "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbcc4c",
    "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-
east-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": "nrqlngr/nginx-curl",
        "ImageID": "sha256:2e00ae64383cf865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",
        "Labels": {
            "com.amazonaws.ecs.cluster": "default",
            "com.amazonaws.ecs.container-name": "nginx-curl",
            "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-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",

```

```
    "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS container agent endpoint

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Container agent endpoint](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The Amazon ECS container agent automatically injects the ECS_AGENT_URI environment variable into the containers of Amazon ECS tasks to provide a method to interact with the container agent API endpoint.

This applies to Amazon ECS tasks launched on Amazon EC2 Linux and Windows instances that are running at least version 1.65.0 of the Amazon ECS container agent.

Note

You can add support for this feature on Amazon EC2 instances using older versions of the Amazon ECS container agent by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).

Task scale-in protection endpoint

Important

If you use Amazon ECS tasks hosted on AWS Fargate, see [Task scale-in protection endpoint](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Use Amazon ECS task scale-in protection to protect your tasks from being terminated by scale-in events from either [Service Auto Scaling](#) or [deployments](#). You can set and get the status of task scale-in protection in the task container application code.

The Amazon ECS container agent uses AWS APIs [GetTaskProtection](#) and [UpdateTaskProtection](#) to offer task scale-in protection. For a list of failure reasons, see [API failure reasons](#).

For more information about task scale-in protection considerations, see [Task scale-in protection considerations \(p. 527\)](#).

Task scale-in protection endpoint path

The following task scale-in protection endpoint path is available to containers: \${ECS_AGENT_URI}/task-protection/v1/state

A PUT request to this URI from within a container will set task scale-in protection. A GET request to this URI will get the current protection status of a task.

Task scale-in protection endpoint request

You can set task scale-in protection using the \${ECS_AGENT_URI}/task-protection/v1/state endpoint with the following request parameters.

Protect

ProtectionEnabled

Specify true to mark a task for protection and false to unset protection, making it eligible for termination.

Type: Boolean

Required: Yes

ExpiresInMinutes

If you set protectionEnabled to true, you can use the expiresInMinutes parameter to specify how long your task must be protected. You can specify a minimum of 1 minute to up to 2,880 minutes (48 hours). During this time period, your task will not be terminated by scale-in events from service Auto Scaling or deployments. After this time period lapses, the protectionEnabled parameter is set to false.

If you don't specify the time, then the task is automatically protected for 120 minutes (2 hours).

Type: Integer

Required: No

The following examples show how to set task protection with different durations.

Example of how to protect a task with the default time period

This example shows how to protect a task with the default time period of 2 hours.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true}'
```

Example of how to protect a task for 60 minutes

This example shows how to protect a task for 60 minutes using the expiresInMinutes parameter.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":60}'
```

Example of how to protect a task for 24 hours

This example shows how to protect a task for 24 hours using the expiresInMinutes parameter.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}'
```

Task scale-in protection endpoint response

The following information is returned from the task scale-in protection endpoint \${ECS_AGENT_URI}/task-protection/v1/state in the JSON response.

Protect

ExpirationDate

The epoch time when protection for the task will expire. If the task is not protected, this value will be null.

ProtectionEnabled

The protection status of the task. If scale-in protection is enabled for a task, the value is true. Otherwise, it is false.

TaskArn

The full Amazon Resource Name (ARN) of the task that the container belongs to.

The following example shows the details returned for a protected task.

```
curl --request GET ${ECS_AGENT_URI}/task-protection/v1/state
```

```
{  
    "protection":{  
        "ExpirationDate":"2022-10-06T02:29:16Z",  
        "ProtectionEnabled":true,  
        "TaskArn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"  
    }  
}
```

Failure

The following information is returned when a failure occurs.

Arn

The full Amazon Resource Name (ARN) of the task.

Detail

The details related to the failure.

Reason

The reason for the failure.

The following example shows the details returned for a task that is not protected.

```
{  
    "failure":{  
        "Arn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0",  
        "Detail":null,  
        "Reason":"TASK_NOT_VALID"  
    }  
}
```

Error

The following information is returned when an exception occurs.

requestID

The AWS request ID for the Amazon ECS API call that results in an exception.

Arn

The full Amazon Resource Name (ARN) of the task or service.

Code

The error code.

Message

The error message.

Note

If a RequestError or RequestTimeout error appears, it is likely that it's a networking issue. Try using VPC endpoints for Amazon ECS.

The following example shows the details returned when an error occurs.

```
{  
    "requestID": "12345-abc-6789-0123-abc",  
    "error": {  
        "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",  
        "Code": "AccessDeniedException",  
        "Message": "User: arn:aws:sts::444455556666:assumed-role/my-ecs-task-  
role/1234567890abcdef0 is not authorized to perform: ecs:GetTaskProtection on resource:  
arn:aws:ecs:us-west-2:555555555555:task/test/1234567890abcdef0 because no identity-based  
policy allows the ecs:GetTaskProtection action"  
    }  
}
```

The following error appears if the Amazon ECS agent is unable to get a response from the Amazon ECS endpoint for reasons such as network issues or the Amazon ECS control plane is down.

```
{  
    "error": {  
        "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",  
        "Code": "RequestCanceled",  
        "Message": "Timed out calling Amazon ECS Task Protection API"  
    }  
}
```

The following error appears when the Amazon ECS agent gets a throttling exception from Amazon ECS.

```
{  
    "requestID": "12345-abc-6789-0123-abc",  
    "error": {  
        "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",  
        "Code": "ThrottlingException",  
        "Message": "Rate exceeded"  
    }  
}
```

Amazon ECS container agent introspection

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

Important

Your container instance must have an IAM role that allows access to Amazon ECS in order to retrieve the metadata. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

To view container instance metadata, log in to your container instance via SSH and run the following command. Metadata includes the container instance ID, the Amazon ECS cluster in which the container instance is registered, and the Amazon ECS container agent version information.

```
curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Output:

```
{
```

```
    "Cluster": "cluster_name",  
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-  
instance/cluster_name/container_instance_id",  
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"  
}
```

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

```
curl http://localhost:51678/v1/tasks
```

Output:

```
{  
  "Tasks": [  
    {  
      "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-  
ae05-543f8example",  
      "DesiredStatus": "RUNNING",  
      "KnownStatus": "RUNNING",  
      "Family": "hello_world",  
      "Version": "8",  
      "Containers": [  
        {  
          "DockerId": "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",  
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",  
          "Name": "mysql"  
        },  
        {  
          "DockerId": "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",  
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",  
          "Name": "wordpress"  
        }  
      ]  
    }  
  ]  
}
```

You can view information for a particular task that is running on a container instance. To specify a specific task or container, append one of the following to the request:

- The task ARN (`?taskarn=task_arn`)
- The Docker ID for a container (`?dockerid=docker_id`)

To get task information with a container's Docker ID, log in to your container instance via SSH and run the following command.

Note

Amazon ECS container agents before version 1.14.2 require full Docker container IDs for the introspection API, not the short version that is shown with `docker ps`. You can get the full Docker ID for a container by running the `docker ps --no-trunc` command on the container instance.

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

Output:

```
{  
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-  
    bc01-22647b9ecfec",  
    "Containers": [  
        {  
            "DockerId": "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",  
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",  
            "Name": "nginx"  
        }  
    ],  
    "DesiredStatus": "RUNNING",  
    "Family": "nginx-efs",  
    "KnownStatus": "RUNNING",  
    "Version": "2"  
}
```

HTTP proxy configuration

You can configure your Amazon ECS container instances to use an HTTP proxy for both the Amazon ECS container agent and the Docker daemon. This is useful if your container instances do not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance. The process differs for Linux and Windows instances, so be sure to read the appropriate section below for your application.

Topics

- [Amazon Linux container instance configuration \(p. 412\)](#)
- [Windows container instance configuration \(p. 415\)](#)

Amazon Linux container instance configuration

To configure your Amazon ECS Linux container instance to use an HTTP proxy, set the following variables in the relevant files at launch time (with Amazon EC2 user data). You could also manually edit the configuration file and restart the agent afterwards.

/etc/ecs/ecs.config (Amazon Linux 2 and Amazon Linux AMI)

HTTP_PROXY=**10.0.0.131:3128**

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Amazon ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

NO_PROXY=169.254.169.254,169.254.170.2.,/var/run/docker.sock

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

/etc/systemd/system/ecs.service.d/http-proxy.conf (Amazon Linux 2 only)

Environment="HTTP_PROXY=**10.0.0.131:3128**"

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for ecs-init to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

Environment="NO_PROXY=169.254.169.254,169.254.170.2.,/var/run/docker.sock"

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

/etc/init/ecs.override (Amazon Linux AMI only)

env HTTP_PROXY=**10.0.0.131:3128**

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for ecs-init to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock

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

/etc/systemd/system/docker.service.d/http-proxy.conf (Amazon Linux 2 only)

Environment="HTTP_PROXY=http://**10.0.0.131:3128**"

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

Environment="NO_PROXY=169.254.169.254"

Set this value to 169.254.169.254 to filter EC2 instance metadata from the proxy.

/etc/sysconfig/docker (Amazon Linux AMI and Amazon Linux 2 only)

export HTTP_PROXY=http://**10.0.0.131:3128**

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

export NO_PROXY=169.254.169.254,169.254.170.2

Set this value to 169.254.169.254 to filter EC2 instance metadata from the proxy.

Setting these environment variables in the above files only affects the Amazon ECS container agent, ecs-init, and the Docker daemon. They do not configure any other services (such as **yum**) to use the proxy.

Example Amazon Linux HTTP proxy user data script

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

To use this script when you launch a container instance, follow the steps in [Launching an Amazon ECS Linux container instance \(p. 272\)](#), and in [Step 6.g \(p. 277\)](#). Then, copy and paste the cloud-boothook script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

Note

The user data script below only supports Amazon Linux 2 and Amazon Linux AMI variants of the Amazon ECS-optimized AMI.

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

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

```

if grep -q 'Amazon Linux release 2' /etc/system-release ; then
    OS=AL2
    echo "Setting OS to Amazon Linux 2"
elif grep -q 'Amazon Linux AMI' /etc/system-release ; then
    OS=ALAMI
    echo "Setting OS to Amazon Linux AMI"
else
    echo "This user data script only supports Amazon Linux 2 and Amazon Linux AMI."
fi

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

# Set Docker HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ]; then
    mkdir /etc/systemd/system/docker.service.d
    cat <<EOF > /etc/systemd/system/docker.service.d/http-proxy.conf
[Service]
Environment="HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/"
Environment="HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/"
Environment="NO_PROXY=169.254.169.254,169.254.170.2"
EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active docker)" == "active" ]
    then
        systemctl restart docker
    fi
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi
# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ]; then
    echo "export HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export NO_PROXY=169.254.169.254,169.254.170.2" >> /etc/sysconfig/docker
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi

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

# Set ecs-init HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ]; then
    mkdir /etc/systemd/system/ecs.service.d
    cat <<EOF > /etc/systemd/system/ecs.service.d/http-proxy.conf
[Service]
Environment="HTTP_PROXY=$PROXY_HOST:$PROXY_PORT/"
Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"
EOF

```

```

systemctl daemon-reload
if [ "$(systemctl is-active ecs)" == "active" ]; then
    systemctl restart ecs
fi
echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-
init_http_proxy
fi
# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ];
then
    cat <<EOF > /etc/init/ecs.override
env HTTP_PROXY=$PROXY_HOST:$PROXY_PORT
env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
EOF
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-
init_http_proxy
fi

```

Windows container instance configuration

To configure your Amazon ECS Windows container instance to use an HTTP proxy, set the following variables at launch time (with Amazon EC2 user data).

```
[Environment]::SetEnvironmentVariable("HTTP_PROXY",
"http://proxy.mydomain:port", "Machine")
```

Set `HTTP_PROXY` to the hostname (or IP address) and port number of an HTTP proxy to use for the Amazon ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
[Environment]::SetEnvironmentVariable("NO_PROXY",
"169.254.169.254,169.254.170.2,\\".\pipe\docker_engine", "Machine")
```

Set `NO_PROXY` to `169.254.169.254,169.254.170.2,\\".\pipe\docker_engine` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

Example Windows HTTP proxy user data script

The example user data PowerShell script below configures the Amazon ECS container agent and the Docker daemon to use an HTTP proxy that you specify. You can also specify a cluster into which the container instance registers itself.

To use this script when you launch a container instance, follow the steps in [the section called “Launching a container instance” \(p. 326\)](#). Just copy and paste the PowerShell script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

Note

The `-EnableTaskIAMRole` option is required to enable IAM roles for tasks. For more information, see [Additional configuration for Windows IAM roles for tasks \(p. 637\)](#).

```

<powershell>
Import-Module ECSTools

$proxy = "http://proxy.mydomain:port"
[Environment]::SetEnvironmentVariable("HTTP_PROXY", $proxy, "Machine")
[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\\".\pipe\docker_engine", "Machine")

Restart-Service Docker
Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole

```

```
</powershell>
```

Using gMSAs for Windows Containers

Amazon ECS supports Active Directory authentication for Windows containers through a special kind of service account called a *group Managed Service Account* (gMSA).

Windows based network applications such as .NET applications often use Active Directory to facilitate authentication and authorization management between users and services. Developers commonly design their applications to integrate with Active Directory and run on domain-joined servers for this purpose. Because Windows containers cannot be domain-joined, you must configure a Windows container to run with gMSA.

A Windows container running with gMSA relies on its host Amazon EC2 instance to retrieve the gMSA credentials from the Active Directory domain controller and provide them to the container instance. For more information, see [Create gMSAs for Windows containers](#).

Note

This feature is not supported on Windows containers on Fargate.

Topics

- [Considerations \(p. 416\)](#)
- [Prerequisites \(p. 417\)](#)
- [Setting up gMSA for Windows Containers on Amazon ECS \(p. 417\)](#)

Considerations

The following should be considered when using gMSAs for Windows containers:

- When using the Amazon ECS-optimized Windows Server 2016 Full AMI for your container instances, the container hostname must be the same as the gMSA account name defined in the credential spec file. To specify a hostname for a container, use the `hostname` container definition parameter. For more information, see [Network settings \(p. 817\)](#).
- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

gMSA data storage options by Operating System

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials

Storage location	Linux	Windows
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

Prerequisites

Before you use the gMSA for Windows containers feature with Amazon ECS, make sure to complete the following:

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
 - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see [Getting Started with AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.
 - An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see [AWS Direct Connect](#).
- You have an existing gMSA account in the Active Directory. For more information, see [Create gMSAs for Windows containers](#).
- **You chose to use *domainless gMSA* or the Amazon ECS Windows container instance hosting the Amazon ECS task must be *domain joined* to the Active Directory and be a member of the Active Directory security group that has access to the gMSA account.**

By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

- You added the required IAM permissions. The permissions that are required depend on the methods that you choose for the initial credentials and for storing the credential specification:
 - If you use *domainless gMSA* for initial credentials, IAM permissions for AWS Secrets Manager are required on the Amazon EC2 instance role.
 - If you store the credential specification in SSM Parameter Store, IAM permissions for Amazon EC2 Systems Manager Parameter Store are required on the task execution role.
 - If you store the credential specification in Amazon S3, IAM permissions for Amazon Simple Storage Service are required on the task execution role.

Setting up gMSA for Windows Containers on Amazon ECS

To set up gMSA for Windows Containers on Amazon ECS, you can follow the complete tutorial that includes configuring the prerequisites [Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI \(p. 743\)](#).

The following sections cover the CredSpec configuration in detail.

Topics

- [Example CredSpec \(p. 418\)](#)
- [Domainless gMSA setup \(p. 418\)](#)
- [Referencing a Credential Spec File in a Task Definition \(p. 419\)](#)

Example CredSpec

Amazon ECS uses a credential spec file that contains the gMSA metadata used to propagate the gMSA account context to the Windows container. You can generate the credential spec file and reference it in the `credentialSpec` field in your task definition. The credential spec file does not contain any secrets.

The following is an example credential spec file:

```
{  
    "CmsPlugins": [  
        "ActiveDirectory"  
    ],  
    "DomainJoinConfig": {  
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",  
        "MachineAccountName": "WebApp01",  
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",  
        "DnsTreeName": "contoso.com",  
        "DnsName": "contoso.com",  
        "NetBiosName": "contoso"  
    },  
    "ActiveDirectoryConfig": {  
        "GroupManagedServiceAccounts": [  
            {  
                "Name": "WebApp01",  
                "Scope": "contoso.com"  
            }  
        ]  
    }  
}
```

Domainless gMSA setup

We recommend domainless gMSA instead of joining the container instances to a single domain. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

1. Before uploading the CredSpec to one of the storage options, add information to the CredSpec with the ARN of the secret in Secrets Manager or SSM Parameter Store. For more information, see [Additional credential spec configuration for non-domain-joined container host use case](#) on the Microsoft Learn website.

Domainless gMSA credential format

The following is the JSON format for the domainless gMSA credentials for your Active Directory. Store the credentials in Secrets Manager or SSM Parameter Store.

```
{  
    "username": "WebApp01",  
    "password": "Test123!",  
    "domainName": "contoso.com"  
}
```

2. Add the following information to the CredSpec file inside the `ActiveDirectoryConfig`. Replace the ARN with the secret in Secrets Manager or SSM Parameter Store.

Note that the `PluginGUID` value must match the GUID in the following example snippet and is required.

```

    "HostAccountConfig": {
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input\"}"
    }
}

```

You can also use a secret in SSM Parameter Store by using the ARN in this format:
`\\"arn:aws:ssm:aws-region:111122223333:parameter/gmsa-plugin-input\".`

3. After you modify the CredSpec file, it should look like the following example:

```

{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-4066351383-705263209-1606769140",
        "MachineAccountName": "WebApp01",
        "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
        "DnsTreeName": "contoso",
        "DnsName": "contoso",
        "NetBiosName": "contoso"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "contoso"
            },
            {
                "Name": "WebApp01",
                "Scope": "contoso"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input\"}"
        }
    }
}

```

Referencing a Credential Spec File in a Task Definition

Amazon ECS supports the following ways to reference the file path in the `credentialSpecs` field of the task definition. For each of these options, you can provide `credentialspec:` or `domainlesscredentialspec:`, depending on whether you are joining the container instances to a single domain, or using domainless gMSA, respectively.

Amazon S3 Bucket

Add the credential spec to an Amazon S3 bucket and then reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {

```

```

        "name": "",
        ...
        "credentialSpecs": [
            "domainlesscredentialspec:arn:aws:s3:::${BucketName}/${ObjectName}"
        ],
        ...
    ],
    ...
}

```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the Amazon S3 bucket.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3>List*"
            ],
            "Resource": [
                "arn:aws:s3:::${bucket_name}",
                "arn:aws:s3:::${bucket_name}/{$object}"
            ]
        }
    ]
}

```

SSM Parameter Store parameter

Add the credential spec to an SSM Parameter Store parameter and then reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `credentialSpecs` field of the task definition.

```

{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "domainlesscredentialspec:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        },
        ...
    ],
    ...
}

```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the SSM Parameter Store parameter.

```

{
    "Version": "2012-10-17",
    "Statement": [

```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "ssm:GetParameters"  
    ],  
    "Resource": [  
        "arn:aws:ssm:region:111122223333:parameter/parameter_name"  
    ]  
}  
}  
]
```

Local File

With the credential spec details in a local file, reference the file path in the `credentialSpecs` field of the task definition. The file path referenced must be relative to the `C:\ProgramData\ Docker \CredentialSpecs` directory and use the backslash ('\`\`') as the file path separator.

```
{  
    "family": "",  
    "executionRoleArn": "",  
    "containerDefinitions": [  
        {  
            "name": "",  
            ...  
            "credentialSpecs": [  
                "credentialspec:file://CredentialSpecDir\CredentialSpecFile.json"  
            ],  
            ...  
        },  
        ...  
    ]  
}
```

Using gMSAs for Linux Containers

Amazon ECS supports Active Directory authentication for Linux containers through a special kind of service account called a *group Managed Service Account* (gMSA).

Linux based network applications, such as .NET Core applications, can use Active Directory to facilitate authentication and authorization management between users and services. You can use this feature by designing applications that integrate with Active Directory and run on domain-joined servers. But, because Linux containers can't be domain-joined, you need to configure a Linux container to run with gMSA.

A Linux container that runs with gMSA relies on the `credentials-fetcher` daemon that runs on the container's host Amazon EC2 instance. That is, the daemon retrieves the gMSA credentials from the Active Directory domain controller and then transfers these credentials to the container instance. For more information about service accounts, see [Create gMSAs for Windows containers](#) on the Microsoft Learn website.

Note

This feature isn't supported on Fargate. For Linux on Fargate, you can follow the example [Using Windows Authentication with Linux Containers on Amazon ECS](#).

Topics

- [Considerations \(p. 422\)](#)

- [Prerequisites \(p. 422\)](#)
- [Setting up gMSA-capable Linux Containers on Amazon ECS \(p. 423\)](#)
- [Credential specification file \(p. 427\)](#)

Considerations

Consider the following before you use gMSA for Linux containers:

- If your containers run on EC2, you can use gMSA for Windows containers and Linux containers. Fargate isn't supported.
- You might need a Windows computer that's joined to the domain to complete the prerequisites. For example, you might need a Windows computer that's joined to the domain to create the gMSA in Active Directory with PowerShell. The RSAT Active Director PowerShell tools are only available for Windows. For more information, see [Installing the Active Directory administration tools](#).
- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

gMSA data storage options by Operating System

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

Prerequisites

Before you use the gMSA for Linux containers feature with Amazon ECS, make sure to complete the following:

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
 - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see [Getting Started with AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.
 - An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see [AWS Direct Connect](#).

- You have an existing gMSA account in the Active Directory. For more information, see [Using gMSAs for Linux Containers \(p. 421\)](#).
- You installed and are running the `credentials-fetcher` daemon on an Amazon ECS Linux container instance. You also added an initial set of credentials to the `credentials-fetcher` daemon to authenticate with the Active Directory.

Note

The `credentials-fetcher` daemon is only available for Amazon Linux 2023 and Fedora 37 and later. The daemon isn't available for Amazon Linux 2. For more information, see [aws/credentials-fetcher](#) on GitHub.

- You set up the credentials for the `credentials-fetcher` daemon to authenticate with the Active Directory. The credentials must be a member of the Active Directory security group that has access to the gMSA account. There are multiple options in [Decide if you want to join the instances to the domain, or use domainless gMSA. \(p. 423\)](#).
- You added the required IAM permissions. The permissions that are required depend on the methods that you choose for the initial credentials and for storing the credential specification:
 - If you use *domainless gMSA* for initial credentials, IAM permissions for AWS Secrets Manager are required on the task execution role.
 - If you store the credential specification in SSM Parameter Store, IAM permissions for Amazon EC2 Systems Manager Parameter Store are required on the task execution role.
 - If you store the credential specification in Amazon S3, IAM permissions for Amazon Simple Storage Service are required on the task execution role.

Setting up gMSA-capable Linux Containers on Amazon ECS

Prepare the infrastructure

The following steps are considerations and setup that are performed once. After you complete these steps, you can automate creating container instances to reuse this configuration.

Decide how the initial credentials are provided and configure the EC2 user data in a reusable EC2 launch template to install the `credentials-fetcher` daemon.

1. **Decide if you want to join the instances to the domain, or use domainless gMSA.**
 - **Join EC2 instances to the Active Directory domain**

- **Join the instances by user data**

Add the steps to join the Active Directory domain to your EC2 user data in an EC2 launch template. Multiple Amazon EC2 Auto Scaling groups can use the same launch template.

You can use these steps [Joining an Active Directory or FreeIPA domain](#) in the Fedora Docs.

- **Make an Active Directory user for domainless gMSA**

The `credentials-fetcher` daemon has a feature that's called *domainless gMSA*. This feature requires a domain, but the EC2 instance doesn't need to be joined to the domain. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance. Instead, you provide the name of a secret in AWS Secrets Manager in the CredSpec file. The secret must contain a username, password, and the domain to log in to.

This feature is supported and can be used with Linux and Windows containers.

This feature is similar to the *gMSA support for non-domain-joined container hosts* feature. For more information about the Windows feature, see [gMSA architecture and improvements](#) on the Microsoft Learn website.

- a. Make a user in your Active Directory domain. The user in Active Directory must have permission to access the gMSA service accounts that you use in the tasks.
- b. Create a secret in AWS Secrets Manager, after you made the user in Active Directory. For more information, see [Create an AWS Secrets Manager secret](#).
- c. Enter the user's username, password, and the domain into JSON key-value pairs called `username`, `password` and `domainName`, respectively.

```
{"username": "username", "password": "passw0rd", "domainName": "example.com"}
```

- d. Add configuration to the CredSpec file for the service account. The additional `HostAccountConfig` contains the Amazon Resource Name (ARN) of the secret in Secrets Manager.

On Windows, the `PluginGUID` must match the GUID in the following example snippet. On Linux, the `PluginGUID` is ignored. Replace `MySecret` with example with the Amazon Resource Name (ARN) of your secret.

```
"ActiveDirectoryConfig": {
    "HostAccountConfig": {
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": {
            "CredentialArn": "arn:aws:secretsmanager:aws-region:111122223333:secret:MySecret"
        }
    }
}
```

- e. The *domainless gMSA* feature needs additional permissions in the task execution role. Follow the step [\(Optional\) domainless gMSA secret \(p. 425\)](#).

2. Configure instances and install credentials-fetcher daemon

You can install the `credentials-fetcher` daemon with a user data script in your EC2 Launch Template. The following examples demonstrate two types of user data, `cloud-config` YAML or `bash` script. These examples are for Amazon Linux 2023 (AL2023). Replace `MyCluster` with the name of the Amazon ECS cluster that you want these instances to join.

- **cloud-config YAML**

```
Content-Type: text/cloud-config
package_reboot_if_required: true
packages:
  # prerequisites
  - dotnet
  - realmd
  - oddjob
  - oddjob-mkhomedir
  - sssd
  - adcli
  - krb5-workstation
  - samba-common-tools
  # https://github.com/aws/credentials-fetcher gMSA credentials management for containers
  - credentials-fetcher
write_files:
  # configure the ECS Agent to join your cluster.
```

```
# replace MyCluster with the name of your cluster.
- path: /etc/ecs/ecs.config
  owner: root:root
  permissions: '0644'
  content: |
    ECS_CLUSTER=MyCluster
    ECS_GMSA_SUPPORTED=true
runcmd:
# start the credentials-fetcher daemon and if it succeeded, make it start after
every reboot
- "systemctl start credentials-fetcher"
- "systemctl is-active credentials-fetch && systemctl enable credentials-fetcher"
```

- **bash script**

If you're more comfortable with bash scripts and have multiple variables to write to `/etc/ecs/ecs.config`, use the following heredoc format. This format writes everything between the lines beginning with `cat` and `EOF` to the configuration file.

```
#!/usr/bin/env bash
set -euxo pipefail

# prerequisites
timeout 30 dnf install -y dotnet realmd oddjob oddjob-mkhomedir sssd adcli krb5-workstation samba-common-tools
# install https://github.com/aws/credentials-fetcher gMSA credentials management
# for containers
timeout 30 dnf install -y credentials-fetcher

# start credentials-fetcher
systemctl start credentials-fetcher
systemctl is-active credentials-fetch && systemctl enable credentials-fetcher

cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_GMSA_SUPPORTED=true
EOF
```

There are optional configuration variables for the `credentials-fetcher` daemon that you can set in `/etc/ecs/ecs.config`. We recommend that you set the variables in the user data in the YAML block or heredoc similar to the previous examples. Doing so prevents issues with partial configuration that can happen with editing a file multiple times. For more information about the ECS agent configuration, see [Amazon ECS Container Agent](#) on GitHub.

- Optionally, you can use the variable `CREDENTIALS_FETCHER_HOST` if you change the `credentials-fetcher` daemon configuration to move the socket to another location.

Setting up permissions and secrets

Do the following steps once for each application and each task definition. We recommend that you use the best practice of granting the least privilege and narrow the permissions used in the policy. This way, each task can only read the secrets that it needs.

1. (Optional) domainless gMSA secret

If you use the domainless method where the instance isn't joined to the domain, follow this step.

You must add the following permissions as an inline policy to the task execution IAM role. Doing so gives the `credentials-fetcher` daemon access to the Secrets Manager secret. Replace the `MySecret` example with the Amazon Resource Name (ARN) of your secret in the Resource list.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:ssm:aws-region:111122223333:secret:MySecret"
            ]
        }
    ]
}
```

Note

If you use your own KMS key to encrypt your secret, you must add the necessary permissions to this role and add this role to the AWS KMS key policy.

2. Decide if you're using SSM Parameter Store or S3 to store the CredSpec

Amazon ECS supports the following ways to reference the file path in the `credentialSpecs` field of the task definition.

If you join the instances to a single domain, use the prefix `credentialspec:` at the start of the ARN in the string. If you use domainless gMSA, then use `credentialspecdomainless:`.

For more information about the CredSpec, see [Credential specification file \(p. 427\)](#).

- **Amazon S3 Bucket**

Add the credential spec to an Amazon S3 bucket. Then, reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        },
        ...
    ],
    ...
}
```

To give your tasks access to the S3 bucket, add the following permissions as an inline policy to the Amazon ECS task execution IAM role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::${BucketName}/*"
            ]
        }
    ]
}
```

```

        "s3>List"
    ],
    "Resource": [
        "arn:aws:s3:::{bucket_name}",
        "arn:aws:s3:::{bucket_name}/{object}"
    ]
}
]
}
}

```

- **SSM Parameter Store parameter**

Add the credential spec to an SSM Parameter Store parameter. Then, reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `credentialSpecs` field of the task definition.

```

{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:ssm:aws-
region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
    ...
}

```

To give your tasks access to the SSM Parameter Store parameter, add the following permissions as an inline policy to the Amazon ECS task execution IAM role.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:aws-region:111122223333:parameter/parameter_name"
            ]
        }
    ]
}

```

Credential specification file

Amazon ECS uses an Active Directory credential specification file (*CredSpec*). This file contains the gMSA metadata that's used to propagate the gMSA account context to the Linux container. You generate the CredSpec and reference it in the `credentialSpecs` field in your task definition. The CredSpec file doesn't contain any secrets.

The following is an example CredSpec file.

```
{  
    "CmsPlugins": [  
        "ActiveDirectory"  
    ],  
    "DomainJoinConfig": {  
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",  
        "MachineAccountName": "WebApp01",  
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",  
        "DnsTreeName": "example.com",  
        "DnsName": "example.com",  
        "NetBiosName": "example"  
},  
    "ActiveDirectoryConfig": {  
        "GroupManagedServiceAccounts": [  
            {  
                "Name": "WebApp01",  

```

Creating a CredSpec

You create a CredSpec by using the CredSpec PowerShell module on a Windows computer that's joined to the domain. Follow the steps in [Create a credential spec](#) on the Microsoft Learn website.

Updating the Amazon ECS container agent with the console

Use the console to update your container agent.

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**, and then choose the cluster.
4. On the **Cluster : cluster_name** page, choose **Infrastructure**.
5. Select the container instances to update.
6. Under **Container Instance**, choose **Actions, Update agent**.

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 use the same cluster state information as the Amazon ECS API to make appropriate placement decisions.

Each task that uses the Fargate launch type has its own isolation boundary and doesn't share underlying resources with any other tasks. These resources include the underlying kernel, CPU resources, memory resources, and elastic network interface.

Amazon ECS provides a service scheduler for long-running tasks and applications. It also provides the ability to run tasks manually for batch jobs or single run tasks. Amazon ECS provides one whenever it places tasks on your cluster. You can specify the task placement strategies and constraints for running tasks that best meet your needs. For example, you can specify whether tasks run across multiple Availability Zones or within a single Availability Zone. And, optionally, you can integrate tasks with your own custom or third-party schedulers.

Service scheduler

The service scheduler is suitable for long running stateless services and applications. The service scheduler ensures that the scheduling strategy that you specify is followed and reschedules tasks when a task fails. For example, if the underlying infrastructure fails, the service scheduler can reschedule tasks.

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 \(p. 455\)](#).
- **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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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 \(p. 453\)](#).

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. This might include deploying a new task definition or changing the number of desired tasks that are running. By default, the service scheduler spreads tasks across multiple Availability Zones. However, you can use task placement strategies and constraints to customize task placement decisions. For more information, see [Amazon ECS services \(p. 453\)](#).

Manually running tasks

The RunTask action is suitable for processes such as batch jobs that perform work and then stop. For example, you can 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. This 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 [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#) 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, you can use EventBridge Scheduler to create a schedule. You can run tasks for a backup operation or a log scan. The EventBridge Scheduler schedule that you create can run one or more tasks in your cluster at specified times. Your scheduled event can be set to a specific interval (run every *N* minutes, hours, or days). Otherwise, for more complicated scheduling, you can use a cron expression. For more information, see [Scheduled tasks \(p. 445\)](#).

Custom schedulers

With Amazon ECS, you can create your own schedulers or use third-party schedulers. [Blox](#) is an open-source project that gives you more control over how your containerized applications run on Amazon ECS. You can use it to build schedulers and integrate third-party schedulers with Amazon ECS. At the same time, you can also use Amazon ECS to manage and scale your clusters. Custom schedulers use the [StartTask](#) API operation to place tasks on specific container instances within your cluster.

Note

Custom schedulers are only compatible with tasks hosted on EC2 instances. If you use Amazon ECS on Fargate, the StartTask API doesn't work.

Task placement

Use RunTask and CreateService actions to specify task placement constraints and task placement strategies. These customize how Amazon ECS places and runs your tasks. For more information, see [Amazon ECS task placement \(p. 434\)](#).

Contents

- [Running a standalone task using the Amazon ECS console \(p. 430\)](#)
- [Stopping tasks using the console \(p. 434\)](#)
- [Amazon ECS task placement \(p. 434\)](#)
- [Scheduled tasks \(p. 445\)](#)
- [Task lifecycle \(p. 449\)](#)
- [AWS Fargate task maintenance \(p. 451\)](#)

Running a standalone task using the Amazon ECS console

We recommend that you deploy your application as a standalone task in some situations. For example, suppose that you're developing an application but you're not ready to deploy it with the service scheduler. If your application is a one-time or periodic batch job, it doesn't make sense to keep running or restart when it finishes.

To deploy your application to run continually or to place it behind a load balancer, create an Amazon ECS service. For more information, see [Amazon ECS services \(p. 453\)](#).

Consider the following when you use the console:

- Task definitions that use the awsvpc network mode or services configured to use a load balancer must have a networking configuration. By default, the console selects the default Amazon VPC along with all subnets and the default security group within the default Amazon VPC.

- For the EC2 launch type, the default task placement strategy is to distribute the tasks across Availability Zones and across container instances in the Availability Zone.
- For the **capacity provider strategy**, the console selects a compute option by default. The following describes the order that the console uses to select a default:
 - If your cluster has a default capacity provider strategy defined, it is selected.
 - If your cluster doesn't have a default capacity provider strategy defined but you do have the Fargate capacity providers added to the cluster, a custom capacity provider strategy that uses the FARGATE capacity provider is selected.
 - If your cluster doesn't have a default capacity provider strategy defined but you do have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected and you need to manually define the strategy.
 - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers added to the cluster, the Fargate launch type is selected.

To run a task from the console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Determine the resource from where you launch the service.

To start a service from	Steps
Clusters	<ol style="list-style-type: none"> a. On the Clusters page, select the cluster to create the service in. b. From the Tasks tab, choose Run new task.
Launch type	<ol style="list-style-type: none"> a. On the Task page, choose the task definition. b. If there is more than one revision, select the revision. c. Choose Deploy, Run task.

3. (Optional) Choose how your scheduled task is distributed across your cluster infrastructure. Expand **Compute configuration**, and then do the following:

Distribution method	Steps
Capacity provider strategy	<ol style="list-style-type: none"> a. In the Compute options section, select Capacity provider strategy. b. Choose a strategy: <ul style="list-style-type: none"> • To use the cluster's default capacity provider strategy, choose Use cluster default. • If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose Use custom, Add capacity provider strategy and define your

Distribution method	Steps
	<p>custom capacity provider strategy by specifying a Base, Capacity provider, and Weight.</p> <p>Note To use a capacity provider in a strategy, the capacity provider must be associated with the cluster. For more information about capacity provider strategies, see Amazon ECS capacity providers (p. 220).</p>
Launch type	<ol style="list-style-type: none"> In the Compute options section, select Launch type. For Launch type, choose a launch type. (Optional) When the Fargate launch type is specified, for Platform version, specify the platform version to use. If a platform version isn't specified, the LATEST platform version is used.

4. For **Application type**, choose **Task**.
5. For **Task definition**, choose the task definition family and revision.

Important

The console validates the selection to ensure that the selected task definition family and revision are compatible with the defined compute configuration.

6. For **Desired tasks**, enter the number of tasks to launch.
7. If your task definition uses the awsvpc network mode, expand **Networking**. Use the following steps to specify a custom configuration.
 - For **VPC**, select the VPC to use.
 - For **Subnets**, select one or more subnets in the VPC that the task scheduler considers when placing your tasks.

Important

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

- For **Security group**, you can either choose an existing security group or create a new one. To use an existing security group, choose the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and then add one or more inbound rules for the security group.
- For **Public IP**, choose whether to auto-assign a public IP address to the elastic network interface (ENI) of the task.

AWS Fargate tasks can be assigned a public IP address when run in a public subnet so they have a route to the internet. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

8. (Optional) To use a task placement strategy other than the default, expand **Task Placement**, and then choose from the following options.

For more information, see [Amazon ECS task placement \(p. 434\)](#).

- **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
- **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
- **BinPack** - Distribute tasks based on the least available amount of CPU or memory.
- **One Task Per Host** - Place, at most, one task from the service on each container instance.
- **Custom** - Define your own task placement strategy.

If you chose **Custom**, define the algorithm for placing tasks and the rules that are considered during task placement.

- Under **Strategy**, for **Type** and **Field**, choose the algorithm and the entity to use for the algorithm.

You can enter a maximum of 5 strategies.

- Under **Constraint**, for **Type** and **Expression**, choose the rule and attribute for the constraint.

When you enter the **Expression**, do not enter the double quotation marks (" "). For example, to set the constraint to place tasks on T2 instances, for the **Expression**, enter **attribute:ecs.instance-type =~ t2.***.

You can enter a maximum of 10 constraints.

9. (Optional) To override the task IAM role, or task execution role that is defined in your task definition, expand **Task overrides**, and then complete the following steps:

- a. For **Task role**, choose an IAM role for this task. For more information, see [Task IAM role \(p. 631\)](#).

Only roles with the `ecs-tasks .amazonaws .com` trust relationship are displayed. For instructions on how to create an IAM role for your tasks, see [Creating an IAM role and policy for your tasks \(p. 633\)](#).

- b. For **Task execution role**, choose a task execution role. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

10. (Optional) To override the container commands and environment variables, expand **Container Overrides**, and then expand the container.

- To send a command to the container other than the task definition command, for **Command override**, enter the Docker command.

For more information about the Docker run command, see [Docker Run reference](#) in the Docker Reference Manual.

- To add an environment variable, choose **Add Environment Variable**. For **Key**, enter the name of your environment variable. For **Value**, enter a string value for your environment value (without the surrounding double quotation marks (" ")).

AWS surrounds the strings with double quotation marks (" ") and passes the string to the container in the following format:

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

11. (Optional) To help identify your task, expand the **Tags** section, and then configure your tags.

To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then select **Task definitions**.

Add or remove a tag.

- [Add a tag] Choose **Add tag**, and then do the following:
 - For **Key**, enter the key name.
 - For **Value**, enter the key value.
- [Remove a tag] Next to the tag, choose **Remove tag**.

12. Choose **Create**.

Stopping tasks using the console

If you decide that you no longer need to keep a task running, you can use the new console to stop one or more tasks.

To stop tasks using the console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : name** page, choose the **Tasks** tab.
5. Perform one of the following operations:
 - To stop one or more tasks, select the tasks, and then choose **Stop, Stop selected**.
 - To stop all tasks, choose **Stop, Stop all**.
6. On the **Stop confirmation page**, enter **Stop**, and then choose **Stop**.

Amazon ECS task placement

When a task that uses the EC2 launch type is launched, Amazon ECS must determine where to place the task based on the requirements specified in the task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate. You can apply task placement strategies and constraints to customize how Amazon ECS places and terminates tasks. Task placement strategies and constraints aren't supported for tasks using the Fargate launch type. Fargate tasks are spread across Availability Zones. If the capacity provider includes both Fargate and Fargate Spot, the spread behavior is independent for each capacity provider. With all other tasks, default task placement strategies depend on whether you're running tasks manually or within a service. For tasks running as part of an Amazon ECS service, the task placement strategy is spread using the attribute:ecs.availability-zone. There isn't a default task placement constraint for tasks in services. For more information, see [Scheduling Amazon ECS tasks \(p. 429\)](#).

A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. For example, Amazon ECS can select instances at random, or it can select instances such that tasks are distributed evenly across a group of instances.

A *task placement constraint* is a rule that's considered during task placement. For example, you can use constraints to place tasks based on Availability Zone or instance type. You can also associate *attributes*,

which are name/value pairs, with your container instances and then use a constraint to place tasks based on attribute.

Note

Task placement strategies are a best effort. Amazon ECS still attempts to place tasks even when the most optimal placement option is unavailable. However, task placement constraints are binding, and they can prevent task placement.

You can use task placement strategies and constraints together. For example, you can use a task placement strategy and a task placement constraint to distribute tasks across Availability Zones and bin pack tasks based on memory within each Availability Zone, but only for G2 instances.

When Amazon ECS places tasks, it uses the following process to select container instances:

1. Identify the instances that satisfy the CPU, GPU, memory, and port requirements in the task definition.
2. Identify the instances that satisfy the task placement constraints.
3. Identify the instances that satisfy the task placement strategies.
4. Select the instances for task placement.

Contents

- [Task groups \(p. 435\)](#)
- [Amazon ECS task placement strategies \(p. 435\)](#)
- [Amazon ECS task placement constraints \(p. 437\)](#)
- [Cluster query language \(p. 442\)](#)

Task groups

You can identify a set of related tasks as a *task group*. All tasks with the same task group name are considered as a set when using the spread task placement strategy. For example, suppose that you're running different applications in one cluster, such as databases and web servers. To ensure that your databases are balanced across Availability Zones, add them to a task group named databases and then use the spread task placement strategy. For more information, see [Amazon ECS task placement strategies \(p. 435\)](#).

Task groups can also be used as a task placement constraint. When you specify a task group in the memberOf constraint, tasks are only sent to container instances that run tasks in the specified task group. For an example, see [Example constraints \(p. 441\)](#).

By default, standalone tasks use the task definition family name (for example, `family:my-task-definition`) as the task group name if a custom task group name isn't specified. Tasks launched as part of a service use the service name as the task group name and can't be changed.

The following requirements for the task group name must be considered.

- A task group name must be 255 or fewer characters.
- Each task can be in exactly one group.
- After launching a task, you can't modify its task group.

Amazon ECS task placement strategies

A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new

service. The task placement strategies can be updated for existing services as well. For more information, see [Amazon ECS task placement \(p. 434\)](#).

For tasks running as part of an Amazon ECS service, the task placement strategy is spread using the `attribute:ecs.availability-zone`. There isn't a default task placement constraint for tasks in services.

Strategy types

Amazon ECS supports the following task placement strategies:

`binpack`

Tasks are placed on container instances so as to leave the least amount of unused CPU or memory. This strategy minimizes the number of container instances in use.

When this strategy is used and a scale-in action is taken, Amazon ECS terminates tasks. It does this based on the amount of resources that are left on the container instance after the task is terminated. The container instance that has the most available resources left after task termination has that task terminated.

`random`

Tasks are placed randomly.

`spread`

Tasks are placed evenly based on the specified value. Accepted values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that's applied to a container instance, such as `attribute:ecs.availability-zone`.

Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group. For more information about task groups, see [Task groups \(p. 435\)](#).

When the `spread` strategy is used and a scale-in action is taken, Amazon ECS selects tasks to terminate that maintain a balance across Availability Zones. Within an Availability Zone, tasks are selected at random.

Example strategies

You can specify task placement strategies with the following actions: [CreateService](#), [UpdateService](#), and [RunTask](#).

The following strategy distributes tasks evenly across Availability Zones.

```
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  }
]
```

The following strategy distributes tasks evenly across all instances.

```
"placementStrategy": [
  {
    "field": "instanceId",
    "type": "spread"
  }
]
```

```
        }  
    ]
```

The following strategy bin packs tasks based on memory.

```
"placementStrategy": [  
    {  
        "field": "memory",  
        "type": "binpack"  
    }  
]
```

The following strategy places tasks randomly.

```
"placementStrategy": [  
    {  
        "type": "random"  
    }  
]
```

The following strategy distributes tasks evenly across Availability Zones and then distributes tasks evenly across the instances within each Availability Zone.

```
"placementStrategy": [  
    {  
        "field": "attribute:ecs.availability-zone",  
        "type": "spread"  
    },  
    {  
        "field": "instanceId",  
        "type": "spread"  
    }  
]
```

The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone.

```
"placementStrategy": [  
    {  
        "field": "attribute:ecs.availability-zone",  
        "type": "spread"  
    },  
    {  
        "field": "memory",  
        "type": "binpack"  
    }  
]
```

Amazon ECS task placement constraints

A *task placement constraint* is a rule that's considered during task placement. At least one container instance must match the constraint. If there are no instances that match the constraint, the task remains in a PENDING state. When you create a new service or update an existing one, you can specify task placement constraints for the service's tasks. You can also specify task placement constraints for standalone tasks. For more information, see [Amazon ECS task placement \(p. 434\)](#).

Constraints consists of a constraint type and a expression in the cluster query language. The constraint type is required, but the expression is optional.

Constraint types

Amazon ECS supports the following types of task placement constraints:

`distinctInstance`

Place each task on a different container instance. This task placement constraint can be specified when either running a task or creating a new service.

`memberOf`

Place tasks on container instances that satisfy an expression. For more information about the expression syntax for constraints, see [Cluster query language \(p. 442\)](#).

The `memberOf` task placement constraint can be specified with the following actions:

- Running a task
- Creating a new service
- Creating a new task definition
- Creating a new revision of an existing task definition

Attributes

You can add custom metadata to your container instances, known as *attributes*. Each attribute has a name and an optional string value. You can use the built-in attributes provided by Amazon ECS or define custom attributes.

The following sections contain sample built-in, optional, and custom attributes.

Built-in attributes

Amazon ECS automatically applies the following attributes to your container instances.

`ecs.ami-id`

The ID of the AMI used to launch the instance. An example value for this attribute is `ami-1234abcd`.

`ecs.availability-zone`

The Availability Zone for the instance. An example value for this attribute is `us-east-1a`.

`ecs.instance-type`

The instance type for the instance. An example value for this attribute is `g2.2xlarge`.

`ecs.os-type`

The operating system for the instance. The possible values for this attribute are `linux` and `windows`.

`ecs.os-family`

The operating system version for the instance.

For Linux instances, the valid value is `LINUX`. For Windows instances, ECS sets the value in the `WINDOWS_SERVER_<OS_Release>_<FULL or CORE>` format. The valid values are `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_20H2_CORE`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, and `WINDOWS_SERVER_2016_FULL`.

This is important for Windows containers and Windows containers on AWS Fargate because the OS version of every Windows container must match that of the host. If the Windows version of the

container image is different than the host, the container doesn't start. For more information, see [Windows container version compatibility](#) on the Microsoft documentation website.

If your cluster runs multiple Windows versions, you can ensure that a task is placed on an EC2 instance running on the same version by using the placement constraint: `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>).` For more information, see [the section called "Retrieving Amazon ECS-Optimized AMI metadata" \(p. 308\)](#).

ecs.cpu-architecture

The CPU architecture for the instance. Example values for this attribute are `x86_64` and `arm64`.

ecs.vpc-id

The VPC the instance was launched into. An example value for this attribute is `vpc-1234abcd`.

ecs.subnet-id

The subnet the instance is using. An example value for this attribute is `subnet-1234abcd`.

Optional attributes

Amazon ECS may add the following attributes to your container instances.

ecs.awsvpc-trunk-id

If this attribute exists, the instance has a trunk network interface. For more information, see [Elastic network interface trunking \(p. 283\)](#).

ecs.outpost-arn

If this attribute exists, it contains the Amazon Resource Name (ARN) of the Outpost. For more information, see [the section called "Amazon Elastic Container Service on AWS Outposts" \(p. 665\)](#).

ecs.capability.external

If this attribute exists, the instance is identified as an external instance. For more information, see [External instances \(Amazon ECS Anywhere\) \(p. 336\)](#).

Custom attributes

You can apply custom attributes to your container instances. For example, you can define an attribute with the name "stack" and a value of "prod".

When specifying custom attributes, you must consider the following.

- The name must contain between 1 and 128 characters and may contain letters (uppercase and lowercase), numbers, hyphens, underscores, forward slashes, back slashes, or periods.
- The value must contain between 1 and 128 characters and may contain letters (uppercase and lowercase), numbers, hyphens, underscores, periods, at signs (@), forward slashes, back slashes, colons, or spaces. The value can't contain any leading or trailing whitespace.

Adding an attribute using the classic console

You can add custom attributes at instance registration time using the container agent or manually, using the AWS Management Console. For more information about container agent attributes, [Amazon ECS container agent configuration \(p. 370\)](#)

To add custom attributes using the classic console

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters** and select a cluster.
3. On the **ECS Instances** tab, select the check box for the container instance.
4. Choose **Actions, View/Edit Attributes**.
5. For each attribute, do the following:
 - a. Choose **Add attribute**.
 - b. Enter a name and a value for the attribute and choose the checkmark icon.
6. When you're finished adding attributes, choose **Close**.

Adding custom attributes using the AWS CLI

The following examples demonstrate how to add custom attributes using the [put-attributes](#) command.

Example: Single Attribute

The following example adds the custom attribute "stack=prod" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn
```

Example: Multiple Attributes

The following example adds the custom attributes "stack=prod" and "project=a" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn  
name=project,value=a,targetId=arn
```

Filtering by attribute using the console

You can apply a filter for your container instances, allowing you to see custom attributes.

Filter container instances by attribute using the classic console

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. Choose a cluster that has container instances.
3. Choose **ECS Instances**.
4. Set column visibility preferences by choosing the gear icon and selecting the attributes to display. This setting persists across all container clusters associated with your account.
5. Using the **Filter by attributes** text field, enter or select the attributes you want to filter by. The format must be *AttributeName:AttributeValue*.

For **Filter by attributes**, enter or select the attributes to filter by. After you select the attribute name, you're prompted for the attribute value.

6. Add additional attributes to the filter as needed. Remove an attribute by choosing the X next to it.

Filter container instances by attribute using the AWS CLI

The following examples demonstrate how to filter container instances by attribute using the [list-container-instances](#) command. For more information about the filter syntax, see [Cluster query language \(p. 442\)](#).

Example: Built-in attribute

The following example uses built-in attributes to list the g2.2xlarge instances.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type == g2.2xlarge"
```

Example: Custom attribute

The following example lists the instances with the custom attribute "stack=prod".

```
aws ecs list-container-instances --filter "attribute:stack == prod"
```

Example: Exclude an attribute value

The following example lists the instances with the custom attribute "stack" unless the attribute value is "prod".

```
aws ecs list-container-instances --filter "attribute:stack != prod"
```

Example: Multiple attribute values

The following example uses built-in attributes to list the instances of type t2.small or t2.medium.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type in [t2.small, t2.medium]"
```

Example: Multiple attributes

The following example uses built-in attributes to list the T2 instances in the us-east-1a Availability Zone.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type =~ t2.* and attribute:ecs.availability-zone == us-east-1a"
```

Example constraints

The following are task placement constraint examples.

This example uses the memberOf constraint to place tasks on T2 instances. It can be specified with the following actions: [CreateService](#), [UpdateService](#), [RegisterTaskDefinition](#), and [RunTask](#).

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

The example uses the memberOf constraint to place tasks on instances with other tasks in the databases task group, respecting any task placement strategies that are also specified. For more information about task groups, see [Task groups \(p. 435\)](#). It can be specified with the following actions: [CreateService](#), [UpdateService](#), [RegisterTaskDefinition](#), and [RunTask](#).

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
```

```
        "type": "memberOf"
    ]
```

The `distinctInstance` constraint places each task in the group on a different instance. It can be specified with the following actions: [CreateService](#), [UpdateService](#), and [RunTask](#)

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

Cluster query language

Cluster queries are expressions that allow you to group objects. For example, you can group container instances by attributes such as Availability Zone, instance type, or custom metadata. For more information, see [Attributes \(p. 438\)](#).

After you have defined a group of container instances, you can customize Amazon ECS to place tasks on container instances based on group. For more information, see [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#), and [Creating an Amazon ECS service in the classic console \(p. 899\)](#). You can also apply a group filter when listing container instances. For more information, see [Filtering by attribute using the console \(p. 440\)](#).

Expression syntax

Expressions have the following syntax:

```
subject operator [argument]
```

Subject

The attribute or field to be evaluated.

`agentConnected`

Select container instances by their Amazon ECS container agent connection status. You can use this filter to search for instances with container agents that are disconnected.

Valid operators: `equals (==)`, `not_equals (!=)`, `in`, `not_in (!in)`, `matches (=~)`, `not_matches (!~)`
`agentVersion`

Select container instances by their Amazon ECS container agent version. You can use this filter to find instances that are running outdated versions of the Amazon ECS container agent.

Valid operators: `equals (==)`, `not_equals (!=)`, `greater_than (>)`, `greater_than_equal (>=)`, `less_than (<)`, `less_than_equal (<=)`

`attribute:attribute-name`

Select container instances by attribute. For more information, see [Attributes \(p. 438\)](#).

`ec2InstanceId`

Select container instances by their Amazon EC2 instance ID.

Valid operators: `equals (==)`, `not_equals (!=)`, `in`, `not_in (!in)`, `matches (=~)`, `not_matches (!~)`

`registeredAt`

Select container instances by their container instance registration date. You can use this filter to find newly registered instances or instances that are very old.

Valid operators: `equals (==)`, `not_equals (!=)`, `greater_than (>)`, `greater_than_equal (>=)`, `less_than (<)`, `less_than_equal (<=)`

Valid date formats: `2018-06-18T22:28:28+00:00`, `2018-06-18T22:28:28Z`, `2018-06-18T22:28:28`, `2018-06-18`

`runningTasksCount`

Select container instances by number of running tasks. You can use this filter to find instances that are empty or near empty (few tasks running on them).

Valid operators: `equals (==)`, `not_equals (!=)`, `greater_than (>)`, `greater_than_equal (>=)`, `less_than (<)`, `less_than_equal (<=)`

`task:group`

Select container instances by task group. For more information, see [Task groups \(p. 435\)](#).

Operator

The comparison operator. The following operators are supported.

Operator	Description
<code>==, equals</code>	String equality
<code>!=, not_equals</code>	String inequality
<code>>, greater_than</code>	Greater than
<code>>=, greater_than_equal</code>	Greater than or equal to
<code><, less_than</code>	Less than
<code><=, less_than_equal</code>	Less than or equal to
<code>exists</code>	Subject exists
<code>!exists, not_exists</code>	Subject doesn't exist
<code>in</code>	Value in argument list
<code>!in, not_in</code>	Value not in argument list
<code>=~, matches</code>	Pattern match
<code>!~, not_matches</code>	Pattern mismatch

Note

A single expression can't contain parentheses. However, parentheses can be used to specify precedence in compound expressions.

Argument

For many operators, the argument is a literal value.

The `in` and `not_in` operators expect an argument list as the argument. You specify an argument list as follows:

```
[argument1, argument2, ..., argumentN]
```

The `matches` and `not_matches` operators expect an argument that conforms to the Java regular expression syntax. For more information, see [java.util.regex.Pattern](#).

Compound expressions

You can combine expressions using the following Boolean operators:

- `&&`, and
- `||`, or
- `!`, not

You can specify precedence using parentheses:

```
(expression1 or expression2) and expression3
```

Example expressions

The following are example expressions.

Example: String Equality

The following expression selects instances with the specified instance type.

```
attribute:ecs.instance-type == t2.small
```

Example: Argument List

The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

Example: Compound Expression

The following expression selects G2 instances that aren't in the us-east-1d Availability Zone.

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

Example: Task Affinity

The following expression selects instances that are hosting tasks in the `service:production` group.

```
task:group == service:production
```

Example: Task Anti-Affinity

The following expression selects instances that aren't hosting tasks in the database group.

```
not(task:group == database)
```

Example: Running task count

The following expression selects instances that are only running one task.

```
runningTasksCount == 1
```

Example: Amazon ECS container agent version

The following expression selects instances that are running a container agent version below 1.14.5.

```
agentVersion < 1.14.5
```

Example: Instance registration time

The following expression selects instances that were registered before February 13, 2018.

```
registeredAt < 2018-02-13
```

Example: Amazon EC2 instance ID

The following expression selects instances with the following Amazon EC2 instance IDs.

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

Scheduled tasks

Amazon ECS supports creating scheduled tasks. Scheduled tasks use Amazon EventBridge Scheduler.

Contents

- [Create a scheduled task in the EventBridge Scheduler console \(p. 445\)](#)
- [View your EventBridge scheduled tasks in the console \(p. 448\)](#)
- [Edit an EventBridge scheduled task \(p. 449\)](#)

Create a scheduled task in the EventBridge Scheduler console

Scheduled tasks are started by Amazon EventBridge Scheduler schedule, which you can create using the EventBridge console. Although you can create a scheduled task in the Amazon ECS console, currently the EventBridge console provides more functionality so the following steps walk you through creating an EventBridge Scheduler schedule that starts a scheduled task.

Complete the following steps before you schedule a task:

1. Use the VPC console to get the subnet IDs where the tasks run and the security group IDs for the subnets. For more information, see [View your subnets](#), and [View your security groups](#) in the *Amazon VPC User Guide*.
2. Configure the EventBridge Scheduler execution role. For more information, see [Set up the execution role](#) in the *Amazon EventBridge Scheduler User Guide*.
3. Configure the EventBridge Scheduler target. For more information, see [Set up a target](#) in the *Amazon EventBridge Scheduler User Guide*.

To create a new schedule using the console

1. Open the Amazon EventBridge Scheduler console at <https://console.aws.amazon.com/scheduler/home>.
2. On the **Schedules** page, choose **Create schedule**.
3. On the **Specify schedule detail** page, in the **Schedule name and description** section, do the following:
 - a. For **Schedule name**, enter a name for your schedule. For example, **MyTestSchedule**.
 - b. (Optional) For **Description**, enter a description for your schedule. For example, **TestSchedule**.
 - c. For **Schedule group**, choose a schedule group from the dropdown list. If you don't have a group, choose **default**. To create a schedule group, choose **create your own schedule**.

You use schedule groups to add tags to groups of schedules.

4. Choose your schedule options.

Occurrence	Do this...
One-time schedule A one-time schedule invokes a target only once at the date and time that you specify.	For Date and time , do the following: <ul style="list-style-type: none">• Enter a valid date in YYYY/MM/DD format.• Enter a timestamp in 24-hour hh:mm format.• For Timezone, choose the timezone.
Recurring schedule A recurring schedule invokes a target at a rate that you specify using a cron expression or rate expression.	<p>a. For Schedule type, do one of the following:</p> <ul style="list-style-type: none">• To use a cron expression to define the schedule, choose Cron-based schedule and enter the cron expression.• To use a rate expression to define the schedule, choose Rate-based schedule and enter the rate expression. <p>For more information about cron and rate expressions, see Schedule types on EventBridge Scheduler in the Amazon EventBridge Scheduler User Guide.</p> <p>b. For Flexible time window, choose Off to turn off the option, or choose one of the pre-defined time windows. For example, if you choose 15 minutes and you set a recurring schedule to invoke</p>

Occurrence	Do this...
	its target once every hour, the schedule runs within 15 minutes after the start of every hour.

5. (Optional) If you chose **Recurring schedule** in the previous step, in the **Timeframe** section, do the following:
 - a. For **Timezone**, choose a timezone.
 - b. For **Start date and time**, enter a valid date in YYYY/MM/DD format, and then specify a timestamp in 24-hour hh:mm format.
 - c. For **End date and time**, enter a valid date in YYYY/MM/DD format, and then specify a timestamp in 24-hour hh:mm format.
6. Choose **Next**.
7. On the **Select target** page, do the following:
 - a. Choose **All APIs**, and then in the search box enter **ECS**.
 - b. Select **Amazon ECS**.
 - c. In the search box, enter **RunTask**, and then choose **RunTask**.
 - d. For **ECS cluster**, choose the cluster.
 - e. For **ECS task**, choose the task definition to use for the task.
 - f. To use a launch type, expand **Compute options**, and then select **Launch type**. Then, choose the launch type.

When the Fargate launch type is specified, for **Platform version**, enter the platform version to use. If there is no platform specified, the LATEST platform version is used.

- g. For **Subnets**, enter the subnet IDs to run the task in.
 - h. For **Security groups**, enter the security group IDs for the subnet.
 - i. (Optional) To use a task placement strategy other than the default, expand **Placement constraint**, and then enter the constraints.
- For more information, see [Amazon ECS task placement \(p. 434\)](#).
- j. (Optional) To help identify your tasks, under **Tags** configure your tags.

To have Amazon ECS automatically tag all newly launched tasks with the task definition tags, select **Enable Amazon ECS managed tags**.

8. Choose **Next**.
9. On the **Settings** page, do the following:
 - a. To turn on the schedule, under **Schedule state**, toggle **Enable schedule**.
 - b. To configure a retry policy for your schedule, under **Retry policy and dead-letter queue (DLQ)**, do the following:
 - Toggle **Retry**.
 - For **Maximum retention time of event**, enter the maximum **hour(s)** and **min(s)** that EventBridge Scheduler must keep an unprocessed event.
 - The maximum time is 24 hours.
 - For **Maximum retries**, enter the maximum number of times EventBridge Scheduler retries the schedule if the target returns an error.

The maximum value is 185 retries.

With retry policies, if a schedule fails to invoke its target, EventBridge Scheduler re-runs the schedule. If configured, you must set the maximum retention time and retries for the schedule.

- c. Choose where EventBridge Scheduler stores undelivered events.

Dead-letter queue (DLQ) option	Do this...	
Don't store	Choose None .	
Store the event in the same AWS account where you're creating the schedule	a. Choose Select an Amazon SQS queue in my AWS account as a DLQ . b. Choose the Amazon Resource Name (ARN) of the Amazon SQS queue.	
Store the event in a different AWS account from where you're creating the schedule	a. Choose Specify an Amazon SQS queue in other AWS accounts as a DLQ . b. Enter the Amazon Resource Name (ARN) of the Amazon SQS queue.	

- d. To use a customer managed key to encrypt your target input, under **Encryption**, choose **Customize encryption settings (advanced)**.

If you choose this option, enter an existing KMS key ARN or choose **Create an AWS KMS key** to navigate to the AWS KMS console. For more information about how EventBridge Scheduler encrypts your data at rest, see [Encryption at rest](#) in the *Amazon EventBridge Scheduler User Guide*.

- e. For **Permissions**, choose **Use existing role**, then select the role.

To have EventBridge Scheduler create a new execution role for you, choose **Create new role for this schedule**. Then, enter a name for **Role name**. If you choose this option, EventBridge Scheduler attaches the required permissions necessary for your templated target to the role.

10. Choose **Next**.
11. In the **Review and create schedule** page, review the details of your schedule. In each section, choose **Edit** to go back to that step and edit its details.
12. Choose **Create schedule**.

You can view a list of your new and existing schedules on the **Schedules** page. Under the **Status** column, verify that your new schedule is **Enabled**.

View your EventBridge scheduled tasks in the console

Your EventBridge scheduled tasks can be viewed in the Amazon ECS console.

To view your scheduled tasks (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Choose **Clusters**, and then choose the cluster your scheduled tasks are run in.
3. On the Cluster: **cluster-name** page, choose the **Scheduled Tasks** tab.

4. All of your scheduled tasks are listed.

Edit an EventBridge scheduled task

You can modify an existing EventBridge schedule using the console.

To edit an EventBridge scheduled task (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Choose the cluster in which to edit your scheduled task.
3. On the **Cluster: *cluster-name*** page, choose the **Scheduled Tasks** tab.
4. Choose the schedule rule to edit, and then choose **Update**.
5. To turn off the schedule, under **Scheduled rule**, toggle **Turned on**.
6. To modify your schedule options, for **Schedule type**, do one of the following.

Occurrence	Do this...
Run at a fixed interval	For Value , enter the number of hours, minutes or days, and then for Unit , choose the interval unit.
Cron	a. For Cron expression , enter the cron expression. For more information about cron and rate expressions, see Schedule types on EventBridge Scheduler in the <i>Amazon EventBridge Scheduler User Guide</i> .

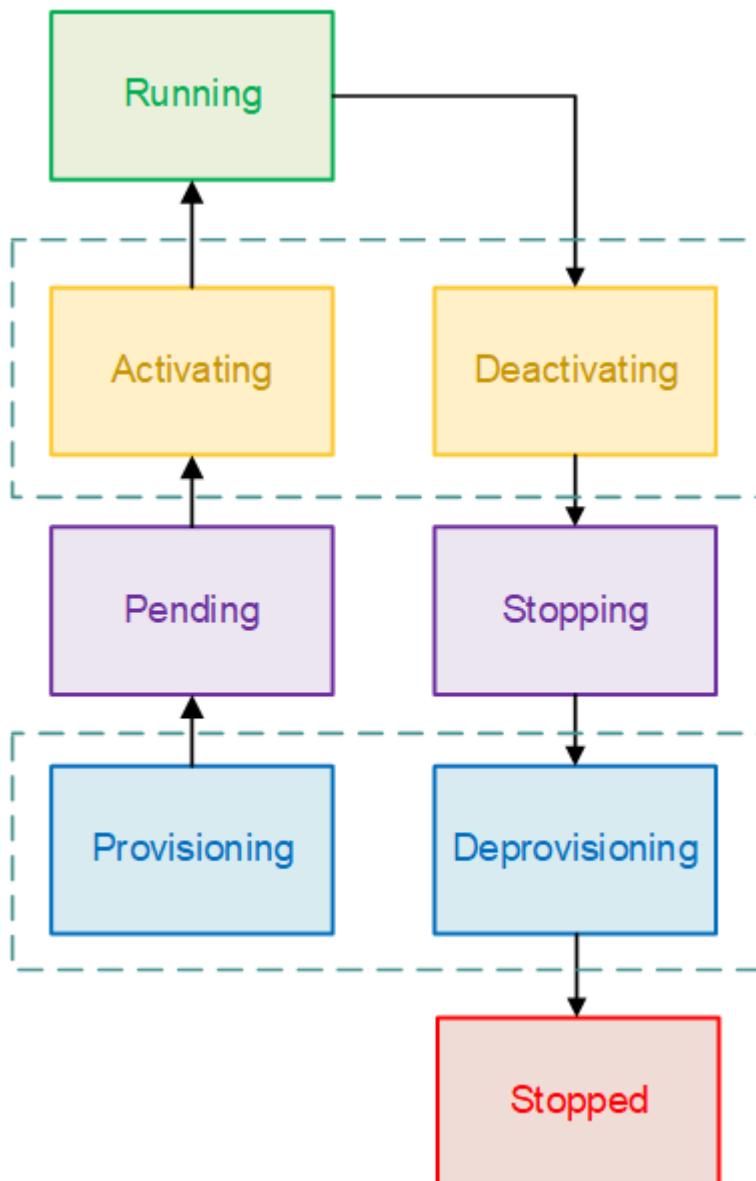
7. Make any additional changes to the targets (clusters and tasks), and then choose **Update**.

Task lifecycle

When a task is started, either manually or as part of a service, it can pass through several states before it finishes on its own or is stopped manually. Some tasks are meant to run as batch jobs that naturally progress through from PENDING to RUNNING to STOPPED. Other tasks, which can be part of a service, are meant to continue running indefinitely, or to be scaled up and down as needed.

When task status changes are requested, such as stopping a task or updating the desired count of a service to scale it up or down, the Amazon ECS container agent tracks these changes as the last known status (`lastStatus`) of the task and the desired status (`desiredStatus`) of the task. Both the last known status and desired status of a task can be seen either in the console or by describing the task with the API or AWS CLI.

The flow chart below shows the task lifecycle flow.



Lifecycle states

The following are descriptions of each of the task lifecycle states.

PROVISIONING

Amazon ECS has to perform additional steps before the task is launched. For example, for tasks that use the `awsvpc` network mode, the elastic network interface needs to be provisioned.

PENDING

This is a transition state where Amazon ECS is waiting on the container agent to take further action. A task stays in the pending state until there are available resources for the task.

ACTIVATING

This is a transition state where Amazon ECS has to perform additional steps after the task is launched but before the task can transition to the `RUNNING` state. For example, for tasks that have

service discovery configured, the service discovery resources must be created. For tasks that are part of a service that's configured to use multiple Elastic Load Balancing target groups, the target group registration occurs during this state.

RUNNING

The task is successfully running.

DEACTIVATING

This is a transition state where Amazon ECS has to perform additional steps before the task is stopped. For example, for tasks that are part of a service that's configured to use multiple Elastic Load Balancing target groups, the target group deregistration occurs during this state.

STOPPING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

For Linux containers, the container agent will send the SIGTERM signal to notify the application needs to finish and shut down, and then sends a SIGKILL after waiting the StopTimeout duration set in the task definition.

DEPROVISIONING

Amazon ECS has to perform additional steps after the task has stopped but before the task transitions to the STOPPED state. For example, for tasks that use the awsvpc network mode, the elastic network interface needs to be detached and deleted.

STOPPED

The task has been successfully stopped.

DELETED

This is a transition state when a task stops. This state is not displayed in the console, but is displayed in `describe-tasks`.

AWS Fargate task maintenance

AWS is responsible for patching and maintaining the underlying infrastructure for AWS Fargate. When AWS determines that a security or infrastructure update is needed for an Amazon ECS task hosted on Fargate, the tasks need to be stopped and new tasks launched to replace them.

Amazon ECS tasks can be categorized as either service tasks or standalone tasks.

service task

Service tasks are tasks deployed as part of an Amazon ECS service and are overseen by the service scheduler.

standalone task

Standalone tasks are tasks started by the RunTask action of the ECS API, either directly or by an external scheduler such as scheduled tasks (which are started by Amazon EventBridge), AWS Batch, or AWS Step Functions.

For service tasks, AWS stops the task when there is an issue with the underlying host or a security issue found with the platform version revision. When AWS stops tasks, AWS uses the [minimum healthy percent](#) launches a new task in an attempt to maintain the desired count for the service. By default, the minimum healthy percent of a service is 100 percent, so a new task is started first before a task is stopped. Service tasks are routinely replaced in the same way when you scale the service or deploy configuration changes or deploy task definition revisions.

For standalone tasks, AWS sends a task retirement notice to your [AWS Health Dashboard](#) when there is an issue with the underlying host or a security issue with the platform version revision that the task is using. The notice also is sent to the email address associated with the account. The task retirement notice provides details about the issue, the task retirement date, and what the next steps are. AWS will stop the task on or after the task retirement date. For standalone tasks, Amazon ECS doesn't launch a replacement task when a task is stopped. Therefore, we recommend that customers monitor the state of standalone tasks and if required, implement logic to replace the stopped tasks. For more information, see [Understanding the task retirement notice \(p. 452\)](#).

When a task is stopped in any of the scenarios mentioned here, you can describe the stopped task to retrieve the stoppedReason value. The stoppedReason containing a ECS is performing maintenance on the underlying infrastructure hosting the task message indicates that the task was stopped due to a task maintenance issue.

To prepare for the task retirement process, we recommend that you test your application behavior by simulating this scenario. You can do this by stopping an individual task in your service to test for resiliency.

Understanding the task retirement notice

When a task retirement notice is sent, you're notified by email of the pending retirement. An email is sent before the event with the task ID and retirement date. This email is sent to the address that's associated with your account. This is the same email address that you use to log in to the AWS Management Console. You can update the contact information for your account on the [Account Settings](#) page.

If you use an email account that you don't check regularly, you can use the [AWS Health Dashboard](#) to determine if any of your tasks are scheduled for retirement. AWS Health notifications can be sent through Amazon EventBridge to archival storage such as Amazon Simple Storage Service, take automated actions such as run an AWS Lambda function, or other notification systems such as Amazon Simple Notification Service. For more information, see [Monitoring AWS Health events with Amazon EventBridge](#). For sample configuration to send notifications to Amazon Chime, Slack, or Microsoft Teams, see the [AWS Health Aware](#) repository on GitHub.

When a task reaches its scheduled retirement date, it's stopped or terminated by AWS. This is if it hasn't already been stopped. For service tasks, the service scheduler launches a new task to replace the retired task, and then stops the task that will be retired. The service scheduler maintains the service's desired count. For standalone tasks, they're stopped and you're responsible for launching a replacement.

Amazon ECS services

You can use an Amazon ECS service to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If one of your tasks fails or stops, the Amazon ECS service scheduler launches another instance of your task definition to replace it. This helps maintain your desired number of tasks in the service.

You can also optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

Service scheduler concepts

We recommend that you use the service scheduler for long running stateless services and applications. The service scheduler ensures that the scheduling strategy that you specify is followed and reschedules tasks when a task fails. For example, if the underlying infrastructure fails, the service scheduler reschedules a task. You can use task placement strategies and constraints to customize how the scheduler places and terminates tasks. If a task in a service stops, the scheduler launches a new task to replace it. This process continues until your service reaches your desired number of tasks based on the scheduling strategy that the service uses. The scheduling strategy of the service is also referred to as the *service type*.

The service scheduler includes logic that throttles how often tasks are restarted if tasks repeatedly fail to start. If a task is stopped without having entered a RUNNING state, the service scheduler starts to slow down the launch attempts and sends out a service event message. This behavior prevents unnecessary resources from being used for failed tasks before you can resolve the issue. After the service is updated, the service scheduler resumes normal scheduling behavior. For more information, see [Service throttle logic \(p. 528\)](#) and [Service event messages \(p. 770\)](#).

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 \(p. 455\)](#).
- **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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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 \(p. 453\)](#).

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. The service scheduler also evaluates the task placement constraints for running tasks, and stops tasks that don't meet the placement constraints. When using this strategy, you don't need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies.

Amazon ECS reserves container instance compute resources including CPU, memory, and network interfaces for the daemon tasks. When you launch a daemon service on a cluster with other replica services, Amazon ECS prioritizes the daemon task. This means that the daemon task is the first task to launch on the instances and the last task to stop. This strategy ensures that resources aren't used by pending replica tasks and are available for the daemon tasks.

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

If a deployment configuration 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%.

You must restart the service when you change the placement constraints for the daemon service. Amazon ECS dynamically updates the resources that are reserved on qualifying instances for the daemon task. For existing instances, the scheduler tries to place the task on the instance.

A new deployment starts when there is a change to the task size or container resource reservation in the task definition.. Amazon ECS picks up the updated CPU and memory reservations for the daemon, and then blocks that capacity for the daemon task.

If there are insufficient resources for either of the above cases, the following happens:

- The task placement fails.
- A CloudWatch event is generated.
- Amazon ECS continues to try and schedule the task on the instance by waiting for resources to become available.
- Amazon ECS frees up any reserved instances that no longer meet the placement constraint criteria and stops the corresponding daemon tasks.

The daemon scheduling strategy can be used in the following cases:

- Running application containers
- Running support containers for logging, monitoring and tracing tasks

Tasks using the Fargate launch type or the CODE_DEPLOY or EXTERNAL deployment controller types don't support the daemon scheduling strategy.

When the service scheduler stops running tasks, it attempts to maintain balance across the Availability Zones in your cluster. The scheduler uses the following logic:

- If a placement strategy is defined, use that strategy to select which tasks to terminate. For example, if a service has an Availability Zone spread strategy defined, a task is selected that leaves the remaining tasks with the best spread.
- If no placement strategy is defined, use the following logic to maintain balance across the Availability Zones in your cluster:
 - Sort the valid container instances. Give priority to instances that have the largest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have two running service task, container instances in either zone B or C are considered optimal for termination.
 - Stop the task on a container instance in an optimal Availability Zone based on the previous steps. Favoring container instances with the largest number of running tasks for this service.

Replica

The *replica* scheduling strategy places and maintains the desired number of tasks in your cluster.

For a service that runs tasks on Fargate, when the service scheduler launches new tasks or stops running tasks, the service scheduler uses a best attempt to maintain a balance across Availability Zones. You don't need to specify task placement strategies or restraints.

When you create a service that runs tasks on EC2 instances, you can optionally specify task placement strategies and constraints to customize task placement decisions. If no task placement strategies or constraints are specified, then by default the service scheduler spreads the tasks across Availability Zones. The service scheduler uses the following logic:

- Determines which of the container instances in your cluster can support your service's task definition (for example, required CPU, memory, ports, and container instance attributes).
- Determines which container instances satisfy any placement constraints that are defined for the service.
- When there's a defined placement strategy, use that strategy to select an instance from the remaining candidates.
- When there's no defined placement strategy, use the following logic to balance tasks across the Availability Zones in your cluster:
 - Sorts the valid container instances. Gives priority to instances that have the fewest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have zero, valid container instances in either zone B or C are considered optimal for placement.
 - Places the new service task on a valid container instance in an optimal Availability Zone based on the previous steps. Favors container instances with the fewest number of running tasks for this service.

Additional service concepts

- You can optionally run your service behind a load balancer. For more information, see [Service load balancing \(p. 486\)](#).
- You can optionally specify a deployment configuration for your service. A deployment is initiated by updating the task definition of a service. During a deployment, the service scheduler uses the *minimum healthy percent* and *maximum percent* parameters to determine the deployment strategy. For more information, see [Service definition parameters \(p. 847\)](#).
- You can optionally configure your service to use Amazon ECS service discovery. Service discovery uses the AWS Cloud Map autonaming APIs to manage DNS entries for your service's tasks. This makes them discoverable from within your VPC. For more information, see [Service discovery \(p. 522\)](#).
- When you delete a service, if there are still running tasks that require cleanup, the service moves from an ACTIVE to a DRAINING status, and the service is no longer visible in the console or in the ListServices API operation. After all tasks transition to either a STOPPING or STOPPED status, the service moves from a DRAINING to INACTIVE status. You can view services in the DRAINING or INACTIVE status by using the DescribeServices API operation. However, in the future, INACTIVE services might be cleaned up and purged from Amazon ECS record keeping, and DescribeServices calls on those services return a ServiceNotFoundException error.
- The bake time is a period of time after a new service version has scaled out and the old service version has scaled in, during which Amazon ECS continues to monitor the alarm associated with the deployment. Amazon ECS computes this time period based on the alarm configuration associated with the deployment.

The bake time applies only when you use CloudWatch alarms to detect deployment failures. For more information, see [the section called "Failure detection methods" \(p. 473\)](#)

Creating a service using the console

You can create a service using the console.

Consider the following when you use the console:

- Currently, the console supports only the **Rolling update** (ECS) deployment type. To use any other deployment type, switch to the classic console.
- Currently, the console supports only the **Replica** service type. To use the Daemon service type, switch to the classic console.
- There are two compute options that distribute your tasks.
 - A **capacity provider strategy** causes Amazon ECS to distribute your tasks in one or across multiple capacity providers.
 - A **launch type** causes Amazon ECS to launch our tasks directly on either Fargate or on the Amazon EC2 instances registered to your clusters.
- Task definitions that use the awsvpc network mode or services configured to use a load balancer must have a networking configuration. By default, the console selects the default Amazon VPC along with all subnets and the default security group within the default Amazon VPC.
- The default task placement strategy distributes tasks evenly across Availability Zones.
- When you use the **Launch Type** for your service deployment, by default the service starts in the subnets in your cluster VPC.
- For the **capacity provider strategy**, the console selects a compute option by default. The following describes the order that the console uses to select a default:
 - If your cluster has a default capacity provider strategy defined, it is selected.
 - If your cluster doesn't have a default capacity provider strategy defined but you do have the Fargate capacity providers added to the cluster, a custom capacity provider strategy that uses the FARGATE capacity provider is selected.
 - If your cluster doesn't have a default capacity provider strategy defined but you do have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected and you need to manually define the strategy.
 - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers added to the cluster, the Fargate launch type is selected.
- The default Deployment failure detection default options are to use the **Amazon ECS deployment circuit breaker** option with the **Rollback on failures** option.

For more information, see [Deployment circuit breaker \(p. 474\)](#).

Quickly create a service

You can use the console to quickly create and deploy a service. The service has the following configuration:

- Deploys in the VPC and subnets associated with your cluster
- Deploys one task
- Uses the rolling deployment
- Uses the capacity provider strategy with your default capacity provider
- Uses the deployment circuit breaker to detect failures and sets the option to automatically roll back the deployment on failure

To deploy a service using the default parameters follow these steps.

To create a service (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation page, choose **Clusters**.
3. On the **Clusters** page, select the cluster to create the service in.
4. From the **Services** tab, choose **Create**.
5. Under **Deployment configuration**, specify how your application is deployed.
 - a. For **Application type**, choose **Service**.
 - b. For **Task definition**, choose the task definition family and revision to use.
 - c. For **Service name**, enter a name for your service.
 - d. For **Desired tasks**, enter the number of tasks to launch and maintain in the service.
6. (Optional) To help identify your service and tasks, expand the **Tags** section, and then configure your tags.

To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then select **Task definitions**.

To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the service tags, select **Turn on Amazon ECS managed tags**, and then select **Service**.

Add or remove a tag.

- [Add a tag] Choose **Add tag**, and then do the following:
 - For **Key**, enter the key name.
 - For **Value**, enter the key value.
- [Remove a tag] Next to the tag, choose **Remove tag**.

Create a service using defined parameters

To create a service using defined parameters, follow these steps.

To create a service (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Determine the resource from where you launch the service.

To start a service from	Steps
Clusters	<ol style="list-style-type: none">a. On the Clusters page, select the cluster to create the service in.b. From the Services tab, choose Create.
Launch type	<ol style="list-style-type: none">a. On the Task page, select the task definition.b. Choose Deploy, Create Service.

3. (Optional) Choose how your tasks are distributed across your cluster infrastructure. Expand **Compute configuration**, and then choose your option.

Distribution method	Steps
Capacity provider strategy	<p>a. Under Compute options, choose Capacity provider strategy.</p> <p>b. Choose a strategy:</p> <ul style="list-style-type: none"> • To use the cluster's default capacity provider strategy, choose Use cluster default. • If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose Use custom, Add capacity provider strategy, and then define your custom capacity provider strategy by specifying a Base, Capacity provider, and Weight. <p>Note To use a capacity provider in a strategy, the capacity provider must be associated with the cluster. For more information about capacity provider strategies, see Amazon ECS capacity providers (p. 220).</p>
Launch type	<p>a. In the Compute options section, select Launch type.</p> <p>b. For Launch type, choose a launch type.</p> <p>c. (Optional) When the Fargate launch type is specified, for Platform version, specify the platform version to use. If a platform version isn't specified, the LATEST platform version is used.</p>

4. To specify how your service is deployed, expand **Deployment configuration**, and then choose your options.
 - a. For **Application type**, leave the choice as **Service**.
 - b. For **Task definition** and **Revision**, choose the task definition family and revision to use.
 - c. For **Service name**, enter a name for your service.

- d. For **Service type**, choose the service scheduling strategy.
 - To have the scheduler place and maintain the desired number of tasks in your cluster, choose **Replica**.

For more information, see [the section called "Service scheduler concepts" \(p. 453\)](#).
 - e. If you chose **Replica**, for **Desired tasks**, enter the number of tasks to launch and maintain in the service.
 - f. To change the minimum healthy percent and maximum percent of running tasks allowed during a service deployment, expand **Deployment options**, and then specify the following parameters.
 - i. For **Min running tasks**, enter the lower limit on the number of tasks in the service 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). For more information, see [Deployment configuration](#).
 - ii. For **Max running tasks**, enter the upper limit on the number of tasks in the service 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).
 - g. To configure how Amazon ECS detects and handles deployment failures, expand **Deployment failure detection**, and then choose your options.
 - i. To stop a deployment when the tasks cannot start, select **Use the Amazon ECS deployment circuit breaker**.

To have the software automatically roll back the deployment to the last completed deployment state when the deployment circuit breaker sets the deployment to a failed state, select **Rollback on failure**.
 - ii. To stop a deployment based on application metrics., select **Use CloudWatch alarms**. Then, from **CloudWatch alarm names**, choose the alarms. To create a new alarm, choose **Create new alarm**.

To have the software automatically roll back the deployment to the last completed deployment state when a CloudWatch alarm sets the deployment to a failed state, select **Rollback on failure**.
5. (Optional) To configure service auto scaling, expand **Service auto scaling**, and then specify the following parameters.
- a. To use service auto scaling, select **Service auto scaling**.
 - b. For **Minimum number of tasks**, enter the lower limit of the number of tasks for Service Auto Scaling to use. The desired count will not go below this count.
 - c. For **Maximum number of tasks**, enter the upper limit of the number of tasks for Service Auto Scaling to use. The desired count will not go above this count.
 - d. Choose the policy type. Under **Service auto scaling** choose one of the following options.

To use this policy type...	Do this...	
Target tracking	<ul style="list-style-type: none"> a. For Scaling policy type, choose Target tracking. b. For Policy name, enter the name of the policy. c. For ECS service metric, select one of the following metrics. 	

To use this policy type...	Do this...
	<ul style="list-style-type: none">• ECSServiceAverageCPUUtilization: Average CPU utilization of the service.• ECSServiceAverageMemoryUtilization: Average memory utilization of the service.• ALBRequestCountPerTarget: Number of requests completed per target in an Application Load Balancer target group.d. For Target value, enter the value the service maintains for the selected metric.e. For Scale-out cooldown period, enter time in seconds after a scale-out activity that no other scale outs take place.f. For Scale-in cooldown period, enter time in seconds after a scale-in activity that no other scale ins take place.g. To prevent the policy from performing a scale-in activity, select Turn off scale-in.

To use this policy type...	Do this...	
Step scaling	<p>a. For Scaling policy type, choose Step scaling.</p> <p>b. For Policy name, enter the policy name.</p> <p>c. For Alarm name, enter a unique name for the alarm.</p> <p>d. For Amazon ECS service metric, choose the metric to use for the alarm.</p> <p>e. For Statistic, choose the alarm statistic.</p> <p>f. For Period, choose the period for the alarm.</p> <p>g. For Alarm condition, choose how to compare the selected metric to the defined threshold.</p> <p>h. For Threshold to compare metrics and Evaluation period to initiate alarm, enter the threshold used for the alarm and how long to evaluate the threshold.</p> <p>i. Under Scaling actions, do the following:</p> <ul style="list-style-type: none"> • For Action, select whether to add to, subtract from, or set a specific desired count for your service. • If you chose to add or subtract tasks, for Value, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is initiated. If you chose to set the desired count, enter the number of tasks. For Type, select whether the Value is an integer or a percent value of the existing desired count. • For Lower bound and Upper bound, enter the lower boundary and upper boundary of your step scaling adjustment. • (Optional) Add additional scaling 	

To use this policy type...	Do this...	
	options. Choose Add new scaling option , and then repeat the Scaling action steps.	

6. (Optional) To use Service Connect, select **Turn on Service Connect**, and then specify the following:
 - a. Under **Service Connect configuration**, specify the client mode.
 - If your service runs a network client application that only needs to connect to other services in the namespace, choose **Client side only**.
 - If your service runs a network or web service application and needs to provide endpoints for this service, and connects to other services in the namespace, choose **Client and server**.
 - b. To use a namespace that is not the default cluster namespace, for **Namespace**, choose the service namespace.
 - c. (Optional) Select the **Use log collection** option to specify a log configuration. For each available log driver, there are log driver options to specify. The default option sends container logs to CloudWatch Logs. The other log driver options are configured using AWS FireLens. For more information, see [Custom log routing \(p. 166\)](#).

The following describes each container log destination in more detail.

- **Amazon CloudWatch** — Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided which creates a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.
- **Amazon Kinesis Data Firehose** — Configure the task to send container logs to Kinesis Data Firehose. The default log driver options are provided which sends logs to an Kinesis Data Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
- **Amazon Kinesis Data Streams** — Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided which sends logs to an Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
- **Amazon OpenSearch Service** — Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided. For more information, see [Forwarding logs to an Amazon OpenSearch Service domain \(p. 193\)](#).
- **Amazon S3** — Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided but you must specify a valid Amazon S3 bucket name.

7. (Optional) To configure a load balancer for your service, expand **Load balancing**.

Choose the load balancer.

To use this load balancer	Do this	
Application Load Balancer	<ol style="list-style-type: none"> a. For Load balancer type, select Application Load Balancer. b. Choose Create a new load balancer to create a new Application Load Balancer or Use an existing load balancer to select an existing Application Load Balancer. 	

To use this load balancer	Do this
	<p>c. For Load balancer name, enter a unique name.</p> <p>d. For Choose container to load balance, choose the container that hosts the service.</p> <p>e. For Listener, enter a port and protocol for the Application Load Balancer to listen for connection requests on. By default, the load balancer will be configured to use port 80 and HTTP.</p> <p>f. For Target group name, enter a name and a protocol for the target group that the Application Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition.</p> <p>g. For Health check path, enter an existing path within your container where the Application Load Balancer periodically sends requests to verify the connection health between the Application Load Balancer and the container. The default is the root directory (/).</p> <p>h. For Health check grace period, enter the amount of time (in seconds) that the service scheduler should ignore unhealthy Elastic Load Balancing target health checks.</p>

To use this load balancer	Do this
Network Load Balancer	<p>a. For Load balancer type, select Network Load Balancer.</p> <p>b. For Load Balancer, choose an existing Network Load Balancer.</p> <p>c. For Choose container to load balance, choose the container that hosts the service.</p> <p>d. For Target group name, enter a name and a protocol for the target group that the Network Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition.</p> <p>e. For Health check path, enter an existing path within your container where the Network Load Balancer periodically sends requests to verify the connection health between the Application Load Balancer and the container. The default is the root directory (/).</p> <p>f. For Health check grace period, enter the amount of time (in seconds) that the service scheduler should ignore unhealthy Elastic Load Balancing target health checks.</p>

8. (Optional) To use a task placement strategy other than the default, expand **Task Placement**, and then choose from the following options.

For more information, see [Amazon ECS task placement \(p. 434\)](#).

- **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
- **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
- **BinPack** - Distribute tasks based on the least available amount of CPU or memory.
- **One Task Per Host** - Place, at most, one task from the service on each container instance.
- **Custom** - Define your own task placement strategy.

If you chose **Custom**, define the algorithm for placing tasks and the rules that are considered during task placement.

- Under **Strategy**, for **Type** and **Field**, choose the algorithm and the entity to use for the algorithm.

You can enter a maximum of 5 strategies.

- Under **Constraint**, for **Type** and **Expression**, choose the rule and attribute for the constraint.

When you enter the **Expression**, do not enter the double quotation marks (" ") . For example, to set the constraint to place tasks on T2 instances, for the **Expression**, enter **attribute:ecs.instance-type =~ t2.***.

You can enter a maximum of 10 constraints.

9. If your task definition uses the awsvpc network mode, expand **Networking**. Use the following steps to specify a custom configuration.

- a. For **VPC**, select the VPC to use.

- b. For **Subnets**, select one or more subnets in the VPC that the task scheduler considers when placing your tasks.

Important

Only private subnets are supported for the awsvpc network mode. Tasks don't receive public IP addresses. Therefore, a NAT gateway is required for outbound internet access, and inbound internet traffic is routed through a load balancer.

- c. For **Security group**, you can either select an existing security group or create a new one. To use an existing security group, select the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and then add one or more inbound rules for the security group.

10. (Optional) To help identify your service and tasks, expand the **Tags** section, and then configure your tags.

To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Task definitions**.

To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the service tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Service**.

Add or remove a tag.

- [Add a tag] Choose **Add tag**, and then do the following:
 - For **Key**, enter the key name.
 - For **Value**, enter the key value.
- [Remove a tag] Next to the tag, choose **Remove tag**.

Updating a service using the console

You can update an Amazon ECS service using the Amazon ECS console. The current service configuration is pre-populated. You can update the task definition, desired task count, capacity provider strategy, platform version, and deployment configuration; or any combination of these.

For information about how to update the blue/green deployment configuration, see [Updating a blue/green deployment configuration using the console \(p. 471\)](#).

Consider the following when you use the new console:

- Only services using the **Rolling update** (ECS) deployment type can be updated using the new Amazon ECS experience.
- Currently, the console supports only the **Replica** service type. To use the Daemon service type, switch to the classic console.
- If you are changing the ports used by containers in a task definition, you might need to update the security groups for the container instances to work with the updated ports.
- Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.
- If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed using the console. You can instead use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*.
- If you update the task definition for the service, the container name and container port that are specified in the load balancer configuration must remain in the task definition.

You can update an existing service to change some of the service configuration parameters, such as the number of tasks that are maintained by a service, which task definition is used by the tasks, or if your tasks are using the Fargate launch type, you can change the platform version your service uses. A service using a Linux platform version cannot be updated to use a Windows platform version and vice versa. If you have an application that needs more capacity, you can scale up your service. If you have unused capacity to scale down, you can reduce the number of desired tasks in your service and free up resources.

If you want to use an updated container image for your tasks, you can create a new task definition revision with that image and deploy it to your service by using the **force new deployment** option in the console.

The service scheduler uses the minimum healthy percent and maximum percent parameters (in the deployment configuration for the service) to determine the deployment strategy.

If a service is using the rolling update (ECS) deployment type, the **minimum healthy percent** represents a lower limit on the number of tasks in a service 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). The parameter also applies while any container instances are in the DRAINING state if the service contains tasks using the EC2 launch type. Use this parameter 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 percent, 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 they are reported as healthy by the load balancer. The default value for minimum healthy percent is 100 percent.

If a service is using the rolling update (ECS) deployment type, the **maximum percent** parameter represents an upper limit on the number of tasks in a service that are allowed in the PENDING, RUNNING, or STOPPING state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer). The parameter also applies while any container instances are in the DRAINING state if the service contains tasks using the EC2 launch type. Use this parameter 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 percent, the scheduler may start four new tasks before stopping the four older tasks. That is provided that the cluster resources required to do this are available. The default value for the maximum percent is 200 percent.

When the service scheduler replaces a task during an update, the service first removes the task from the load balancer (if used) 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 the security groups for the container instances to work with the updated ports.

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 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 service (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster.
3. On the **Cluster overview** page, select the service, and then choose **Update**.
4. To have your service start a new deployment, select **Force new deployment**.
5. For **Task definition**, choose the task definition family and revision.

Important

The console validates that the selected task definition family and revision is compatible with the defined compute configuration. If you receive a warning, verify both your task definition compatibility and the compute configuration selected.

6. For **Desired tasks**, enter the number of tasks you want to run for the service.
7. For **Min running tasks**, enter the lower limit on the number of tasks in the service 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). For more information, see [Deployment configuration](#).
8. For **Max running tasks**, enter the upper limit on the number of tasks in the service 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).
9. To configure how Amazon ECS detects and handles deployment failures, expand **Deployment failure detection**, and then choose your options.
 - a. To stop a deployment when the tasks cannot start, select **Use the Amazon ECS deployment circuit breaker**.

To have the software automatically roll back the deployment to the last completed deployment state when the deployment circuit breaker sets the deployment to a failed state, select **Rollback on failure**.
 - b. To stop a deployment based on application metrics., select **Use CloudWatch alarms**. Then, from **CloudWatch alarm names**, choose the alarms. To create a new alarm, choose **Create new alarm**.

To have the software automatically roll back the deployment to the last completed deployment state when a CloudWatch alarm sets the deployment to a failed state, select **Rollback on failure**.
10. To change the compute options, expand **Deployment options**, **Compute configuration**, and then do the following:
 - a. For services on AWS Fargate, for **Platform version**, choose the new version.
 - b. For services that use a capacity provider strategy, for **capacity provider strategy**, do the following:

- To add an additional capacity provider, choose **Add more**. Then, for **Capacity provider**, choose the capacity provider.
- To remove a capacity provider, to the right of the capacity provider, choose **Remove**.

A service using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider and vice versa.

11. (Optional) To configure service auto scaling, expand **Service auto scaling**, and then specify the following parameters.

- To use service auto scaling, select **Service auto scaling**.
- For **Minimum number of tasks**, enter the lower limit of the number of tasks for Service Auto Scaling to use. The desired count will not go below this count.
- For **Maximum number of tasks**, enter the upper limit of the number of tasks for Service Auto Scaling to use. The desired count will not go above this count.
- Choose the policy type. Under **Service auto scaling** choose one of the following options.

To use this policy type...	Do this...
Target tracking	<p>a. For Scaling policy type, choose Target tracking.</p> <p>b. For Policy name, enter the name of the policy.</p> <p>c. For ECS service metric, select one of the following metrics.</p> <ul style="list-style-type: none"> • ECSServiceAverageCPUUtilization: Average CPU utilization of the service. • ECSServiceAverageMemoryUtilization: Average memory utilization of the service. • ALBRequestCountPerTarget: Number of requests completed per target in an Application Load Balancer target group. <p>d. For Target value, enter the value the service maintains for the selected metric.</p> <p>e. For Scale-out cooldown period, enter time in seconds after a scale-out activity that no other scale outs take place.</p> <p>f. For Scale-in cooldown period, enter time in seconds after a scale-in activity that no other scale ins take place.</p> <p>g. To prevent the policy from performing a scale-</p>

To use this policy type...	Do this...	
	in activity, select Turn off scale-in .	

To use this policy type...	Do this...	
Step scaling	<p>a. For Scaling policy type, choose Step scaling.</p> <p>b. For Policy name, enter the policy name.</p> <p>c. For Alarm name, enter a unique name for the alarm.</p> <p>d. For Amazon ECS service metric, choose the metric to use for the alarm.</p> <p>e. For Statistic, choose the alarm statistic.</p> <p>f. For Period, choose the period for the alarm.</p> <p>g. For Alarm condition, choose how to compare the selected metric to the defined threshold.</p> <p>h. For Threshold to compare metrics and Evaluation period to initiate alarm, enter the threshold used for the alarm and how long to evaluate the threshold.</p> <p>i. Under Scaling actions, do the following:</p> <ul style="list-style-type: none"> • For Action, select whether to add to, subtract from, or set a specific desired count for your service. <ul style="list-style-type: none"> • If you chose to add or subtract tasks, for Value, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is initiated. If you chose to set the desired count, enter the number of tasks. For Type, select whether the Value is an integer or a percent value of the existing desired count. • For Lower bound and Upper bound, enter the lower boundary and upper boundary of your step scaling adjustment. • (Optional) Add additional scaling 	

To use this policy type...	Do this...	
	options. Choose Add new scaling option , and then repeat the Scaling action steps.	

12. (Optional) To use Service Connect, select **Turn on Service Connect**, and then specify the following:
 - a. Under **Service Connect configuration**, specify the client mode.
 - If your service runs a network client application that only needs to connect to other services in the namespace, choose **Client side only**.
 - If your service runs a network or web service application and needs to provide endpoints for this service, and connects to other services in the namespace, choose **Client and server**.
 - b. To use a namespace that is not the default cluster namespace, for **Namespace**, choose the service namespace.
13. (Optional) To help identify your service, expand the **Tags** section, and then configure your tags.
 - [Add a tag] Choose **Add tag** and do the following:
 - For **Key**, enter the key name.
 - For **Value**, enter the key value.
 - [Remove a tag] Next to the tag, choose **Remove tag**.
14. Choose **Update**.

Updating a blue/green deployment configuration using the console

You can update a blue/green deployment configuration using the Amazon ECS console. The current blue/green deployment configuration is pre-populated. You can update the following blue/green deployment options:

- Deployment group name - The CodeDeploy deployment settings
- Application name - The CodeDeploy deployment group
- Deployment configuration - How CodeDeploy routes production traffic to your replacement task set during a deployment
- Test listener on the load balancer - CodeDeploy uses the test listener to route your test traffic to the replacement task set during a deployment

You must configure the new option before you update the configuration.

To update a blue/green deployment configuration (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster.
3. On the **Cluster overview** page, select the service, and then choose **Update**.
4. Expand **Deployment options - Powered by CodeDeploy**, and then choose which options to update:
 - To modify the CodeDeploy deployment group, for **Application name**, choose the deployment group.
 - To modify the CodeDeploy deployment settings, for **Deployment group name**, choose the group.

- To modify how CodeDeploy routes production traffic to your replacement task set during a deployment, for **Deployment configuration**, choose the option.
5. Select the deployment lifecycle event hooks and the associated Lambda functions to run as part of the new revision of the service deployment. The available lifecycle hooks are:
 - **BeforeInstall** – Use this deployment lifecycle event hook to invoke a Lambda function before the replacement task set is created. The result of the Lambda function at this lifecycle event does not initiate a rollback.
 - **AfterInstall** – Use this deployment lifecycle event hook to invoke a Lambda function after the replacement task set is created. The result of the Lambda function at this lifecycle event can initiate a rollback.
 - **BeforeAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function before the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can initiate a rollback.
 - **AfterAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function after the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can initiate a rollback.
 6. To modify the test listener, expand **Load balancing**, and then for **Test listener for CodeDeploy deployment**, choose the test listener.
 7. Choose **Update**.

Deleting a service using the console

You can delete an Amazon ECS service using the console. The service is automatically scaled down to zero before it is deleted. Load balancer resources or service discovery resources associated with the service are not affected by the service deletion. To delete your Elastic Load Balancing resources, see one of the following topics, depending on your load balancer type: [Delete an Application Load Balancer](#) or [Delete a Network Load Balancer](#).

To delete a service (Amazon ECS console)

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster for the service.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : name** page, choose the **Services** tab.
5. Select the services, and then choose **Delete**.
6. To delete a service even if it wasn't scaled down to zero tasks, select **Force delete service**.
7. At the confirmation prompt, enter **delete** and then choose **Delete**.

Amazon ECS Deployment types

An Amazon ECS deployment type determines the deployment strategy that your service uses. There are three deployment types: rolling update, blue/green, and external.

You can view information about the service deployment type on the service details page, or by using the `describe-services` API. For more information, see [DescribeServices](#) in the *Amazon Elastic Container Service API Reference*.

Topics

- [Rolling update \(p. 473\)](#)

- [Blue/Green deployment with CodeDeploy \(p. 477\)](#)
- [External deployment \(p. 481\)](#)

Rolling update

When you start a service which uses the *rolling update* (ECS) deployment type, the Amazon ECS service scheduler replaces the currently running tasks with new tasks. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the deployment configuration. The deployment configuration consists of the `minimumHealthyPercent` and `maximumPercent` values which are defined when the service is created, but can also be updated on an existing service.

The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is 50 and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is 200 and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks. Likewise, if the maximum percent is 125 and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

Important

When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is initiated. If your service has a deployment that is stuck due to an invalid deployment configuration, a service event message will be sent. For more information, see [service \(*service-name*\) was unable to stop or start tasks during a deployment because of the service deployment configuration. Update the `minimumHealthyPercent` or `maximumPercent` value and try again. \(p. 773\)](#).

When a new service deployment is started or when a deployment is completed, Amazon ECS sends a service deployment state change event to EventBridge. This provides a programmatic way to monitor the status of your service deployments. For more information, see [Service deployment state change events \(p. 568\)](#).

In order to take advantage of all the features, use the console or the AWS CLI to deploy your service.

Failure detection methods

The rolling update deployment has two methods which provide a way for you to quickly identify when a deployment has failed, and then to optionally roll back the failure to the last working deployment.

- [the section called “Deployment circuit breaker” \(p. 474\)](#)
- [the section called “CloudWatch alarms” \(p. 475\)](#)

You can use either method, or both methods together. When you use both methods together, the deployment is set to failed as soon as the failure criteria for either failure method is met.

Use the following guidelines to help determine which method to use:

- Circuit breaker - Use this method when you want to stop a deployment when the tasks can't start.
- CloudWatch alarms - Use this method when you want to stop a deployment based on application metrics.

For information about how Amazon ECS deployment process, see [Task deployment](#) in the [Amazon ECS Best Practices Guide](#).

Deployment circuit breaker

A deployment transitions to the failed state when the deployment circuit breaker logic determines that the tasks cannot reach a steady state.

The deployment circuit breaker logic has an option that will automatically roll back a failed deployment to the last working state.

Consider the following when you use the deployment circuit breaker method on a service.

- The `DescribeServices` response provides insight into the state of a deployment, the `rolloutState` and `rolloutStateReason`. When a new deployment is started, the rollout state begins in an `IN_PROGRESS` state. When the service reaches a steady state, the rollout state transitions to `COMPLETED`. If the service fails to reach a steady state and circuit breaker is turned on, the deployment will transition to a `FAILED` state. A deployment in a `FAILED` state doesn't launch any new tasks.
- In addition to the service deployment state change events Amazon ECS sends for deployments that have started and have completed, Amazon ECS also sends an event when a deployment with circuit breaker turned on fails. These events provide details about why a deployment failed or if a deployment was started because of a rollback. For more information, see [Service deployment state change events \(p. 568\)](#).
- If a new deployment is started because a previous deployment failed and a rollback occurred, the `reason` field of the service deployment state change event indicates the deployment was started because of a rollback.
- The deployment circuit breaker is only supported for Amazon ECS services that use the rolling update (ECS) deployment controller.
- You must use the new Amazon ECS console, or the AWS CLI when you use the deployment circuit breaker with the CloudWatch option. For more information, see [the section called "Create a service using defined parameters" \(p. 457\)](#) and [create-service](#) in the [AWS Command Line Interface Reference](#).

The following `create-service` AWS CLI example shows how to create a Linux service when the deployment circuit breaker is used with rollback.

```
aws ecs create-service \
    --service-name MyService \
    --deployment-controller type=ECS \
    --desired-count 2 \
    --deployment-configuration "deploymentCircuitBreaker={enable=true,rollback=true}" \
    --task-definition sample-fargate:1 \
    --launch-type FARGATE \
    --platform-family LINUX \
    --platform-version 1.4.0 \
    --network-configuration
    "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=ENABLED}"
```

Failure threshold

The deployment circuit breaker calculates the threshold value, and then uses the value to determine when to move the deployment to a `FAILED` state.

The deployment circuit breaker has a minimum threshold of 10 and a maximum threshold of 200. and uses the values in the following formula to determine the deployment failure.

```
Minimum threshold <= 0.5 * desired task count => maximum threshold
```

When the result of the calculation is less than the minimum of 10, the failure threshold is set to 10. When the result of the calculation is greater than the maximum of 200, the failure threshold is set to 200.

Note

You cannot change either of the threshold values.

There are two stages for the deployment status check.

1. The deployment circuit breaker monitors tasks that are part of the deployment and checks for tasks that are in the RUNNING state. The scheduler ignores the failure criteria when a task in the current deployment is in the RUNNING state and proceeds to the next stage. When tasks fail to reach in the RUNNING state, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as FAILED.
2. This stage is entered when there are one or more tasks in the RUNNING state. The deployment circuit breaker performs health checks on the following resources for the tasks in the current deployment:
 - Elastic Load Balancing load balancers
 - AWS Cloud Map service
 - Amazon ECS container health checks

When a health check fails for the task, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as FAILED.

The following table provides some examples.

Desired task count	Calculation	Threshold
1	$10 \leq 0.5 * 1 \Rightarrow 200$	10 (the calculated value is less than the minimum)
25	$10 \leq 0.5 * 25 \Rightarrow 200$	13 (the value is rounded up)
400	$10 \leq 0.5 * 400 \Rightarrow 200$	200
800	$10 \leq 0.5 * 800 \Rightarrow 200$	200 (the calculated value is greater than the maximum)

For additional examples about how to use the rollback option, see [Announcing Amazon ECS deployment circuit breaker](#).

CloudWatch alarms

You can configure Amazon ECS to set the deployment to failed when it detects that a specified CloudWatch alarm has gone into the ALARM state.

You can optionally set the configuration to roll back a failed deployment to the last completed deployment.

The following `create-service` AWS CLI example shows how to create a Linux service when the deployment alarms are used with the rollback option.

```
aws ecs create-service \
--service-name MyService \
```

```
--deployment-controller type=ECS \
--desired-count 2 \
--deployment-configuration
"alarms={alarmNames=[alarm1Name,alarm2Name],enable=true,rollback=true}" \
--task-definition sample-fargate:1 \
--launch-type FARGATE \
--platform-family LINUX \
--platform-version 1.4.0 \
--network-configuration
"awsVpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=ENABLED}"
```

Consider the following when you use the Amazon CloudWatch alarms method on a service.

- The deploymentConfiguration request parameter now contains the alarms data type. You can specify the alarm names, whether to use the method, and whether to initiate a rollback when the alarms indicate a deployment failure. For more information, see [CreateService](#) in the *Amazon Elastic Container Service API Reference*.
- The DescribeServices response provides insight into the state of a deployment, the rolloutState and rolloutStateReason. When a new deployment starts, the rollout state begins in an IN_PROGRESS state. When the service reaches a steady state and the bake time is complete, the rollout state transitions to COMPLETED. If the service fails to reach a steady state and the alarm has gone into the ALARM state, the deployment will transition to a FAILED state. A deployment in a FAILED state won't launch any new tasks.
- In addition to the service deployment state change events Amazon ECS sends for deployments that have started and have completed, Amazon ECS also sends an event when a deployment that uses alarms fails. These events provide details about why a deployment failed or if a deployment was started because of a rollback. For more information, see [Service deployment state change events \(p. 568\)](#).
- If a new deployment is started because a previous deployment failed and rollback was turned on, the reason field of the service deployment state change event will indicate the deployment was started because of a rollback.
- If you use the deployment circuit breaker and the Amazon CloudWatch alarms to detect failures, either one can initiate a deployment failure as soon as the criteria for either method is met. A rollback occurs when you use the rollback option for the method that initiated the deployment failure.
- The Amazon CloudWatch alarms is only supported for Amazon ECS services that use the rolling update (ECS) deployment controller.
- You can configure this option by using the new Amazon ECS console, or the AWS CLI. For more information, see [the section called "Create a service using defined parameters" \(p. 457\)](#) and [create-service](#) in the *AWS Command Line Interface Reference*.
- You might notice that the deployment status remains IN_PROGRESS for a prolonged amount of time. The reason for this is that Amazon ECS does not change the status until it has deleted the active deployment, and this does not happen until after the bake time. Depending on your alarm configuration, the deployment might appear to take several minutes longer than it does when you don't use alarms (even though the new primary task set is scaled up and the old deployment is scaled down). If you use CloudFormation timeouts, consider increasing the timeouts. For more information, see [Creating wait conditions in a template](#) in the *AWS CloudFormation User Guide*.
- Amazon ECS calls DescribeAlarms to poll the alarms. The calls to DescribeAlarms count toward the CloudWatch service quotas associated with your account. If you have other AWS services that call DescribeAlarms, there might be an impact on Amazon ECS to poll the alarms. For example, if another service makes enough DescribeAlarms calls to reach the quota, that service is throttled and Amazon ECS' is also throttled and unable to poll alarms. If an alarm is generated during the throttling period, Amazon ECS' might miss the alarm and the roll back might not occur. There is no other impact on the deployment. For more information on CloudWatch service quotas, see [CloudWatch service quotas](#) in the *CloudWatch User Guide*.
- If an alarm is in the ALARM state at the beginning of a deployment, Amazon ECS will not monitor alarms for the duration of that deployment (Amazon ECS ignores the alarm configuration). This

behavior address the case where you want to start a new deployment to fix an initial deployment failure.

Recommended alarms

We recommend that you use the following alarm metrics:

- If you use an Application Load Balancer, use the `HTTPCode_ELB_5XX_Count` and `HTTPCode_ELB_4XX_Count` Application Load Balancer metrics. These metrics check for HTTP spikes. For more information about the Application Load Balancer metrics, see [CloudWatch metrics for your Application Load Balancer](#) in the *User Guide for Application Load Balancers*.
- If you have an existing application, use the `CPUUtilization` and `MemoryUtilization` metrics. These metrics check for the percentage of CPU and memory that the cluster or service uses. For more information, see [the section called “Using CloudWatch metrics” \(p. 547\)](#).
- If you use Amazon Simple Queue Service queues in your tasks, use `ApproximateNumberOfMessagesDelayed` Amazon SQS metric. This metric checks for number of messages in the queue that are delayed and not available for reading immediately. For more information about Amazon SQS metrics, see [Available CloudWatch metrics for Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*.

Blue/Green deployment with CodeDeploy

The *blue/green* deployment type uses the blue/green deployment model controlled by CodeDeploy. Use this deployment type to verify a new deployment of a service before sending production traffic to it. For more information, see [What Is CodeDeploy?](#) in the *AWS CodeDeploy User Guide*.

There are three ways traffic can shift during a blue/green deployment:

- **Canary** — Traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated task set in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.
- **Linear** — Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.
- **All-at-once** — All traffic is shifted from the original task set to the updated task set all at once.

The following are components of CodeDeploy that Amazon ECS uses when a service uses the blue/green deployment type:

CodeDeploy application

A collection of CodeDeploy resources. This consists of one or more deployment groups.

CodeDeploy deployment group

The deployment settings. This consists of the following:

- Amazon ECS cluster and service
- Load balancer target group and listener information
- Deployment roll back strategy
- Traffic rerouting settings
- Original revision termination settings
- Deployment configuration
- CloudWatch alarms configuration that can be set up to stop deployments

- SNS or CloudWatch Events settings for notifications

For more information, see [Working with Deployment Groups](#) in the *AWS CodeDeploy User Guide*.

CodeDeploy deployment configuration

Specifies how CodeDeploy routes production traffic to your replacement task set during a deployment. The following pre-defined linear and canary deployment configuration are available. You can also create custom defined linear and canary deployments as well. For more information, see [Working with Deployment Configurations](#) in the *AWS CodeDeploy User Guide*.

Deployment configuration	Description
CodeDeployDefault.ECSLinear10PercentEvery10Seconds	Shifts 10 percent of traffic every minute until all traffic is shifted.
CodeDeployDefault.ECSLinear10PercentEvery3Minutes	Shifts 10 percent of traffic every three minutes until all traffic is shifted.
CodeDeployDefault.ECSCanary10percent5Minutes	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed five minutes later.
CodeDeployDefault.ECSCanary10percent15Minutes	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed 15 minutes later.
CodeDeployDefault.ECSAllAtOnce	Shifts all traffic to the updated Amazon ECS container at once.

Revision

A revision is the CodeDeploy application specification file (AppSpec file). In the AppSpec file, you specify the full ARN of the task definition and the container and port of your replacement task set where traffic is to be routed when a new deployment is created. The container name must be one of the container names referenced in your task definition. If the network configuration or platform version has been updated in the service definition, you must also specify those details in the AppSpec file. You can also specify the Lambda functions to run during the deployment lifecycle events. The Lambda functions allow you to run tests and return metrics during the deployment. For more information, see [AppSpec File Reference](#) in the *AWS CodeDeploy User Guide*.

Blue/Green Deployment Considerations

Consider the following when using the blue/green deployment type:

- When an Amazon ECS service using the blue/green deployment type is initially created, an Amazon ECS task set is created.
- You must configure the service to use either an Application Load Balancer or Network Load Balancer. The following are the load balancer requirements:
 - You must add a production listener to the load balancer, which is used to route production traffic.
 - An optional test listener can be added to the load balancer, which is used to route test traffic. If you specify a test listener, CodeDeploy routes your test traffic to the replacement task set during a deployment.
 - Both the production and test listeners must belong to the same load balancer.
 - You must define a target group for the load balancer. The target group routes traffic to the original task set in a service through the production listener.

- When a Network Load Balancer is used, only the `CodeDeployDefault.ECSAllAtOnce` deployment configuration is supported.
- For services configured to use service auto scaling and the blue/green deployment type, auto scaling is not blocked during a deployment but the deployment may fail under some circumstances. The following describes this behavior in more detail.
 - If a service is scaling and a deployment starts, the green task set is created and CodeDeploy will wait up to an hour for the green task set to reach steady state and won't shift any traffic until it does.
 - If a service is in the process of a blue/green deployment and a scaling event occurs, traffic will continue to shift for 5 minutes. If the service doesn't reach steady state within 5 minutes, CodeDeploy will stop the deployment and mark it as failed.
 - If a service is in the process of a blue/green deployment and a scaling event occurs, the desired task count might be set to an unexpected value. This is caused by auto scaling considering the running task count as current capacity, which is twice the appropriate number of tasks being used in the desired task count calculation.
- Tasks using the Fargate launch type or the `CODE_DEPLOY` deployment controller types don't support the DAEMON scheduling strategy.
- When you initially create a CodeDeploy application and deployment group, you must specify the following:
 - You must define two target groups for the load balancer. One target group should be the initial target group defined for the load balancer when the Amazon ECS service was created. The second target group's only requirement is that it can't be associated with a different load balancer than the one the service uses.
 - When you create a CodeDeploy deployment for an Amazon ECS service, CodeDeploy creates a *replacement task set* (or *green task set*) in the deployment. If you added a test listener to the load balancer, CodeDeploy routes your test traffic to the replacement task set. This is when you can run any validation tests. Then CodeDeploy reroutes the production traffic from the original task set to the replacement task set according to the traffic rerouting settings for the deployment group.

Amazon ECS console experience

The service create and service update workflows in the Amazon ECS console supports blue/green deployments.

To create an Amazon ECS service that uses the blue/green deployment type, see [Creating an Amazon ECS service in the classic console \(p. 899\)](#).

When you use the classic Amazon ECS console to create an Amazon ECS service using the blue/green deployment type, an Amazon ECS task set and the following CodeDeploy resources are created automatically with the following default settings.

Resource	Default Setting
Application name	<code>AppECS-<cluster[:47]>-<service[:47]></code>
Deployment group name	<code>DgpECS-<cluster[:47]>-<service[:47]></code>
Deployment group load balancer info	The load balancer production listener, optional test listener, and target groups specified are added to the deployment group configuration.
Traffic rerouting settings	Traffic rerouting – The default setting is Reroute traffic immediately . You can change it on the CodeDeploy console or by updating the <code>TrafficRoutingConfig</code> . For more information,

Resource	Default Setting
	see CreateDeploymentConfig in the <i>AWS CodeDeploy API Reference</i> .
Original revision termination settings	The original revision termination settings are configured to wait 1 hour after traffic has been rerouted before terminating the blue task set.
Deployment configuration	The deployment configuration is set to <code>CodeDeployDefault.ECSAllAtOnce</code> by default, which routes all traffic at one time from the blue task set to the green task set. The deployment configuration can be changed using the AWS CodeDeploy console after the service is created.
Automatic rollback configuration	If a deployment fails, the automatic rollback settings are configured to roll it back.

To view details of an Amazon ECS service using the blue/green deployment type, use the **Deployments** tab on the Amazon ECS console.

To view the details of a CodeDeploy deployment group in the CodeDeploy console, see [View Deployment Group Details with CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

To modify the settings for a CodeDeploy deployment group in the CodeDeploy console, see [Change Deployment Group Settings with CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see [Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation](#) in the *AWS CloudFormation User Guide*.

Blue/green deployment required IAM permissions

Amazon ECS blue/green deployments are made possible by a combination of the Amazon ECS and CodeDeploy APIs. Users must have the appropriate permissions for these services before they can use Amazon ECS blue/green deployments in the AWS Management Console or with the AWS CLI or SDKs.

In addition to the standard IAM permissions for creating and updating services, Amazon ECS requires the following permissions. These permissions have been added to the `AmazonECS_FullAccess` IAM policy. For more information, see [AmazonECS_FullAccess \(p. 612\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy>CreateApplication",
        "codedeploy>CreateDeployment",
        "codedeploy>CreateDeploymentGroup",
        "codedeploy>GetApplication",
        "codedeploy>GetDeployment",
        "codedeploy>GetDeploymentGroup",
        "codedeploy>ListApplications",
        "codedeploy>ListDeploymentGroups",
        "codedeploy>ListDeployments",
        "codedeploy>StopDeployment",
        "codedeploy>GetDeploymentTarget",
        "codedeploy>GetDeploymentStatus"
      ]
    }
  ]
}
```

```
"codedeploy>ListDeploymentTargets",
"codedeploy>GetDeploymentConfig",
"codedeploy>GetApplicationRevision",
"codedeploy>RegisterApplicationRevision",
"codedeploy>BatchGetApplicationRevisions",
"codedeploy>BatchGetDeploymentGroups",
"codedeploy>BatchGetDeployments",
"codedeploy>BatchGetApplications",
"codedeploy>ListApplicationRevisions",
"codedeploy>ListDeploymentConfigs",
"codedeploy>ContinueDeployment",
"sns>ListTopics",
"cloudwatch>DescribeAlarms",
"lambda>ListFunctions"
],
"Resource": [
  "*"
]
}
]
```

Note

In addition to the standard Amazon ECS permissions required to run tasks and services, users also require `iam:PassRole` permissions to use IAM roles for tasks.

CodeDeploy needs permissions to call Amazon ECS APIs, modify your Elastic Load Balancing, invoke Lambda functions, and describe CloudWatch alarms, as well as permissions to modify your service's desired count on your behalf. Before creating an Amazon ECS service that uses the blue/green deployment type, you must create an IAM role (`ecsCodeDeployRole`). For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 644\)](#).

The [Create service example \(p. 609\)](#) and [Update service example \(p. 610\)](#) IAM policy examples show the permissions that are required for users to use Amazon ECS blue/green deployments on the AWS Management Console.

External deployment

The *external* deployment type allows you to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service. The details for your service are managed by either the service management API actions (`CreateService`, `UpdateService`, and `DeleteService`) or the task set management API actions (`CreateTaskSet`, `UpdateTaskSet`, `UpdateServicePrimaryTaskSet`, and `DeleteTaskSet`). Each API action manages a subset of the service definition parameters.

The `UpdateService` API action updates the desired count and health check grace period parameters for a service. If the launch type, platform version, load balancer details, network configuration, or task definition need to be updated, you must create a new task set.

The `UpdateTaskSet` API action updates only the scale parameter for a task set.

The `UpdateServicePrimaryTaskSet` API action modifies which task set in a service is the primary task set. When you call the `DescribeServices` API action, it returns all fields specified for a primary task set. If the primary task set for a service is updated, any task set parameter values that exist on the new primary task set that differ from the old primary task set in a service are updated to the new value when a new primary task set is defined. If no primary task set is defined for a service, when describing the service, the task set fields are null.

External deployment considerations

Consider the following when using the external deployment type:

- The supported load balancer types are either an Application Load Balancer or a Network Load Balancer.
- The Fargate launch type or EXTERNAL deployment controller types don't support the DAEMON scheduling strategy.

External deployment workflow

The following is the basic workflow to managing an external deployment on Amazon ECS.

To manage an Amazon ECS service using an external deployment controller

1. Create an Amazon ECS service. The only required parameter is the service name. You can specify the following parameters when creating a service using an external deployment controller. All other service parameters are specified when creating a task set within the service.

`serviceName`

Type: String

Required: Yes

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.

`desiredCount`

The number of instantiations of the specified task set task definition to place and keep running within the service.

`deploymentConfiguration`

Optional deployment parameters that control how many tasks run during a deployment and the ordering of stopping and starting tasks. For more information, see [deploymentConfiguration](#).

`tags`

Type: Array of objects

Required: No

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. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see [Tagging your Amazon ECS resources \(p. 529\)](#).

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

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`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

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

enableECSManagedTags

Specifies whether to use Amazon ECS managed tags for the tasks within the service. For more information, see [Tagging your resources for billing \(p. 532\)](#).

propagateTags

Type: String

Valid values: TASK_DEFINITION | SERVICE

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the TagResource API action.

healthCheckGracePeriodSeconds

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health checks after a task enters a RUNNING state. This is only valid if your service is configured to use a load balancer. If your service has a load balancer defined and you do not specify a health check grace period value, the default value of 0 is used.

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 2,147,483,647 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.

If you do not use an Elastic Load Balancing, we recommend that you use the `startPeriod` in the task definition health check parameters. For more information, see [Health check](#).

schedulingStrategy

The scheduling strategy to use. Services using an external deployment controller support only the REPLICA scheduling strategy. For more information, see [Service scheduler concepts \(p. 453\)](#).

placementConstraints

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints aren't supported.

placementStrategy

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

The following is an example service definition for creating a service using an external deployment controller.

```
{  
    "cluster": "",  
    "serviceName": "",  
    "desiredCount": 0,  
    "role": "",  
    "deploymentConfiguration": {  
        "maximumPercent": 0,  
        "minimumHealthyPercent": 0  
    },  
    "placementConstraints": [  
        {  
            "type": "distinctInstance",  
            "expression": ""  
        }  
    ],  
    "placementStrategy": [  
        {  
            "type": "binpack",  
            "field": ""  
        }  
    ],  
    "healthCheckGracePeriodSeconds": 0,  
    "schedulingStrategy": "REPLICA",  
    "deploymentController": {  
        "type": "EXTERNAL"  
    },  
    "tags": [  
        {  
            "key": "",  
            "value": ""  
        }  
    ],  
    "enableECSManagedTags": true,  
    "propagateTags": "TASK_DEFINITION"  
}
```

2. Create an initial task set. The task set contains the following details about your service:

taskDefinition

The task definition for the tasks in the task set to use.

launchType

Type: String

Valid values: EC2 | FARGATE | EXTERNAL

Required: No

The launch type on which to run your service. If a launch type is not specified, the default capacityProviderStrategy is used by default. For more information, see [Amazon ECS launch types \(p. 85\)](#).

If a launchType is specified, the capacityProviderStrategy parameter must be omitted.

platformVersion

Type: String

Required: No

The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. 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 \(p. 77\)](#).

Note

Platform versions are not specified for tasks using the EC2 launch type.

loadBalancers

A load balancer object representing the load balancer to use with your service. When using an external deployment controller, only Application Load Balancers and Network Load Balancers are supported. If you're using an Application Load Balancer, only one Application Load Balancer target group is allowed per task set.

The following snippet shows an example `loadBalancer` object to use.

```
"loadBalancers": [
    {
        "targetGroupArn": "",
        "containerName": "",
        "containerPort": 0
    }
]
```

Note

When specifying a `loadBalancer` object, you must specify a `targetGroupArn` and omit the `loadBalancerName` parameters.

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's not supported for other network modes. For more information, see [Task networking for tasks that are hosted on Amazon EC2 instances \(p. 90\)](#).

serviceRegistries

The details of the service discovery registries to assign to this service. For more information, see [Service discovery \(p. 522\)](#).

scale

A floating-point percentage of the desired number of tasks to place and keep running in the task set. The value is specified as a percent total of a service's `desiredCount`. Accepted values are numbers between 0 and 100.

The following is a JSON example for creating a task set for an external deployment controller.

```
{
    "service": "",
    "cluster": "",
    "externalId": "",
```

```

    "taskDefinition": "",
    "networkConfiguration": [
        "awsVpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    ],
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}

```

3. When service changes are needed, use the `UpdateService`, `UpdateTaskSet`, or `CreateTaskSet` API action depending on which parameters you're updating. If you created a task set, use the `scale` parameter for each task set in a service to determine how many tasks to keep running in the service. For example, if you have a service that contains `tasksetA` and you create a `tasksetB`, you might test the validity of `tasksetB` before wanting to transition production traffic to it. You could set the `scale` for both task sets to `100`, and when you were ready to transition all production traffic to `tasksetB`, you could update the `scale` for `tasksetA` to `0` to scale it down.

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.

Note

When you use tasks sets, all the tasks in the set must all be configured to use Elastic Load Balancing or to not use Elastic Load Balancing.

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

- Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups.
- They are supported by tasks hosted on both Fargate and EC2 instances.
- 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](#).

With your load balancer, you pay only for what you use. For more information, see [Elastic Load Balancing pricing](#).

Topics

- [Load balancer types \(p. 487\)](#)
- [Creating a load balancer \(p. 490\)](#)
- [Registering multiple target groups with a service \(p. 496\)](#)

Load balancer types

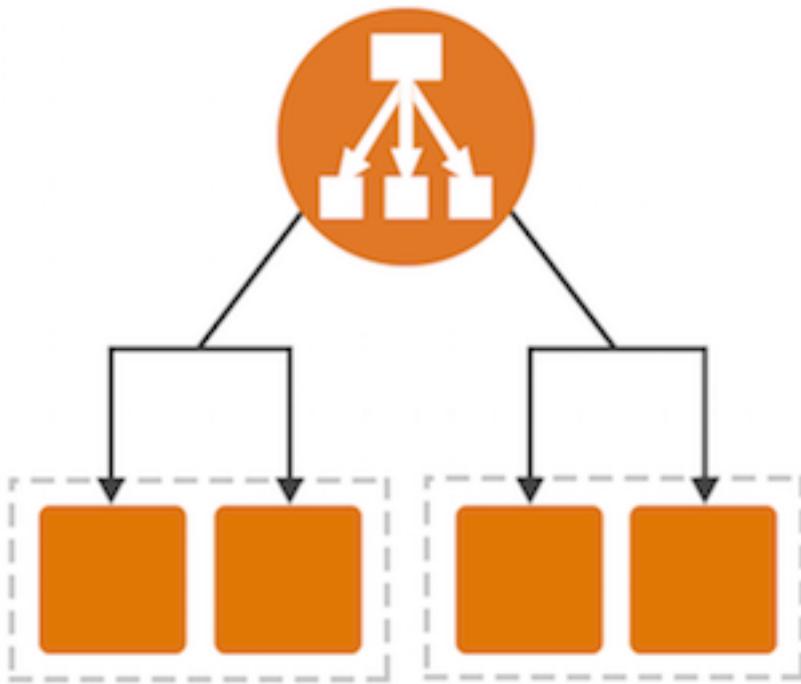
Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, and Network Load Balancers. Amazon ECS services can use these types of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load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

Topics

- [Application Load Balancer \(p. 487\)](#)
- [Network Load Balancer \(p. 488\)](#)
- [Application Load Balancer and Network Load Balancer considerations \(p. 489\)](#)

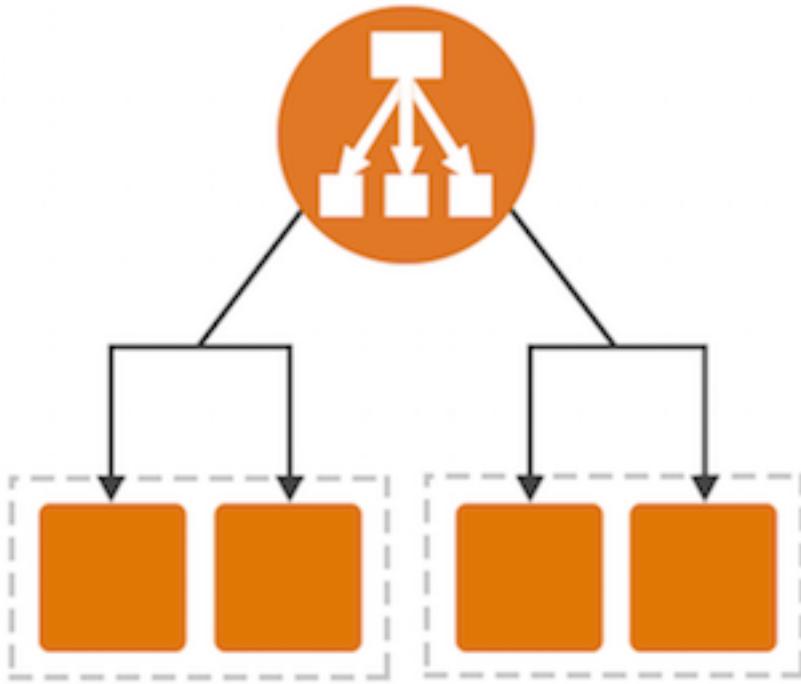
Application Load Balancer

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](#).



Application Load Balancer and Network Load Balancer considerations

The following considerations are specific to Amazon ECS services using Application Load Balancers or Network Load Balancers:

- Amazon ECS requires the service-linked IAM role which provides the permissions needed to register and deregister targets with your load balancer when tasks are created and stopped. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).
- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- For services with tasks using the awsvpc network mode, when you create a target group for your service, 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.
- If your service uses an Application Load Balancer and requires access to multiple load balanced ports, such as port 80 and port 443 for an HTTP/HTTPS service, you can configure two listeners. One listener is responsible for HTTPS that forwards the request to the service, and another listener that is responsible for redirecting HTTP requests to the appropriate HTTPS port. For more information, see [Create a listener to your Application Load Balancer](#) in the *User Guide for Application Load Balancers*.
- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the ECS rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes

tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*.

- If a service's task fails the load balancer health check criteria, the task is stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- When you use a Network Load Balancer configured with IP addresses as targets and Client IP Preservation disabled, requests are seen as coming from the Network Load Balancers private IP address. This means that services behind an Network Load Balancer are effectively open to the world as soon as you allow incoming requests and health checks in the target's security group.
- Using a Network Load Balancer to route UDP traffic to your Amazon ECS tasks on Fargate require the task to use platform version 1.4.0 (Linux) or 1.0.0 (Windows).
- Minimize errors in your client applications by setting the StopTimeout in the task definition longer than the target group deregistration delay, which should be longer than your client connection timeout. See the Builders Library for more information on recommended client configuration [here](#).

Also, the Network Load Balancer target group attribute for connection termination closes all remaining connections after the deregistration time. This can cause clients to display undesired error messages, if the client does not handle them.

- If you are experiencing problems with your load balancer-enabled services, see [Troubleshooting service load balancers \(p. 778\)](#).
- Your tasks and load balancer (Application Load Balancer or Network Load Balancer) must be in the same VPC.
- The Network Load Balancer client IP address preservation is also compatible with Fargate targets.

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, and Network 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 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](#).

Prior to using a load balancer with your Amazon ECS service, your account must already have the Amazon ECS service-linked role created. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

Topics

- [Creating an Application Load Balancer \(p. 491\)](#)

- [Creating a Network Load Balancer \(p. 493\)](#)

Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console. For information about how to create an Application Load Balancer using the AWS CLI, see [Tutorial: Create an Application Load Balancer using the AWS CLI](#) in the *User Guide for Application Load Balancers*.

Configure a target group for 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.

Each target group is used to route requests to one or more registered targets. When a rule condition is met, traffic is forwarded to the corresponding 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 create a target group using the console

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **LOAD BALANCING**, choose **Target Groups**.
3. Choose **Create target group**.
4. For **Choose a target type**, **Instances** to register targets by instance ID, **IP addresses** to register targets by IP address, or **Lambda function** to register a Lambda function as a target.

Important

If your service's task definition uses the awsvpc network mode (which is required for the Fargate launch type), you must choose **IP addresses** as the target type. This is because tasks that use the awsvpc network mode are associated with an elastic network interface, not an Amazon EC2 instance.

5. For **Target group name**, enter a name for the target group. This name must be unique per region per account, can have a maximum of 32 characters, must contain only alphanumeric characters or hyphens, and must not begin or end with a hyphen.
6. (Optional) For **Protocol** and **Port**, modify the default values as needed.
7. If the target type is **IP addresses**, choose **IPv4** as the **IP address type**, otherwise skip to the next step.

Note that only targets that have the selected IP address type can be included in this target group. The IP address type cannot be changed after the target group is created.

8. For **VPC**, select a virtual private cloud (VPC). Note that for **IP addresses** target types, the VPCs available for selection are those that support the **IP address type** that you chose in the previous step.
9. (Optional) For **Protocol version**, modify the default value as needed.
10. (Optional) In the **Health checks** section, modify the default settings as needed.
11. If the target type is **Lambda function**, you can enable health checks by selecting **Enable** in the **Health checks** section.
12. (Optional) Add one or more tags as follows:
 - a. Expand the **Tags** section.

- b. Choose **Add tag**.
 - c. Enter the tag key and the tag value.
13. Choose **Next**.
14. Choose **Create target group**.

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 configure your load balancer and listener

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Load Balancing**, choose **Load Balancers**.
3. Choose **Create Load Balancer**.
4. Under **Application Load Balancer**, choose **Create**.
5. Under **Basic configuration**, do the following:
 - a. For **Load balancer name**, enter a name for your load balancer. For example, **my-nlb**.

The name of your Application Load Balancer must be unique within your set of Application Load Balancers and Network Load Balancers for the Region. Names can have a maximum of 32 characters, and can contain only alphanumeric characters and hyphens. They cannot begin or end with a hyphen, or with `internal-`.
 - b. For **Scheme**, choose **Internet-facing or Internal**.

An internet-facing load balancer routes requests from clients to targets over the internet. An internal load balancer routes requests to targets using private IP addresses.
6. Under **Network mapping**, do the following:
 - a. For **VPC**, select the same VPC that you used for the container instances on which you intend to run your service.
 - b. For **Mappings**, select the Availability Zones to use 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.
7. Under **Security groups**, do the following:

For **Security groups**, select an existing security group, or create a new one.

The security group for your load balancer must allow it to communicate with registered targets on both the listener port and the health check port. The console can create a security group for your load balancer on your behalf with rules that allow this communication. You can also create a security group and select it instead. For information about how to create a security group, see [Security groups for your Application Load Balancer](#) in *Elastic Load Balancing Application Load Balancers*.

(Optional) To create a new security group for your load balancer, choose **Create a new security group**.

8. Under **Listeners and routing**, do the following:

The default listener accepts HTTP traffic on port 80. You can keep the default protocol and port. For **Default action**, choose the target group that you created.

You can optionally add an HTTPS listener after you create the load balancer. For information about how to add the listener, see [Add an HTTPS listener](#) in *Elastic Load Balancing Application Load Balancers*.

9. (Optional) You can use **Add-on services**, such as the **AWS Global Accelerator** to create an accelerator and associate the load balancer with the accelerator.

The accelerator name can have up to 64 characters. Allowed characters are a-z, A-Z, 0-9, . and - (hyphen). After the accelerator is created, you can use the AWS Global Accelerator console to manage it.

10. (Optional) Tag your Application Load Balancer. Under **Tag and create**, do the following

- a. Expand the **Tags** section.
- b. Choose **Add tag**.
- c. Enter the tag key and the tag value.

11. Review your configuration, and choose **Create 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 inbound rules**.
5. For **Type**, choose **All traffic**.
6. For **Source**, choose **Custom**, and then select the Application Load Balancer security group.

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 using the console \(p. 456\)](#).

Creating a Network Load Balancer

Learn how to create an Network Load Balancer in the AWS Management Console.

Configure a target group for 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.

Each target group is used to route requests to one or more registered targets. When a rule condition is met, traffic is forwarded to the corresponding 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 create a target group using the console

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **LOAD BALANCING**, choose **Target Groups**.
3. Choose **Create target group**.
4. For **Choose a target type**, **Instances** to register targets by instance ID, **IP addresses** to register targets by IP address, or **Lambda function** to register a Lambda function as a target.

Important

If your service's task definition uses the awsvpc network mode (which is required for the Fargate launch type), you must choose **IP addresses** as the target type. This is because tasks that use the awsvpc network mode are associated with an elastic network interface, not an Amazon EC2 instance.

5. For **Target group name**, enter a name for the target group. This name must be unique per region per account, can have a maximum of 32 characters, must contain only alphanumeric characters or hyphens, and must not begin or end with a hyphen.
6. (Optional) For **Protocol** and **Port**, modify the default values as needed.
7. If the target type is **IP addresses**, choose **IPv4** as the **IP address type**, otherwise skip to the next step.

Note that only targets that have the selected IP address type can be included in this target group. The IP address type cannot be changed after the target group is created.

8. For **VPC**, select a virtual private cloud (VPC). Note that for **IP addresses** target types, the VPCs available for selection are those that support the **IP address type** that you chose in the previous step.
9. (Optional) For **Protocol version**, modify the default value as needed.
10. (Optional) In the **Health checks** section, modify the default settings as needed.
11. (Optional) Add one or more tags as follows:
 - a. Expand the **Tags** section.
 - b. Choose **Add tag**.
 - c. Enter the tag key and the tag value.
12. Choose **Next**.
13. 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.

14. Choose **Create target group**.

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.

To create a Network Load Balancer

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation bar, choose a Region for your load balancer. Be sure to choose the same Region that you used for your EC2 instances.
3. In the navigation pane, under **Load Balancing**, choose **Load Balancers**.
4. Choose **Create load balancer**.
5. For **Network Load Balancer**, choose **Create**.
6. For **Load balancer name**, enter a name for your load balancer. For example, my-nlb.
7. For **Scheme**, choose **Internet-facing** or **Internal**.

An internet-facing load balancer routes requests from clients to targets over the internet. An internal load balancer routes requires private IP addresses for targets.

8. For **IP address type**, choose the IP addressing for the containers subnets.
9. For **Network mapping**, select the VPC that you used for your EC2 instances. For each Availability Zone that you used to launch your EC2 instances, select the Availability Zone and then select one public subnet for that Availability Zone.

By default, AWS assigns an IPv4 address to each load balancer node from the subnet for its Availability Zone. Alternatively, when you create an internet-facing load balancer, you can select an Elastic IP address for each Availability Zone. This provides your load balancer with static IP addresses.

10. For **Listeners and routing**, keep the default protocol and port, and select your target group from the list. This configures a listener that accepts TCP traffic on port 80 and forwards traffic to the selected target group by default.
11. For **Default action**, select the target group that you created.
12. (Optional) Add tags to categorize your load balancer. Tag keys must be unique for each load balancer. Allowed characters are letters, spaces, numbers (in UTF-8), and the following special characters: + - = . _ : / @. Do not use leading or trailing spaces. Tag values are case-sensitive.
13. Review your configuration, and choose **Create 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 using the console \(p. 456\)](#).

Registering multiple target groups with a service

Your Amazon ECS service can serve traffic from multiple load balancers and expose multiple load balanced ports when you specify multiple target groups in a service definition.

To create a service specifying multiple target groups, you must create the service using the Amazon ECS API, SDK, AWS CLI, or an AWS CloudFormation template. After the service is created, you can view the service and the target groups registered to it with the AWS Management Console. You must use [UpdateService](#) to modify the load balancer configuration of an existing service.

Multiple target groups can be specified in a service definition using the following format. For the full syntax of a service definition, see [Service definition template \(p. 861\)](#).

```
"loadBalancers": [
    {
        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/target_group_name_1/1234567890123456",
        "containerName": "container_name",
        "containerPort": "container_port"
    },
    {

        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/target_group_name_2/6543210987654321",
        "containerName": "container_name",
        "containerPort": "container_port"
    }
]
```

Multiple target group considerations

The following should be considered when you specify multiple target groups in a service definition.

- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- Specifying multiple target groups in a service definition is only supported under the following conditions:
 - The service must use either an Application Load Balancer or Network Load Balancer.
 - The service must use the rolling update (ECS) deployment controller type.
- Specifying multiple target groups is supported for services containing tasks using both the Fargate and EC2 launch types.
- When creating a service that specifies multiple target groups, the Amazon ECS service-linked role must be created. The role is created by omitting the `role` parameter in API requests, or the `Role` property in AWS CloudFormation. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

Example service definitions

Following are a few example use cases for specifying multiple target groups in a service definition. For the full syntax of a service definition, see [Service definition template \(p. 861\)](#).

Example: Having separate load balancers for internal and external traffic

In the following use case, a service uses two separate load balancers, one for internal traffic and a second for internet-facing traffic, for the same container and port.

```

"loadBalancers": [
    //Internal ELB
    {
        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_1/1234567890123456",
            "containerName": "nginx",
            "containerPort": 8080
    },
    //Internet-facing ELB
    {
        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_2/6543210987654321",
            "containerName": "nginx",
            "containerPort": 8080
    }
]

```

Example: Exposing multiple ports from the same container

In the following use case, a service uses one load balancer but exposes multiple ports from the same container. For example, a Jenkins container might expose port 8080 for the Jenkins web interface and port 50000 for the API.

```

"loadBalancers": [
    {

        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_1/1234567890123456",
            "containerName": "jenkins",
            "containerPort": 8080
    },
    {

        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_2/6543210987654321",
            "containerName": "jenkins",
            "containerPort": 50000
    }
]

```

Example: Exposing ports from multiple containers

In the following use case, a service uses one load balancer and two target groups to expose ports from separate containers.

```

"loadBalancers": [
    {

        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_1/1234567890123456",
            "containerName": "webserver",
            "containerPort": 80
    },
    {

        "targetGroupArn": "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
        target_group_name_2/6543210987654321",
            "containerName": "database",
            "containerPort": 3306
    }
]

```

```
}
```

Service auto scaling

Automatic scaling is the ability to increase or decrease the desired count of tasks in your Amazon ECS service automatically. Amazon ECS leverages the Application Auto Scaling service to provide this functionality. For more information, see the [Application Auto Scaling User Guide](#).

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. For more information, see [Service utilization \(p. 556\)](#). You can use these and other CloudWatch metrics to scale out your service (add more tasks) to deal with high demand at peak times, and to scale in your service (run fewer tasks) to reduce costs during periods of low utilization.

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

- [Target tracking scaling policies \(p. 501\)](#)— 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 \(p. 501\)](#)— Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.
- [Scheduled Scaling](#)—Increase or decrease the number of tasks that your service runs based on the date and time.

Service auto scaling and deployments

Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. If you want to suspend scale-out processes while deployments are in progress, take the following steps.

1. Call the [describe-scalable-targets](#) command, specifying the resource ID of the service associated with the scalable target in Application Auto Scaling (Example: service/default/sample-webapp). Record the output. You will need it when you call the next command.
2. Call the [register-scalable-target](#) command, specifying the resource ID, namespace, and scalable dimension. Specify true for both DynamicScalingInSuspended and DynamicScalingOutSuspended.
3. After deployment is complete, you can call the [register-scalable-target](#) command to resume scaling.

For more information, see [Suspending and resuming scaling for Application Auto Scaling](#).

IAM permissions required for service auto scaling

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.

In addition to the standard IAM permissions for creating and updating services, you must give your users, groups, or roles permissions to interact with Service Auto Scaling settings as shown in the following example policy.

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:*",
                "ecs:DescribeServices",
                "ecs:UpdateService",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch>ListMetrics",
                "cloudwatch:DisableAlarmActions",
                "cloudwatch:EnableAlarmActions",
                "iam>CreateServiceLinkedRole",
                "sns>CreateTopic",
                "sns:Subscribe",
                "sns:Get*",
                "sns>List*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}

```

The [Create service example \(p. 609\)](#) and [Update service example \(p. 610\)](#) IAM policy examples show the required permissions to use Service Auto Scaling in the AWS Management Console.

The Application Auto Scaling service also needs permission to describe your Amazon ECS services and CloudWatch alarms, and permissions to modify your service's desired count on your behalf. The sns : permissions are for the notifications that CloudWatch sends to an Amazon SNS topic when a threshold has been exceeded. If you use automatic scaling for your Amazon ECS services, it creates a service-linked role named AWSServiceRoleForApplicationAutoScaling_ECSService. This service-linked role grants Application Auto Scaling permission to describe the alarms for your policies, to monitor the current running task count of the service, and to modify the desired count of the service. The original managed Amazon ECS role for Application Auto Scaling was ecsAutoscaleRole, but it is no longer required. The service-linked role is the default role for Application Auto Scaling. For more information, see [Service-linked roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

If you created your Amazon ECS container instance role before CloudWatch metrics are available for Amazon ECS, you might need to add the ecs:StartTelemetrySession permission. For more information, see [Using CloudWatch metrics \(p. 547\)](#).

Considerations

When using scaling policies, consider the following:

- Amazon ECS sends metrics in 1-minute intervals to CloudWatch. Metrics are not available until the clusters and services send the metrics to CloudWatch, and you cannot create CloudWatch alarms for metrics that do not exist.
- The scaling policies support a cooldown period. This is the number of seconds to wait for a previous scaling activity to take effect.
 - For scale-out events, the intention is to continuously (but not excessively) scale out. After Service Auto Scaling successfully scales out using a scaling policy, it starts to calculate the cooldown time. The scaling policy won't increase the desired capacity again unless either a larger scale out is

initiated or the cooldown period ends. While the scale-out cooldown period is in effect, the capacity added by the initiating scale-out activity is calculated as part of the desired capacity for the next scale-out activity.

- For scale-in events, the intention is to scale in conservatively to protect your application's availability, so scale-in activities are blocked until the cooldown period has expired. However, if another alarm initiates a scale-out activity during the scale-in cooldown period, Service Auto Scaling scales out the target immediately. In this case, the scale-in cooldown period stops and doesn't complete.
- The 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, Service 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, Service Auto Scaling scales the desired count out 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. This is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that might not be satisfied, for example, if there aren't 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.
- If you want your task count to scale to zero when there's no work to be done, set a minimum capacity of 0. With target tracking scaling policies, when actual capacity is 0 and the metric indicates that there is workload demand, Service Auto Scaling waits for one data point to be sent before scaling out. In this case, it scales out by the minimum possible amount as a starting point and then resumes scaling based on the actual running task count.
- Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see [Service auto scaling and deployments \(p. 498\)](#).
- You have several Application Auto Scaling options for Amazon ECS tasks. Target tracking is the easiest mode to use. With it, all you need to do is set a target value for a metric, such as CPU average utilization. Then, the auto scaler automatically manages the number of tasks that are needed to attain that value. With step scaling you can more quickly react to changes in demand, because you define the specific thresholds for your scaling metrics, and how many tasks to add or remove when the thresholds are crossed. And, more importantly, you can react very quickly to changes in demand by minimizing the amount of time a threshold alarm is in breach.

AWS CLI and SDK experience

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](#).

To configure scaling policies for your Amazon ECS service using the AWS CLI

1. Register your Amazon ECS service as a scalable target using the [register-scalable-target](#) command.
2. Create a scaling policy using the [put-scaling-policy](#) command.
3. [Step scaling] Create an alarm that triggers the scaling policy using the [put-metric-alarm](#) command.

For more information about configuring scaling policies using the AWS CLI, see the [Application Auto Scaling User Guide](#).

Target tracking scaling policies

With target tracking scaling policies, you select a metric and set a target value. Amazon ECS Service Auto Scaling creates and manages the CloudWatch alarms that control 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.

Considerations

Consider the following when using target tracking policies:

- 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.
- You may see gaps between the target value and the actual metric data points. This is because Service 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.
- To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually.
- Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see [Service auto scaling and deployments \(p. 498\)](#).
- You can have multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The intention of Service Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion turned on) are ready to scale in.
- Do not edit or delete the CloudWatch alarms that Service Auto Scaling manages for a target tracking scaling policy. Service Auto Scaling deletes the alarms automatically when you delete the scaling policy.
- The ALBRequestCountPerTarget metric for target tracking scaling policies is not supported for the blue/green deployment type.

Step scaling policies

With step scaling policies, you create and manage the CloudWatch alarms that initiate the scaling process. If the target tracking alarms don't work for your use case, you can use step scaling. You can also

use target tracking scaling with step scaling for an advanced scaling policy configuration. For example, you can configure a more aggressive response when utilization reaches a certain level.

When you create a step scaling policy, you specify one or more step adjustments that automatically scale the number of instances dynamically based on the size of the alarm breach. Each step adjustment specifies the following:

- A lower bound for the metric value
- An upper bound for the metric value
- The amount by which to scale, based on the scaling adjustment type

CloudWatch aggregates metric data points based on the statistic for the metric that is associated with your CloudWatch alarm. When the alarm is breached, the appropriate scaling policy is invoked. Application Auto Scaling applies the aggregation type to the most recent metric data points from CloudWatch (as opposed to the raw metric data). It compares this aggregated metric value against the upper and lower bounds defined by the step adjustments to determine which step adjustment to perform.

Interconnecting services

Applications that run in Amazon ECS tasks often need to receive connections from the internet or to connect to other applications that run in Amazon ECS services. If you need external connections from the internet, we recommend using Elastic Load Balancing. For more information about integrated load balancing, see [the section called “Service load balancing” \(p. 486\)](#).

Choosing an interconnection method

If you need an application to connect to other applications that run in Amazon ECS services, Amazon ECS provides three ways to do this without a load balancer:

- *Amazon ECS Service Connect*

Amazon ECS Service Connect provides management of service-to-service communication as Amazon ECS configuration. It does this by building both service discovery and a service mesh in Amazon ECS. This provides the complete configuration inside each Amazon ECS service that you manage by service deployments, a unified way to refer to your services within namespaces that doesn't depend on the Amazon VPC DNS configuration, and standardized metrics and logs to monitor all of your applications on Amazon ECS. Amazon ECS Service Connect only interconnects Amazon ECS services.

You must configure any cross-VPC connectivity that you want to use with Amazon ECS Service Connect. There's no additional Amazon VPC or network infrastructure configuration required for service-to-service communication when using Service Connect beyond the cross-VPC connectivity. Service Connect configures each task for your applications to discover services. Service Connect configures DNS names for your services in the task itself, and doesn't require nor create DNS records in your hosted zones.

For more information, see [Service Connect \(p. 503\)](#).

- *Amazon ECS service discovery*

Amazon ECS service discovery integrates services with AWS Cloud Map namespaces to add entries (specifically, AWS Cloud Map service instances) to the namespace for each task in the Amazon ECS service. To connect, an app resolves these entries as DNS hostname records or uses the AWS Cloud Map API to get the IP address of the tasks.

Amazon ECS service discovery can be used with any applications, including UDP connections. Service discovery doesn't affect the connecting protocol or traffic route.

For more information, see [Service discovery \(p. 522\)](#)

- **AWS App Mesh**

AWS App Mesh is a service mesh that you can use to monitor and control services. App Mesh standardizes how your services communicate and can help ensure high availability. You can use App Mesh to monitor your applications and network traffic for each of your services and applications that run outside of Amazon ECS.

App Mesh is configured in a new task definition revision and isn't applied to Amazon ECS services. Add the `proxyConfiguration` field to the task definition and a container with the proxy inside of it. You can configure the proxy container for advanced features.

App Mesh has advanced traffic routing for release controls that are unaffected by the Amazon ECS service deployment methods.

For more information, see [Use App Mesh with Amazon ECS \(p. 668\)](#).

Network mode compatibility table

The following table covers the compatibility between these options and the task network modes. In the table, "client" refers to the application that's making the connections from inside an Amazon ECS task.

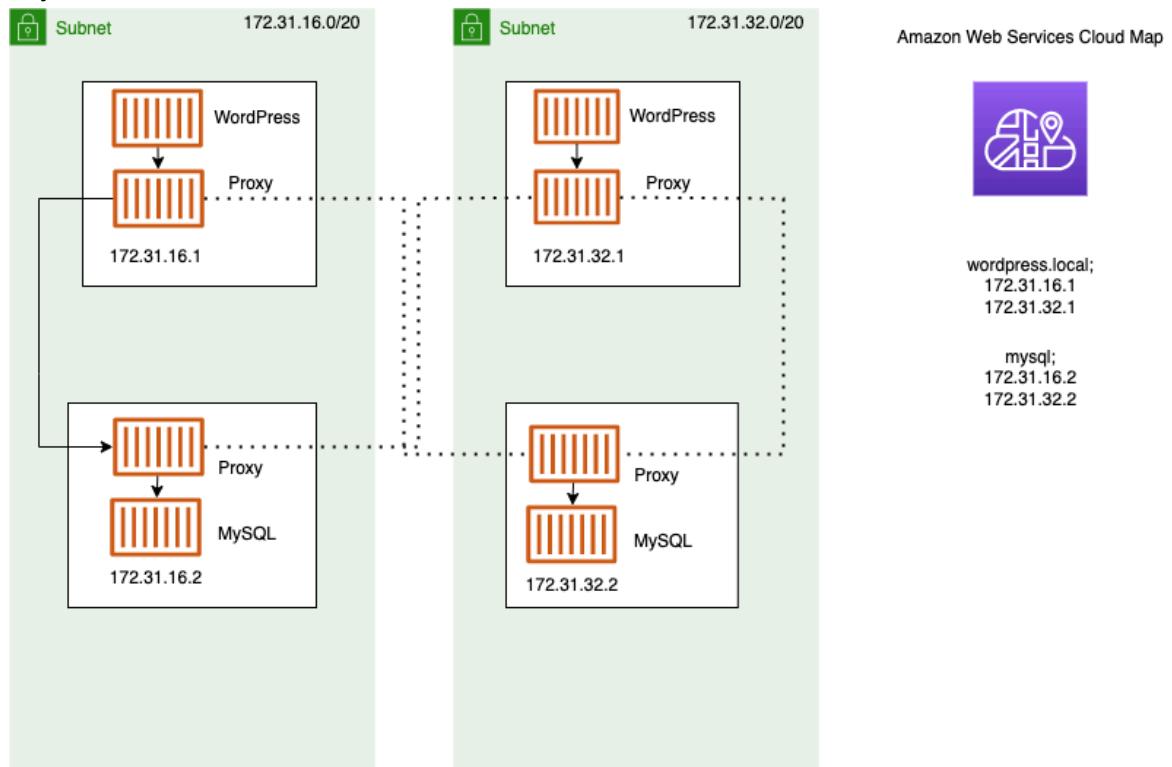
Interconnection Options	Bridged	awsVPC	Host
Service discovery	yes, but requires clients be aware of SRV records in DNS without hostPort.	yes	yes, but requires clients be aware of SRV records in DNS without hostPort.
Service Connect	yes	yes	no
App Mesh	no	yes	no

Service Connect

Amazon ECS Service Connect provides management of service-to-service communication as Amazon ECS configuration. It does this by building both service discovery and a service mesh in Amazon ECS. This provides the complete configuration inside each Amazon ECS service that you manage by service deployments, a unified way to refer to your services within namespaces that doesn't depend on the Amazon VPC DNS configuration, and standardized metrics and logs to monitor all of your applications on Amazon ECS. Amazon ECS Service Connect only interconnects Amazon ECS services.

The following diagram shows an example Service Connect network with 2 subnets in the VPC and 2 services. A client service that runs WordPress with 1 task in each subnets. A server service that runs MySQL with 1 task in each subnet. Both services are highly available and resilient to task and Availability Zone issues because each service runs multiple tasks that are spread out over 2 subnets. The solid arrows show a connection from WordPress to MySQL. For example, a `mysql --host=mysql` CLI command that is run from inside the WordPress container in the task with the IP address 172.31.16.1. The command uses the short name `mysql` on the default port for MySQL. This name and port connects to the Service

Connect proxy in the same task. The proxy in the WordPress task uses round-robin load balancing and any previous failure information in outlier detection to pick which MySQL task to connect to. As shown by the solid arrows in the diagram, the proxy connects to the second proxy in the MySQL task with the IP Address 172.31.16.2. The second proxy connects to the local MySQL server in the same task. Both proxies report connection performance that is visible in graphs in the Amazon ECS and Amazon CloudWatch consoles so that you can get performance metrics from all kinds of applications in the same way.



Overview of steps to configure Service Connect

Follow these steps to configure Service Connect for a group of related services.

1. Add port names to the port mappings in your task definitions. Additionally, you can identify the layer 7 protocol of the application, to get additional metrics.
2. Create an ECS cluster with a AWS Cloud Map namespace or create the namespace separately. For simple organization, create an Amazon ECS cluster with the name that you want for the namespace and specify the identical name for the namespace. In this case, Amazon ECS creates a new HTTP namespace with the necessary configuration. Amazon ECS Service Connect doesn't use or create DNS hosted zones in Amazon Route 53.
3. Configure services to create Service Connect endpoints within the namespace.
4. Deploy services to create the endpoints. Amazon ECS adds a Service Connect proxy container to each task, and creates the Service Connect endpoints in AWS Cloud Map. This container isn't configured in the task definition, and the task definition can be reused without modification to create multiple services in the same namespace or in multiple namespaces.
5. Deploy client apps as services to connect to the endpoints. Amazon ECS connects them to the Service Connect endpoints through the Service Connect proxy in each task.

Applications only use the proxy to connect to Service Connect endpoints. There is no additional configuration to use the proxy. The proxy performs round-robin load balancing, outlier detection, and retries. For more information about the proxy, see [Service Connect proxy \(p. 510\)](#) and [Proxy configuration \(p. 511\)](#).

6. Monitor traffic through the Service Connect proxy in Amazon CloudWatch.

Regions with Service Connect

Amazon ECS Service Connect is available in the following AWS Regions:

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Stockholm)	eu-north-1
Europe (Zurich)	eu-central-2
Israel (Tel Aviv)	il-central-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1
South America (São Paulo)	sa-east-1

Service Connect concepts

The Service Connect feature creates a virtual network of related services. The same service configuration can be used across multiple different namespaces to run independent yet identical sets of applications. Service Connect defines the proxy container in the Amazon ECS service. This way, the same task definition can be used to run identical applications in different namespaces with different Service Connect configurations. Each task that the Amazon ECS service makes runs a proxy container in the task.

Service Connect is suitable for connections between Amazon ECS services within the same namespace. For the following applications, you need to use an additional interconnection method to connect to an Amazon ECS service that is configured with Service Connect:

- Amazon ECS tasks that are configured in other namespaces
- Amazon ECS tasks that aren't configured for Service Connect
- other applications outside of Amazon ECS

These applications can connect through the Service Connect proxy but can't resolve Service Connect endpoint names.

For these applications to resolve the IP addresses of ECS tasks, you need to use another interconnection method. For a list of interconnection methods, see [Choosing an interconnection method \(p. 502\)](#).

Service Connect terminology

The following terms are used with Service Connect.

port name

The Amazon ECS task definition configuration that assigns a name to a particular port mapping. This configuration is only used by Amazon ECS Service Connect.

client alias

The Amazon ECS service configuration that assigns the port number that is used in the endpoint. Additionally, the client alias can assign the DNS name of the endpoint, overriding the discovery name. If a discovery name isn't provided in the Amazon ECS service, the client alias name overrides the port name as the endpoint name. For endpoint examples, see the definition of *endpoint*. Multiple client aliases can be assigned to an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

discovery name

The optional, intermediate name that you can create for a specified port from the task definition. This name is used to create a AWS Cloud Map service. If this name isn't provided, the port name from the task definition is used. Multiple discovery names can be assigned to a specific port an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

AWS Cloud Map service names must be unique within a namespace. Because of this limitation, you can have only one Service Connect configuration without a discovery name for a particular task definition in each namespace.

endpoint

The URL to connect to an API or website. The URL contains the protocol, a DNS name, and the port. For more information about endpoints in general, see [endpoint](#) in the *AWS glossary* in the Amazon Web Services General Reference.

Service Connect creates endpoints that connect to Amazon ECS services and configures the tasks in Amazon ECS services to connect to the endpoints. The URL contains the protocol, a DNS

name, and the port. You select the protocol and port name in the task definition, as the port must match the application that is inside the container image. In the service, you select each port by name and can assign the DNS name. If you don't specify a DNS name in the Amazon ECS service configuration, the port name from the task definition is used by default. For example, a Service Connect endpoint could be `http://blog:80`, `grpc://checkout:8080`, or `http://_db.production.internal:99`.

Service Connect service

The configuration of a single endpoint in an Amazon ECS service. This is a part of the Service Connect configuration, consisting of a single row in the **Service Connect and discovery name configuration** in the console, or one object in the services list in the JSON configuration of an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

For more information, see [ServiceConnectService](#) in the Amazon Elastic Container Service API Reference.

namespace

The short name or full Amazon Resource Name (ARN) of the AWS Cloud Map namespace for use with Service Connect. The namespace must be in the same AWS Region as the Amazon ECS service and cluster. The type of namespace in AWS Cloud Map doesn't affect Service Connect.

Service Connect uses the AWS Cloud Map namespace as a logical grouping of Amazon ECS tasks that talk to one another. Each Amazon ECS service can belong to only one namespace. The services within a namespace can be spread across different Amazon ECS clusters within the same AWS Region in the same AWS account. Because each cluster can run tasks of every operating system, CPU architecture, VPC, and EC2, Fargate, and External types, you can freely organize your services by any criteria that you choose.

client service

An Amazon ECS service that runs a network client application. This service must have a namespace configured. Each task in the service can discover and connect to all of the endpoints in the namespace through a Service Connect proxy container.

If any of your containers in the task need to connect to an endpoint from a service in a namespace, choose a client service. If a frontend, reverse proxy, or load balancer application receives external traffic through other methods such as from Elastic Load Balancing, it could use this type of Service Connect configuration.

client-server service

An Amazon ECS service that runs a network or web service application. This service must have a namespace and at least one endpoint configured. Each task in the service is reachable by using the endpoints. The Service Connect proxy container listens on the endpoint name and port to direct traffic to the app containers in the task.

If any of the containers expose and listen on a port for network traffic, choose a client-server service. These applications don't need to connect to other client-server services in the same namespace, but the client configuration is configured. A backend, middleware, business tier, or most microservices would use this type of Service Connect configuration. If you want a frontend, reverse proxy, or load balancer application to receive traffic from other services configured with Service Connect in the same namespace, these services should use this type of Service Connect configuration.

Cluster configuration

You can set a default namespace for Service Connect when you create the cluster or by updating the cluster. If you specify a namespace name that doesn't exist in the same AWS Region and account, a new HTTP namespace is created.

If you create a cluster and specify a default Service Connect namespace, the cluster waits in the PROVISIONING status while Amazon ECS creates the namespace. You can see an attachment in the status of the cluster that shows the status of the namespace. Attachments aren't displayed by default in the AWS CLI, you must add `--include ATTACHMENTS` to see them.

Service Connect service configuration

Service Connect is designed to require the minimum configuration. You need to set a name for each port mapping that you would like to use with Service Connect in the task definition. In the service, you need to turn on Service Connect and select a namespace to make a client service. To make a client-server service, you need to add a single Service Connect service configuration that matches the name of one of the port mappings. Amazon ECS reuses the port number and port name from the task definition to define the Service Connect service and endpoint. To override those values, you can use the other parameters **Discovery**, **DNS**, and **Port** in the console, or `discoveryName` and `clientAliases`, respectively in the Amazon ECS API.

The following example shows each kind of Service Connect configuration being used together in the same Amazon ECS service. Shell comments are provided, however note that the JSON configuration used to Amazon ECS services doesn't support comments.

```
{
    ...
    serviceConnectConfiguration: {
        enabled: true,
        namespace: "internal",
        #config for client services can end here, only these two parameters are required.
        services: [
            {
                portName: "http"
            }, #minimal client - server service config can end here.portName must match the
            "name"
                parameter of a port mapping in the task definition. {
                    discoveryName: "http-second"
                    #name the discoveryName to avoid a Task def port name collision with the
                    minimal config in the same Cloud Map namespace
                    portName: "http"
                },
                {
                    clientAliases: [
                        {
                            dnsName: "db",
                            port: 81
                        } #use when the port in Task def is not the port that client apps
                        use.Client apps can use http://db:81 to connect
                            discoveryName: "http-three"
                            portName: "http"
                        },
                        {
                            clientAliases: [
                                {
                                    dnsName: "db.app",
                                    port: 81
                                } #use when the port in Task def is not the port that client apps
                                use.duplicates are fine as long as the discoveryName is different.
                                    discoveryName: "http-four"
                                    portName: "http",
                                    ingressPortOverride: 99 #If App should also accept traffic directly on Task
def port.
        ]
    }
}
```

Deployment order

When you use Amazon ECS Service Connect, you configure each Amazon ECS service either to run a server application that receives network requests (client-server service) or to run a client application that makes the requests (client service).

When you prepare to start using Service Connect, start with a client-server service. You can add a Service Connect configuration to a new service or an existing service. After you edit and update an Amazon ECS service to add a Service Connect configuration, Amazon ECS creates a Service Connect endpoint in the namespace. Additionally, Amazon ECS creates a new deployment in the service to replace the tasks that are currently running.

Existing tasks and other applications can continue to connect to existing endpoints, and external applications. If a client-server service adds tasks by scaling out, new connections from clients will be balanced between all of the tasks immediately. If a client-server service is updated, new connections from clients will be balanced between the tasks of the new version immediately.

Existing tasks can't resolve and connect to the new endpoint. Only new Amazon ECS tasks that have a Service Connect configuration in the same namespace and that start running after this deployment can resolve and connect to this endpoint. For example, an Amazon ECS service that runs a client application must be redeployed to connect to a new database server endpoint. Start the client deployment after the deployment completes for the server.

This means that the operator of the client application determines when the configuration of their app changes, even though the operator of the server application can change their configuration at any time. The list of endpoints in the namespace can change every time that any Amazon ECS service in the namespace is deployed, but existing tasks and replacement tasks continue to behave the same as they did after the most recent deployment.

Consider the following examples.

First, assume that you are creating an application that is available to the public internet in a single AWS CloudFormation template and single AWS CloudFormation stack. The public discovery and reachability should be created last by AWS CloudFormation, including the frontend client service. The services need to be created in this order to prevent an time period when the frontend client service is running and available the public, but a backend isn't. This eliminates error messages from being sent to the public during that time period. In AWS CloudFormation, you must use the `dependsOn` to indicate to AWS CloudFormation that multiple Amazon ECS services can't be made in parallel or simultaneously. You should add the `dependsOn` to the frontend client service for each backend client-server service that the client tasks connect to.

Second, assume that a frontend service exists without Service Connect configuration. The tasks are connecting to an existing backend service. Add a client-server Service Connect configuration to the backend service first, using the same name in the `DNS` or `clientAlias` that the frontend uses. This creates a new deployment, so all the deployment rollback detection or AWS Management Console, AWS CLI, AWS SDKs and other methods to roll back and revert the backend service to the previous deployment and configuration. If you are satisfied with the performance and behavior of the backend service, add a client or client-server Service Connect configuration to the frontend service. Only the tasks in the new deployment use the Service Connect proxy that is added to those new tasks. If you have issues with this configuration, you can roll back and revert to your previous configuration by using the deployment rollback detection or AWS Management Console, AWS CLI, AWS SDKs and other methods to roll back and revert the backend service to the previous deployment and configuration. If you use another service discovery system that is based on DNS instead of Service Connect, any frontend or client applications begin using new endpoints and changed endpoint configuration after the local DNS cache expires, commonly taking multiple hours.

Networking

In the default configuration, the Service Connect proxy listens on the `containerPort` from the port mapping in the task definition. You need rules in your security group to allow ingress to this port from the VPC CIDRs, or specifically from subnets where clients will run.

Even if you set a port number in the Service Connect service configuration, this doesn't change the port for the client-server service that the Service Connect proxy listens on. When you set this port number, Amazon ECS changes the port of the endpoint that the client services connect to, on the Service Connect proxy inside those tasks. The proxy in the client service connects to the proxy in the client-server service using the `containerPort`.

If you want to change the port that the Service Connect proxy listens on, change the `ingressPortOverride` in the Service Connect configuration of the client-server service. If you change this port number, you must allow inbound traffic on this port in the Amazon VPC security group that is used by traffic to this service.

Traffic that your applications send to Amazon ECS services configured for Service Connect require that the Amazon VPC and subnets have route table rules and network ACL rules that allow the `containerPort` and `ingressOverridePort` port numbers that you are using.

You can send traffic between VPCs with Service Connect. You must consider the same requirements for the route table rules, network ACLs, and security groups as they apply to both VPCs.

For example, two clusters create tasks in different VPCs. A service in each cluster is configured to use the same namespace. The applications in these two services can resolve every endpoint in the namespace without any VPC DNS configuration. However, the proxies can't connect unless the VPC peering, VPC or subnet route tables, and VPC network ACLs allow the traffic on the `containerPort` and `ingressOverridePort` port numbers you are using.

Service Connect proxy

If you create or update an Amazon ECS service with Service Connect configuration, Amazon ECS adds a new container to each new task as it is started. This pattern of using a separate container is called a *sidecar*. This container isn't present in the task definition and you can't configure it. Amazon ECS manages the configuration of this container in the Amazon ECS service. Because of this, you can reuse the same task definitions between multiple Amazon ECS services, namespaces, and you can run tasks without Service Connect also.

Proxy resources

The only configuration for this container in the task definition is the task CPU and memory limits. The only container configuration for this container in the ECS service is the log configuration inside the Service Connect configuration. For more information about the proxy configuration, see [Proxy configuration \(p. 511\)](#).

The task definition must set the task memory limit to use Service Connect. The additional CPU and memory in the task limits that you don't allocate in the container limits in your other containers are used by the Service Connect proxy container and other containers that don't set container limits.

We recommend adding 256 CPU units and at least 64 MiB of memory to your task CPU and memory for the Service Connect proxy container. On AWS Fargate, the lowest amount of memory that you can set is 512 MiB of memory. On Amazon EC2, task memory is optional, but it is required for Service Connect.

If you expect tasks in this service to receive more than 500 requests per second at their peak load, we recommend adding 512 CPU units to your task CPU in this task definition for the Service Connect proxy container.

If you expect to create more than 100 Service Connect services in the namespace or 2000 tasks in total across all Amazon ECS services within the namespace, we recommend adding 128 MiB of memory to

your task memory for the Service Connect proxy container. You should do this in every task definition that is used by all of the Amazon ECS services in the namespace.

Proxy configuration

Your applications connect to the proxy in the sidecar container in the same task as the application is in. Amazon ECS configures the task and containers so that applications only connect to the proxy if the application is connecting to the endpoint names in the same namespace. All other traffic doesn't use the proxy. The other traffic includes IP addresses in the same VPC, AWS service endpoints, and external traffic.

Load balancing

Load balancing strategies: round-robin

Service Connect configures the proxy to use the round-robin strategy for load balancing between the tasks in a Service Connect endpoint. The local proxy that is in the task where the connection comes from, picks one of the tasks in the client-server service that provides the endpoint.

For example, consider a task that runs WordPress in an Amazon ECS service that is configured as a *client service* in a namespace called *local*. There is another service with 2 tasks that run the MySQL database. This service is configured to provide an endpoint called *mysql* through Service Connect in the same namespace. In the WordPress task, the WordPress application connects to the database using the endpoint name. Because of the Service Connect configuration, connections to this name go to the proxy that runs in a sidecar container in the same task. Then, the proxy can connect to either of the MySQL tasks using the round-robin strategy.

Outlier detection

Service Connect configures the *outlier detection* feature of the proxy to provide passive health checks. This feature uses data that the proxy has about prior failed connections to avoid sending new connections to the hosts that had the failed connections.

For example, consider a task that runs WordPress in an Amazon ECS service that is configured as a *client service* in a namespace called *local*. There is another service with 2 tasks that run the MySQL database. This service is configured to provide an endpoint called *mysql* through Service Connect in the same namespace. In the WordPress task, the WordPress application connects to the proxy that runs in a sidecar container in the same task. The proxy can connect to either of the MySQL tasks. If the proxy made multiple connections to a specific MySQL task, and 5 or more of the connections failed in the last 30 seconds, then the proxy avoids that MySQL task for 30 to 300 seconds.

Retries

Number of retries: 2

Service Connect configures the proxy to retry connection that pass through the proxy and fail, and the second attempt avoids using the host from the previous connection. This ensures that each connection through Service Connect doesn't fail for one-off reasons.

Timeout

Upstream timeout: 15 seconds

Service Connect configures the proxy to wait a maximum time for your client-server applications to respond. The default timeout value is 15 seconds and can't be changed.

Service Connect parameters

The following parameters have extra fields when using Service Connect.

Parameter location	App type	Description	Required?
Task definition	Client	There are no changes available for Service Connect in client task definitions.	N/A
Task definition	Client-server	Servers must add name fields to ports in the <code>portMappings</code> of containers. For more information, see portMappings (p. 807)	Yes
Task definition	Client-server	Servers can optionally provide an application protocol (for example, HTTP) to receive protocol-specific metrics for their server applications (for example, HTTP 5xx).	No
Service definition	Client	Client services must add a <code>serviceConnectConfiguration</code> to configure the namespace to join. This namespace must contain all of the server services that this service needs to discover. For more information, see serviceConnectConfiguration (p. 857) .	Yes
Service definition	Client-server	Server services must add a <code>serviceConnectConfiguration</code> to configure the DNS names, port numbers, and namespace that the service is available from. For more information, see serviceConnectConfiguration (p. 857) .	Yes
Cluster	Client	Clusters can add a default Service Connect namespace. New services in the cluster inherit the namespace when Service Connect is configured in a service. For more information, see Amazon ECS clusters .	No
Cluster	Client-server	There are no changes available for Service Connect in clusters that apply to server services. Server task definitions and services must set the respective configuration.	N/A

Service Connect considerations

- Windows containers aren't supported with Service Connect.
- Tasks that run in Fargate must use the Fargate Linux platform version 1.4.0 or higher to use Service Connect.
- The ECS agent version on the container instance must be 1.67.2 or higher.
- Container instances must run the Amazon ECS-optimized Amazon Linux 2023 AMI version 20230428 or later, or Amazon ECS-optimized Amazon Linux 2 AMI version 2.0.20221115 to use Service Connect. These versions have the Service Connect agent in addition to the Amazon ECS container agent. For more information about the Service Connect agent, see [Amazon ECS Service Connect Agent](#) on GitHub.

- Container instances must have the `ecs:Poll` permission for the resource `arn:aws:ecs:region:0123456789012:task-set/cluster/*`. If you are using the `ecsInstanceRole`, you don't need to add additional permissions. The `AmazonEC2ContainerServiceforEC2Role` managed policy has the necessary permissions. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
- External container instance for Amazon ECS Anywhere aren't supported with Service Connect.
- Only services that use rolling deployments are supported with Service Connect. Services that use the *blue/green* and *external* deployment types aren't supported.
- Task definitions must set the task memory limit to use Service Connect. For more information, see [Service Connect proxy \(p. 510\)](#).
- Task definitions that set container memory limits for all containers instead of setting the task memory limit aren't supported.

You can set container memory limits on your containers, but you must set the task memory limit to a number greater than the sum of the container memory limits. The additional CPU and memory in the task limits that aren't allocated in the container limits are used by the Service Connect proxy container and other containers that don't set container limits. For more information, see [Service Connect proxy \(p. 510\)](#).

- You can configure Service Connect in a service to use any AWS Cloud Map namespace in the same AWS Region in the same AWS account.
 - Each Amazon ECS service can belong to only one namespace.
 - Only the tasks that Amazon ECS services create are supported. Standalone tasks can't be configured for Service Connect.
 - All endpoints must be unique within a namespace.
 - All discovery names must be unique within a namespace.
 - Existing services must be redeployed before the applications in them can resolve new endpoints. New endpoints that are added to the namespace after the most recent deployment won't be added to the task configuration. For more information, see [the section called "Deployment order" \(p. 509\)](#).
 - You can create a namespace when creating a new cluster. Amazon ECS Service Connect doesn't delete namespaces when clusters are deleted. You must delete namespaces directly in AWS Cloud Map if you are done using them.
 - Service Connect doesn't support HTTP 1.0.

Service Connect console experience

Service Connect management is available only in the new Amazon ECS console.

To create a new namespace, either create a new Amazon ECS cluster using the Amazon ECS console and specify a namespace name to create, or use the AWS Cloud Map console. Amazon ECS Service Connect can use any *instance discovery* type of AWS Cloud Map namespace. We recommend the *API calls* type to make the minimum amount of additional resources. To create a new Amazon ECS cluster and namespace in the Amazon ECS console, see [Creating a cluster for the Fargate launch type using the console \(p. 236\)](#).

Every AWS Cloud Map namespace in this AWS account in the selected AWS Region is displayed in the **Namespaces** in the Amazon ECS console.

To delete a namespace, use the AWS Cloud Map console. A namespace must be empty before it can be deleted.

To create a new Amazon ECS task definition, or register a new revision to an existing task definition and use Service Connect, see [Creating a task definition using the console \(p. 125\)](#).

To create a new Amazon ECS service that uses Service Connect, see [Creating a service using the console \(p. 456\)](#).

Service Connect pricing

Amazon ECS Service Connect pricing depends on whether you use AWS Fargate or Amazon EC2 infrastructure to host your containerized workloads. When using Amazon ECS on AWS Outposts, the pricing follows the same model that's used when you use Amazon EC2 directly. For more information, see [Amazon ECS Pricing](#).

Tutorial: Using Service Connect in Fargate with the AWS CLI

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

Amazon ECS supports the Service Connect feature in the AWS Regions listed in [Regions with Service Connect](#).

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, see [Installing the AWS Command Line Interface](#).
- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have a VPC, subnet, route table, and security group created to use. For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).
- You have a task execution role with the name `ecsTaskExecutionRole` and the `AmazonECSTaskExecutionRolePolicy` managed policy is attached to the role. This role allows Fargate to write the NGINX application logs and Service Connect proxy logs to Amazon CloudWatch Logs. For more information, see [Creating the task execution \(ecsTaskExecutionRole\) role \(p. 627\)](#).

Step 1: Create the Amazon ECS cluster

Use the following steps to create your Amazon ECS cluster and namespace.

To create the Amazon ECS cluster and AWS Cloud Map namespace

1. Create an Amazon ECS cluster named `tutorial` to use. The parameter `--service-connect-defaults` sets the default namespace of the cluster. In the example output, a AWS Cloud Map namespace of the name `service-connect` doesn't exist in this account and AWS Region, so the namespace is created by Amazon ECS. The namespace is made in AWS Cloud Map in the account, and is visible with all of the other namespaces, so use a name that indicates the purpose.

```
aws ecs create-cluster --cluster-name tutorial --service-connect-defaults
namespace=service-connect
```

Output:

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "clusterName": "tutorial",
        "serviceConnectDefaults": {
            "namespace": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
EXAMPLE"
```

```

        },
        "status": "PROVISIONING",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": [],
        "attachments": [
            {
                "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                "type": "sc",
                "status": "ATTACHING",
                "details": []
            }
        ],
        "attachmentsStatus": "UPDATE_IN_PROGRESS"
    }
}
}

```

2. Verify that the cluster is created:

```
aws ecs describe-clusters --clusters tutorial
```

Output:

```
{
    "clusters": [
        {
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
            "clusterName": "tutorial",
            "serviceConnectDefaults": {
                "namespace": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-EXAMPLE"
            },
            "status": "ACTIVE",
            "registeredContainerInstancesCount": 0,
            "runningTasksCount": 0,
            "pendingTasksCount": 0,
            "activeServicesCount": 0,
            "statistics": [],
            "tags": [],
            "settings": [],
            "capacityProviders": [],
            "defaultCapacityProviderStrategy": []
        }
    ],
    "failures": []
}
```

3. (Optional) Verify that the namespace is created in AWS Cloud Map. You can use the AWS Management Console or the normal AWS CLI configuration as this is created in AWS Cloud Map.

For example, use the AWS CLI:

```
aws servicediscovery --region us-west-2 get-namespace --id ns-EXAMPLE
```

Output:

```
{  
    "Namespace": {  
        "Id": "ns-EXAMPLE",  
        "Arn": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-EXAMPLE",  
        "Name": "service-connect",  
        "Type": "HTTP",  
        "Properties": {  
            "DnsProperties": {  
                "SOA": {}  
            },  
            "HttpProperties": {  
                "HttpName": "service-connect"  
            }  
        },  
        "CreateDate": 1661749852.422,  
        "CreatorRequestId": "service-connect"  
    }  
}
```

Step 2: Create the Amazon ECS service for the server

The Service Connect feature is intended for interconnecting multiple applications on Amazon ECS. At least one of those applications needs to provide a web service to connect to. In this step, you create:

- The task definition that uses the unmodified official NGINX container image and includes Service Connect configuration.
- The Amazon ECS service definition that configures Service Connect to provide service discovery and service mesh proxying for traffic to this service. The configuration reuses the default namespace from the cluster configuration to reduce the amount of service configuration that you make for each service.
- The Amazon ECS service. It runs one task using the task definition, and inserts an additional container for the Service Connect proxy. The proxy listens on the port from the container port mapping of the task definition. In a client application running in Amazon ECS, the proxy in the client task listens for outbound connections to the task definition port name, service discovery name or service client alias name, and the port number from the client alias.

To create the web service with Amazon ECS Service Connect

1. Register a task definition that's compatible with Fargate and uses the awsvpc network mode. Follow these steps:
 - a. Create a file that's named `service-connect-nginx.json` with the contents of the following task definition.

This task definition configures Service Connect by adding `name` and `appProtocol` parameters to the port mapping. The port name makes this port more identifiable in the service configuration when multiple ports are used. The port name is also used by default as the discoverable name for use by other applications in the namespace.

Important

This task definition uses a logConfiguration to send the nginx output from stdout and stderr to Amazon CloudWatch Logs. This task execution role doesn't have the extra permissions required to make the CloudWatch Logs log group. Create the log group in CloudWatch Logs using the AWS Management Console or AWS CLI. If you don't want to send the nginx logs to CloudWatch Logs you can remove the logConfiguration.

Replace the AWS account id in the execution role with your AWS account id.

```
{  
    "family": "service-connect-nginx",  
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",  
    "networkMode": "awsvpc",  
    "containerDefinitions": [  
        {  
            "name": "webserver",  
            "image": "public.ecr.aws/docker/library/nginx:latest",  
            "cpu": 100,  
            "portMappings": [  
                {  
                    "name": "nginx",  
                    "containerPort": 80,  
                    "protocol": "tcp",  
                    "appProtocol": "http"  
                }  
            ],  
            "essential": true,  
            "logConfiguration": {  
                "logDriver": "awslogs",  
                "options": {  
                    "awslogs-group": "/ecs/service-connect-nginx",  
                    "awslogs-region": "us-west-2",  
                    "awslogs-stream-prefix": "nginx"  
                }  
            }  
        },  
        {  
            "cpu": "256",  
            "memory": "512"  
        }  
    ]  
}
```

- b. Register the task definition using the service-connect-nginx.json file:

```
aws ecs register-task-definition --cli-input-json file://service-connect-nginx.json
```

2. Create an ECS service by following these steps:

- a. Create a file that's named service-connect-nginx-service.json with the contents of the Amazon ECS service that you're creating. This example uses the task definition that was created in the previous step. An awsvpcConfiguration is required because the example task definition uses the awsvpc network mode.

When you create the ECS service, specify the Fargate launch type, and the LATEST platform version that supports Service Connect. The securityGroups and subnets must belong to a VPC that has the requirements for using Amazon ECS. You can obtain the security group and subnet IDs from the Amazon VPC Console.

This service configures Service Connect by adding the serviceConnectConfiguration parameter. The namespace is not required because the cluster has a default namespace

configured. Client applications running in ECS in the namespace connect to this service by using the `portName` and the port in the `clientAliases`. For example, this service is reachable using `http://nginx:80/`, as nginx provides a welcome page in the root location `/`. External applications that are not running in Amazon ECS or are not in the same namespace can reach this application through the Service Connect proxy by using the IP address of the task and the port number from the task definition.

This service uses a `logConfiguration` to send the service connect proxy output from `stdout` and `stderr` to Amazon CloudWatch Logs. This task execution role doesn't have the extra permissions required to make the CloudWatch Logs log group. Create the log group in CloudWatch Logs using the AWS Management Console or AWS CLI. We recommend that you create this log group and store the proxy logs in CloudWatch Logs. If you don't want to send the proxy logs to CloudWatch Logs you can remove the `logConfiguration`.

```
{
    "cluster": "tutorial",
    "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 0
    },
    "deploymentController": {
        "type": "ECS"
    },
    "desiredCount": 1,
    "enableECSManagedTags": true,
    "enableExecuteCommand": true,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [
                "sg-EXAMPLE"
            ],
            "subnets": [
                "subnet-EXAMPLE",
                "subnet-EXAMPLE",
                "subnet-EXAMPLE"
            ]
        }
    },
    "platformVersion": "LATEST",
    "propagateTags": "SERVICE",
    "serviceName": "service-connect-nginx-service",
    "serviceConnectConfiguration": {
        "enabled": true,
        "services": [
            {
                "portName": "nginx",
                "clientAliases": [
                    {
                        "port": 80
                    }
                ]
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "us-west-2",
                "awslogs-stream-prefix": "service-connect-proxy"
            }
        }
    }
}
```

```
        },
    ],
    "taskDefinition": "service-connect-nginx"
}
```

- b. Create an ECS service using the `service-connect-nginx-service.json` file:

```
aws ecs create-service --cluster tutorial --cli-input-json file://service-connect-nginx-service.json
```

Output:

```
{
  "service": {
    "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/tutorial/service-connect-nginx-service",
    "serviceName": "service-connect-nginx-service",
    "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
    "loadBalancers": [],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 1,
    "runningCount": 0,
    "pendingCount": 0,
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "platformFamily": "Linux",
    "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-definition/service-connect-nginx:1",
    "deploymentConfiguration": {
      "deploymentCircuitBreaker": {
        "enable": false,
        "rollback": false
      },
      "maximumPercent": 200,
      "minimumHealthyPercent": 0
    },
    "deployments": [
      {
        "id": "ecs-svc/3763308422771520962",
        "status": "PRIMARY",
        "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-definition/service-connect-nginx:1",
        "desiredCount": 1,
        "pendingCount": 0,
        "runningCount": 0,
        "failedTasks": 0,
        "createdAt": 1661210032.602,
        "updatedAt": 1661210032.602,
        "launchType": "FARGATE",
        "platformVersion": "1.4.0",
        "platformFamily": "Linux",
        "networkConfiguration": {
          "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [
              "sg-EXAMPLE"
            ],
            "subnets": [
              "subnet-EXAMPLEf",
              "subnet-EXAMPLE",
              "subnet-EXAMPLE"
            ]
          }
        }
      }
    ]
  }
}
```

```

        ],
    },
    "rolloutState": "IN_PROGRESS",
    "rolloutStateReason": "ECS deployment ecs-svc/3763308422771520962
in progress.",
    "failedLaunchTaskCount": 0,
    "replacedTaskCount": 0,
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "service-connect",
        "services": [
            {
                "portName": "nginx",
                "clientAliases": [
                    {
                        "port": 80
                    }
                ]
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "us-west-2",
                "awslogs-stream-prefix": "service-connect-proxy"
            },
            "secretOptions": []
        }
    },
    "serviceConnectResources": [
        {
            "discoveryName": "nginx",
            "discoveryArn": "arn:aws:servicediscovery:us-
west-2:123456789012:service/srv-EXAMPLE"
        }
    ]
},
"roleArn": "arn:aws:iam::123456789012:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
"version": 0,
"events": [],
"createdAt": 1661210032.602,
"placementConstraints": [],
"placementStrategy": [],
"networkConfiguration": {
    "awsvpcConfiguration": {
        "assignPublicIp": "ENABLED",
        "securityGroups": [
            "sg-EXAMPLE"
        ],
        "subnets": [
            "subnet-EXAMPLE",
            "subnet-EXAMPLE",
            "subnet-EXAMPLE"
        ]
    }
},
"schedulingStrategy": "REPLICAS",
"enableECSManagedTags": true,
"propagateTags": "SERVICE",
"enableExecuteCommand": true
}
}

```

The `serviceConnectConfiguration` that you provided appears inside the first *deployment* of the output. As you make changes to the ECS service in ways that need to make changes to tasks, a new deployment is created by Amazon ECS.

Step 3: Verify that you can connect

To verify that Service Connect is configured and working, follow these steps to connect to the web service from an external application. Then, see the additional metrics in CloudWatch that are created by the Service Connect proxy.

To connect to the web service from an external application

- Connect to the task IP address and container port using the task IP address

Use the AWS CLI to get the task ID, using the `aws ecs list-tasks --cluster tutorial`.

If your subnets and security group permit traffic from the public internet on the port from the task definition, you can connect to the public IP from your computer. The public IP isn't available from ``describe-tasks`` however, so the steps involve going to the Amazon EC2 AWS Management Console or AWS CLI to get the details of the elastic network interface.

In this example, an Amazon EC2 instance in the same VPC uses the private IP of the task. The application is nginx, but the `server: envoy` header shows that the Service Connect proxy is used. The Service Connect proxy is listening on the container port from the task definition.

```
$ curl -v 10.0.19.50:80/
*   Trying 10.0.19.50:80...
* Connected to 10.0.19.50 (10.0.19.50) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.19.50
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< server: envoy
< date: Tue, 23 Aug 2022 03:53:06 GMT
< content-type: text/html
< content-length: 612
< last-modified: Tue, 16 Apr 2019 13:08:19 GMT
< etag: "5cb5d3c3-264"
< accept-ranges: bytes
< x-envoy-upstream-service-time: 0
<
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

To view Service Connect metrics

The Service Connect proxy creates application (HTTP, HTTP2, gRPC, or TCP connection) metrics in CloudWatch metrics. When you use the CloudWatch console, see the additional metric dimensions of **DiscoveryName**, (**DiscoveryName**, **ServiceName**, **ClusterName**), **TargetDiscoveryName**, and (**TargetDiscoveryName**, **ServiceName**, **ClusterName**) under the ECS namespace. For more information about these metrics and the dimensions, see [Available metrics and dimensions \(p. 548\)](#).

Service discovery

Your Amazon ECS service can optionally be configured to use Amazon ECS service discovery. Service discovery uses AWS Cloud Map API actions to manage HTTP and DNS namespaces for your Amazon ECS services. For more information, see [What Is AWS Cloud Map?](#) in the *AWS Cloud Map Developer Guide*.

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
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4

Region Name	Region
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Stockholm)	eu-north-1
Israel (Tel Aviv)	il-central-1
Europe (Spain)	eu-south-2
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Service Discovery concepts

Service discovery consists of the following components:

- **Service discovery namespace:** A logical group of service discovery services that share the same domain name, such as example.com. This is the domain name where you want to route traffic to. You can create a namespace with a call to the `aws servicediscovery create-private-dns-namespace` command or in the Amazon ECS classic console. You can use the `aws servicediscovery list-namespaces` command to view the summary information about the namespaces that were created by the current account. For more information about the service discovery commands, see [create-private-dns-namespace](#) and [list-namespaces](#) in the *AWS Cloud Map (service discovery) AWS CLI Reference Guide*.
- **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 registry:** Allows you to look up a service via DNS or AWS Cloud Map API actions and get back one or more available endpoints that can be used to connect to the service.
 - **Service discovery instance:** Exists within the service discovery service and consists of the attributes associated with each Amazon ECS service in the service directory.
 - **Instance attributes:** The following metadata is added as custom attributes for each Amazon ECS service that is configured to use service discovery:

- **AWS_INSTANCE_IPV4** – For an A record, the IPv4 address that Route 53 returns in response to DNS queries and AWS Cloud Map returns when discovering instance details, for example, 192.0.2.44.
- **AWS_INSTANCE_PORT** – The port value associated with the service discovery service.
- **AVAILABILITY_ZONE** – The Availability Zone into which the task was launched. For tasks using the EC2 launch type, this is the Availability Zone in which the container instance exists. For tasks using the Fargate launch type, this is the Availability Zone in which the elastic network interface exists.
- **REGION** – The Region in which the task exists.
- **ECS_SERVICE_NAME** – The name of the Amazon ECS service to which the task belongs.
- **ECS_CLUSTER_NAME** – The name of the Amazon ECS cluster to which the task belongs.
- **EC2_INSTANCE_ID** – The ID of the container instance the task was placed on. This custom attribute is not added if the task is using the Fargate launch type.
- **ECS_TASK_DEFINITION_FAMILY** – The task definition family that the task is using.
- **ECS_TASK_SET_EXTERNAL_ID** – If a task set is created for an external deployment and is associated with a service discovery registry, then the **ECS_TASK_SET_EXTERNAL_ID** attribute will contain the external ID of the task set.
- **Amazon ECS health checks:** Amazon ECS performs periodic container-level health checks. If an endpoint does not pass the health check, it is removed from DNS routing and marked as unhealthy.

Service discovery considerations

The following should be considered when using service discovery:

- Service discovery is supported for tasks on Fargate that use platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 77\)](#).
- Services configured to use service discovery have a limit of 1,000 tasks per service. This is due to a Route 53 service quota.
- The Create Service workflow in the Amazon ECS console only supports registering services into private DNS namespaces. When an AWS Cloud Map private DNS namespace is created, a Route 53 private hosted zone will be created automatically.
- The VPC DNS attributes must be configured for successful DNS resolution. For information about how to configure the attributes, see [DNS support in your VPC](#) in the *Amazon VPC User Guide*.
- 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 service task definition 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 service task definition uses the bridge or host network mode, the SRV record is the only supported DNS record type. Create a 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>.
- 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 for each task.
- If you have eight or fewer healthy records, Route 53 responds to all DNS queries with all of the healthy records.
- When all records are unhealthy, Route 53 responds to DNS queries with up to eight unhealthy records.

- You can configure service discovery for a service that's behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.
- Service discovery doesn't support the use of Classic Load Balancers.
- We recommend you use container-level health checks managed by Amazon ECS for your service discovery service.
 - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and health checks, and your task state, to update the health with AWS Cloud Map. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see [HealthCheckCustomConfig](#) in the [AWS Cloud Map API Reference](#).
- If you're using the Amazon ECS classic console, the workflow creates one service discovery service for each service. It 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 creating a service. Updating existing services to configure service discovery for the first time or change the current configuration isn't supported.
- The AWS Cloud Map resources created when service discovery is used must be cleaned up manually.

Amazon ECS classic console experience

The workflow to create a service in the Amazon ECS classic console supports service discovery. 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 isn't supported.

To create a new Amazon ECS service that uses service discovery, see [Creating an Amazon ECS service in the classic console \(p. 899\)](#).

Service discovery pricing

Customers using Amazon ECS service discovery are charged for Route 53 resources and AWS Cloud Map discovery API operations. This involves costs for creating the Route 53 hosted zones and queries to the service registry. For more information, see [AWS Cloud Map Pricing](#) in the [AWS Cloud Map Developer Guide](#).

Amazon ECS performs container level health checks and exposes them to AWS Cloud Map custom health check API operations. This is currently made available to customers at no extra cost. If you configure additional network health checks for publicly exposed tasks, you're charged for those health checks.

Task scale-in protection

You can use Amazon ECS task scale-in protection to protect your tasks from being terminated by scale-in events from either [Service Auto Scaling](#) or [deployments](#).

Certain applications require a mechanism to safeguard mission-critical tasks from termination by scale-in events during times of low utilization or during service deployments. For example:

- You have a queue-processing asynchronous application such as a video transcoding job where some tasks need to run for hours even when cumulative service utilization is low.
- You have a gaming application that runs game servers as Amazon ECS tasks that need to continue running even if all users have logged-out to reduce startup latency of a server reboot.
- When you deploy a new code version, you need tasks to continue running because it would be expensive to reprocess.

To protect tasks belonging to your service from terminating in a scale-in event, set the `protectionEnabled` attribute to `true`. By default, tasks are protected for 2 hours. You can customize the protection period by using the `expiresInMinutes` attribute. You can protect your tasks for a minimum of 1 minute and up to a maximum of 2880 minutes (48 hours).

After a task finishes its requisite work, you can set the `protectionEnabled` attribute to `false`, allowing the task to be terminated by subsequent scale-in events.

Task scale-in protection mechanisms

You can set and get task scale-in protection using either the Amazon ECS container agent endpoint or the Amazon ECS API.

Set task scale-in protection

You can set task scale-in protection in the following ways:

- **Amazon ECS container agent endpoint**

We recommend using the Amazon ECS container agent endpoint for tasks that can self-determine the need to be protected. Use this approach for queue-based or job-processing workloads.

When a container starts processing work, for example by consuming an SQS message, you can set the `ProtectionEnabled` attribute through the task scale-in protection endpoint path `$ECS_AGENT_URI/task-protection/v1/state` from within the container. Amazon ECS will not terminate this task during scale-in events. After your task finishes its work, you can unset the `ProtectionEnabled` attribute using the same endpoint, making the task eligible for termination during subsequent scale-in events.

For more information on using the Amazon ECS container agent endpoint, see [Task scale-in protection endpoint \(p. 407\)](#).

- **Amazon ECS API**

You can use the Amazon ECS API to set task scale-in protection if your application has a component that tracks the status of active tasks. Use `UpdateTaskProtection` to mark one or more tasks as protected.

An example of this approach would be if your application is hosting game server sessions as Amazon ECS tasks. When a user logs in to a session on the server (task), you can mark the task as protected. After the user logs out, you can either clear the protection specifically for this task or periodically clear protection for similar tasks that no longer have active sessions, depending on your requirement to keep idle servers.

For more information, see [UpdateTaskProtection](#) in the *Amazon Elastic Container Service API Reference*.

You can combine both approaches. For example, use the Amazon ECS agent endpoint to set task protection from within a container and use the Amazon ECS API to remove task protection from your external controller service.

Get task protection status

To get the protection status of tasks in an Amazon ECS service, you can do one of the following:

- **Amazon ECS container agent endpoint**

Configure the container definition to use the Amazon task scale-in protection endpoint path. For more information, see [Task scale-in protection endpoint \(p. 407\)](#).

- **Amazon ECS API**

Use `GetTaskProtection`. For more information, see [GetTaskProtection](#) in the *Amazon Elastic Container Service API Reference*.

Task scale-in protection considerations

Consider the following points before using task scale-in protection:

- We recommend using the Amazon ECS container agent endpoint because the Amazon ECS agent has built-in retry mechanisms and a simpler interface.
- You can reset the task scale-in protection expiration period by calling `UpdateTaskProtection` for a task that already has protection turned on.
- Determine how long a task would need to complete its requisite work and set the `expiresInMinutes` property accordingly. If you set the protection expiration longer than necessary, then you will incur costs and face delays in the deployment of new tasks.
- Deployment considerations:
 - If the service uses a rolling update, new tasks will be created but tasks running older version will not be terminated until `protectionEnabled` is cleared or expires. You can adjust the `maximumPercentage` parameter in deployment configuration to a value that allows new tasks to be created when old tasks are protected.
 - If a blue/green update is applied, the blue deployment with protected tasks will not be removed if tasks have `protectionEnabled`. Traffic will be diverted to the new tasks that come up and older tasks will only be removed when `protectionEnabled` is unset or expires. Depending on the timeout of the CodeDeploy or CloudFormation updates, the deployment may timeout and the older Blue tasks may still be present.
 - If you use CloudFormation, the update-stack has a 3 hour timeout. Therefore, if you set your task protection for longer than 3 hours, then your CloudFormation deployment may result in failure and rollback.

During the time your old tasks are protected, the CloudFormation stack shows `UPDATE_IN_PROGRESS`. If task scale-in protection is removed or expires within the 3 hour window, your deployment will succeed and move to the `UPDATE_COMPLETE` status. If the deployment is stuck in `UPDATE_IN_PROGRESS` for more than 3 hours, it will fail and show `UPDATE_FAILED` state, and will then be rolled back to old task set.

- Amazon ECS sends service events when protected tasks keep a deployment (rolling or blue/green) from reaching the steady state, so that you can take remedial actions. While trying to update the protection status of a task, if you receive a `DEPLOYMENT_BLOCKED` error message, it means the service has more protected tasks than the desired count of tasks for the service. To resolve this error, do one of the following:
 - Wait for the current task protection to expire. Then set task protection.
 - Determine which tasks can be stopped. Then use `UpdateTaskProtection` with the `protectionEnabled` option set to `false` for these tasks.
 - Increase the desired task count of the service to more than the number of protected tasks.

IAM permissions required for task scale-in protection

If you plan to use the Amazon ECS container agent API to get or update task protection, the task must have the Amazon ECS task role with the following permissions:

- `ecs:GetTaskProtection`: Allows the Amazon ECS container agent to call `GetTaskProtection`.

- `ecs:UpdateTaskProtection`: Allows the Amazon ECS container agent to call `UpdateTaskProtection`.

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 a service repeatedly fail to enter the RUNNING state (progressing directly from a PENDING to a STOPPED status), then the time between subsequent restart attempts is incrementally increased up to a maximum of 15 minutes. This maximum period is subject to change in the future. This behavior reduces the effect that failing tasks have on your Amazon ECS cluster resources or Fargate infrastructure costs. If your service initiates the throttle logic, you receive the following [service event message \(p. 773\)](#):

(service `service-name`) is unable to consistently start tasks successfully.

Amazon ECS doesn't ever stop a failing service from retrying. It also doesn't attempt to modify it in any way other than increasing the time between restarts. The service throttle logic doesn't 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 using the console \(p. 465\)](#).

The following are some common causes that initiate this logic:

- A lack of resources to host your task with, such as ports, memory, or CPU units in your cluster. In this case, you also see the [insufficient resource service event message \(p. 771\)](#).
- The Amazon ECS container agent can't pull your task Docker image. This might be because 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 \(p. 766\)](#).
- Insufficient disk space on your container instance to create the container. In this case, you also see `CannotCreateContainerError` in your [stopped task errors \(p. 766\)](#). For more information, see [CannotCreateContainerError: API error \(500\): devmapper \(p. 777\)](#).

Important

Tasks that are stopped after they reach the RUNNING state don't start the throttle logic or the associated service event message. For example, assume that failed Elastic Load Balancing health checks for a service cause a task to be flagged as unhealthy, and Amazon ECS deregisters it and stops the task. At this point, the tasks aren't throttled. Even if a task's container command immediately exits with a non-zero exit code, the task already moved to the RUNNING state. Tasks that fail immediately because command errors don't cause the throttle or the service event message.

Resources and tags

Amazon ECS resources are assigned an Amazon Resource Name (ARN) and a unique resource identifier (ID). These resources include task definitions, clusters, tasks, services, and container instances. You can tag these resources with values that you define to help you organize and identify them.

The following topics provide an overview about these resources and tags and describe how you can use tags.

Contents

- [Tagging your Amazon ECS resources \(p. 529\)](#)
- [Amazon ECS service quotas \(p. 536\)](#)
- [Supported Regions for Amazon ECS on AWS Fargate \(p. 540\)](#)
- [Amazon ECS usage reports \(p. 543\)](#)

Tagging your Amazon ECS resources

To help you manage your Amazon ECS resources, you can optionally assign your own metadata to each resource using *tags*. Each *tag* consists of a *key* and an optional *value*.

You can use tags to categorize your Amazon ECS 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 that you assigned to it. For example, you can define a set of tags for your account's Amazon ECS container instances. This helps you track each instance's owner and stack level.

You can use tags for your Cost and Usage reports. You can use these reports to analyze the cost and usage of your Amazon ECS resources. For more information, see [the section called "Usage Reports" \(p. 543\)](#).

Warning

Tag keys and their values are returned by many different API calls. Denying access to `DescribeTags` doesn't automatically deny access to tags returned by other APIs. As a best practice, we recommend that you do not include sensitive data in your tags.

We recommend that you devise a set of tag keys that meets your needs for each resource type. You can use a consistent set of tag keys for easier management of 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. 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 earlier value. If you delete a resource, any tags for the resource are also deleted.

If you use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

How resources are tagged

There are multiple ways that Amazon ECS tasks, services, task definitions, and clusters are tagged:

- A user manually tags a resource by using the AWS Management Console, Amazon ECS API, the AWS CLI, or an AWS SDK.

- A user creates a service or runs a standalone task and selects the Amazon ECS-managed tags option.

Amazon ECS automatically tags all newly launched tasks. For more information, see [the section called "Amazon ECS-managed tags" \(p. 532\)](#).

- A user creates a resource using the console. The console automatically tags the resources.

These tags are returned in the AWS CLI, and AWS SDK responses and are displayed in the console. You cannot modify or delete these tags.

For information about the added tags, see the **Tags automatically added by the console** column in the **Tagging support for Amazon ECS resources** table.

If you specify tags when you create a resource and the tags can't be applied, Amazon ECS rolls back the 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 while they're being created, you can eliminate the need to run custom tagging scripts after resource creation.

The following table describes the Amazon ECS resources that support tagging.

Tagging support for Amazon ECS resources

Resource	Supports tags	Supports tag propagation	Tags automatically added by the console
Amazon ECS tasks	Yes	Yes, from the task definition.	<i>Key:</i> <code>aws:ecs:clusterName</code> <i>Value:</i> <code>cluster-name</code>
Amazon ECS services	Yes	Yes, from either the task definition or the service to the tasks in the service.	<i>Key:</i> <code>ecs:service:stackId</code> <i>Value</i> <code>arn:aws:cloudformation:arn</code>
Amazon ECS task sets	Yes	No	N/A
Amazon ECS task definitions	Yes	No	<i>Key:</i> <code>ecs:taskDefinition:createdFrom</code> <i>Value:</i> <code>ecs-console-v2</code>
Amazon ECS clusters	Yes	No	<i>Key:</i> <code>aws:cloudformation:logical-id</code> <i>Value:</i> <code>ECSCluster</code> <i>Key:</i> <code>aws:cloudformation:stack-id</code> <i>Value:</i> <code>arn:aws:cloudformation:arn</code> <i>Key:</i> <code>aws:cloudformation:stack-name</code>

Resource	Supports tags	Supports tag propagation	Tags automatically added by the console
			<i>Value: ECS-Console-V2-Cluster-EXAMPLE</i>
Amazon ECS container instances	Yes	Yes, from the Amazon EC2 instance. For more information, see Adding tags to an Amazon EC2 container instance (p. 534) .	N/A
Amazon ECS External instances	Yes	No	N/A
Amazon ECS capacity provider	Yes. You cannot tag the predefined FARGATE and FARGATE_SPOT capacity providers.	No	N/A

Tagging resources on creation

The following resources support tagging on creation using the Amazon ECS API, AWS CLI, AWS SDK:

- Amazon ECS tasks
- Amazon ECS services
- Amazon ECS task definition
- Amazon ECS task sets
- Amazon ECS clusters
- Amazon ECS container instances
- Amazon ECS capacity providers

Amazon ECS has the option to use tagging authorization for resource creation. In order to use the feature, perform the following steps:

- You must opt into the feature. For more information, see [the section called "Tagging authorization" \(p. 248\)](#).
- After you opt in, users must have permissions for actions that creates the resource, such as `ecsCreateCluster`. If tags are specified in the resource-creating action, AWS performs additional authorization to verify if users or roles have permissions to create tags. Therefore, you must grant explicit permissions to use the `ecs:TagResource` action. For more information, see [the section called "Tag resources during creation" \(p. 651\)](#).

Tag restrictions

The following restrictions apply to tags:

- A maximum of 50 tags can be associated with a resource.
- Tag keys can't be repeated for one resource. Each tag key must be unique, and can only have one value.

- Keys can be up to 128 characters long in UTF-8.
- Values can be up to 256 characters long in UTF-8.
- If multiple AWS services and resources use your tagging schema, limit the types of characters you use. Some services might have restrictions on allowed characters. Generally, allowed characters are letters, numbers, spaces, and the following characters: + - = . _ : / @.
- Tag keys and values are case sensitive.
- You can't use aws :, AWS :, or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags-per-resource limit.

Amazon ECS-managed tags

When you use Amazon ECS-managed tags, Amazon ECS automatically tags all newly launched tasks with cluster information and either the user added task definition tags or the service tags. The following describes the added tags:

- Standalone tasks – a tag with a *Key* as aws:ecs:clusterName and a *Value* set to the cluster name. All task definition tags that were added by users.
- Tasks that are part of a service – a tag with a *Key* as aws:ecs:clusterName and a *Value* set to the cluster name. A tag with a *Key* as aws:ecs:serviceName and a *Value* set to the service name. Tags from one of the following resources:
 - Task definitions – All task definition tags that were added by users.
 - Services – All service tags that were added by users.

The following options are required for this feature:

- You must opt in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 246\)](#).
- When you use the APIs to create a service or run a task, you must set enableECSManagedTags to true for run-task and create-service. For more information, see [create-service](#) and [run-task](#) in the *AWS Command Line Interface API Reference*.

Tagging your resources for billing

You can use Amazon ECS-managed tags or user-added tags for your Cost and Usage Report. For more information, see [Amazon ECS usage reports \(p. 543\)](#).

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 User Guide*.

Additionally, you can turn on *Split Cost Allocation Data* to get task-level CPU and memory usage data in your Cost and Usage Reports. For more information, see [Task-level Cost and Usage Reports \(p. 544\)](#).

Note

If you've turned on reporting, it can take up to 24 hours before the data for the current month is available for viewing.

Working with tags using the console

Using the Amazon ECS console, you can manage the tags that are 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) that supports tags, you can view and manage its tags on the **Tags** tab.

Warning

As a best practice, we recommend that you do not include sensitive data in your tags.

Contents

- [Adding tags on an individual resource during launch \(p. 533\)](#)
- [Managing individual resource tags using the console \(p. 533\)](#)
- [Adding tags to an Amazon EC2 container instance \(p. 534\)](#)
- [Adding tags to an external container instance \(p. 535\)](#)

Adding tags on an individual resource during launch

You can use the following resources to specify tags when you create the resource.

Task	Console
Run one or more tasks.	Running a standalone task using the Amazon ECS console (p. 430)
Create a service.	Creating a service using the console (p. 456)
Create a task set.	External deployment (p. 481)
Register a task definition.	the section called “Creating a task definition using the console” (p. 125)
Create a cluster.	Creating a cluster for the Fargate launch type using the console (p. 236)
Run one or more container instances.	Launching an Amazon ECS Linux container instance (p. 272)

Managing individual resource tags using the console

Amazon ECS allows you to add or delete tags that are associated with your clusters, services, tasks, and task definitions directly from the resource's page. For information about tagging your container instances, see [Adding tags to an Amazon EC2 container instance \(p. 534\)](#).

Warning

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

To modify a tag for an individual resource

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, select a resource type (for example, **Clusters**).
4. Select the resource from the resource list, choose the **Tags** tab, and then choose **Manage tags**.
5. Configure your tags.

[Add a tag] Choose **Add tag**, and then do the following:

- For **Key**, enter the key name.
- For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

6. Choose **Save**.

Adding tags to an Amazon EC2 container instance

You can associate tags with your container instances using one of the following methods:

- Method 1 – When creating the container instance using the Amazon EC2 API, CLI, or console, specify tags by passing user data to the instance using the container agent configuration parameter **ECS_CONTAINER_INSTANCE_TAGS**. This creates tags that are associated with the container instance in Amazon ECS only, they cannot be listed using the Amazon EC2 API. For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 280\)](#).

Important

If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the **ECS_CONTAINER_INSTANCE_TAGS** agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

The following is an example of a user data script that associates tags with your container instance:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Method 2 – When you create your container instance using the Amazon EC2 API, CLI, or console, first specify tags using the **TagSpecification.N** parameter. Then, pass user data to the instance by using the container agent configuration parameter **ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM**. Doing so propagates them from Amazon EC2 to Amazon ECS.

The following is an example of a user data script that propagates the tags that are associated with an Amazon EC2 instance, and registers the instance with a cluster that's named **MyCluster**.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

To provide access to allow container instance tags to propagate from Amazon EC2 to Amazon ECS, manually add the following permissions as an inline policy to the Amazon ECS container instance IAM role. For more information, see [Adding and Removing IAM Policies](#).

- `ec2:DescribeTags`

The following is an example policy that's used to add these permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
    ]
}
```

Adding tags to an external container instance

You can associate tags with your external container instances by using one of the following methods.

- Method 1 – Before running the installation script to register your external instance with your cluster, create or edit the Amazon ECS container agent configuration file at `/etc/ecs/ecs.config` and add the `ECS_CONTAINER_INSTANCE_TAGS` container agent configuration parameter. This creates tags that are associated with the external instance.

The following is example syntax.

```
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
```

- Method 2 – After your external instance is registered to your cluster, you can use the AWS Management Console to add tags. For more information, see [Managing individual resource tags using the console \(p. 533\)](#).

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.

Warning

Don't add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags aren't intended to be used for private or sensitive data.

Tagging support for Amazon ECS resources

Task	AWS CLI	API action
Add or overwrite one or more tags.	tag-resource	TagResource
Delete one or more tags.	untag-resource	UntagResource

You can use some resource-creating actions to specify tags when you create the resource. The following actions support tagging on creation.

You must have the `ecsTagResource` permission. For more information, see [Grant permission to tag resources on creation \(p. 651\)](#).

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
Create a task set.	create-task-set	New-ECSTaskSet	CreateTaskSet
Register a task definition.	register-task-definition	Register-ECSTaskDefinition	RegisterTaskDefinition
Create a cluster.	create-cluster	New-ECSCluster	CreateCluster
Run one or more container instances.	run-instances	New-EC2Instance	RunInstances

Amazon ECS service quotas

The following tables provide the default service quotas, also referred to as limits, for Amazon ECS for an AWS account. For more information about the service quotas for other AWS services that you can use with Amazon ECS, such as Elastic Load Balancing and Auto Scaling, see [AWS service quotas](#) in the *Amazon Web Services General Reference*. For information about API throttling in the Amazon ECS API, see [Request throttling for the Amazon ECS API](#).

Amazon ECS service quotas

The following are Amazon ECS service quotas.

New AWS accounts might have initial lower quotas that can increase over time. Amazon ECS constantly monitors the account usage within each Region, and then automatically increases the quotas based on your usage. You can also request a quota increase for values that are shown as adjustable, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Name	Default	Adjust	Description
Capacity providers per cluster	Each supported Region: 20	No	The maximum number of capacity providers that can be associated with a cluster.
Classic Load Balancers per service	Each supported Region: 1	No	The maximum number of Classic Load Balancers per service.
Clusters per account	Each supported Region: 10,000	Yes	Number of clusters per account
Container instances per cluster	Each supported Region: 5,000	No	Number of container instances per cluster

Name	Default	Adjust	Description
Container instances per start-task	Each supported Region: 10	No	The maximum number of container instances specified in a StartTask API action.
Containers per task definition	Each supported Region: 10	No	The maximum number of containers definitions within a task definition.
ECS Exec sessions	Each supported Region: 1,000	No	The maximum number of ECS Exec sessions per container.
Rate of tasks launched by a service on AWS Fargate	Each supported Region: 500	No	The maximum number of tasks that can be provisioned per service per minute on Fargate by the Amazon ECS service scheduler.
Rate of tasks launched by a service on an Amazon EC2 or External instance	Each supported Region: 500	No	The maximum number of tasks that can be provisioned per service per minute on an Amazon EC2 or External instance by the Amazon ECS service scheduler.
Revisions per task definition family	Each supported Region: 1,000,000	No	The maximum number of revisions per task definition family. Deregistering a task definition revision does not exclude it from being included in this limit.
Security groups per awsvpcConfiguration	Each supported Region: 5	No	The maximum number of security groups specified within an awsvpcConfiguration.
Services per cluster	Each supported Region: 5,000	<u>Yes</u>	The maximum number of services per cluster
Services per namespace	Each supported Region: 100	<u>Yes</u>	The maximum number of services that can be running within a namespace.
Subnets per awsvpcConfiguration	Each supported Region: 16	No	The maximum number of subnets specified within an awsvpcConfiguration.
Tags per resource	Each supported Region: 50	No	The maximum number of tags per resource. This applies to task definitions, clusters, tasks, and services.

Name	Default	Adjust	Description
Target groups per service	Each supported Region: 5	No	The maximum number of target groups per service, if using an Application Load Balancer or a Network Load Balancer.
Task definition size	Each supported Region: 64 Kilobytes	No	The maximum size, in KiB, of a task definition.
Tasks in PROVISIONING state per cluster	Each supported Region: 500	No	The maximum number of tasks waiting in the PROVISIONING state per cluster. This quota only applies to tasks launched using an EC2 Auto Scaling group capacity provider.
Tasks launched per run-task	Each supported Region: 10	No	The maximum number of tasks that can be launched per RunTask API action.
Tasks per service	Each supported Region: 5,000	No	The maximum number of tasks per service (the desired count).

Note

The default values are the initial quotas set by AWS, which are separate from the actual applied quota value and maximum possible service quota. For more information, see [Terminology in Service Quotas](#) in the [Service Quotas User Guide](#).

Note

Services configured to use Amazon ECS service discovery have a limit of 1,000 tasks per service. This is due to the AWS Cloud Map service quota for the number of instances per service. For more information, see [AWS Cloud Map service quotas](#) in the [Amazon Web Services General Reference](#).

Note

In practice, task launch rates are also dependent on other considerations such as container images to be downloaded and unpacked, health checks and other integrations enabled, such as registering tasks with a load balancer. You might see variations in task launch rates compared with the quotas that are represented here. These variations are causes by the features that you have enabled for your Amazon ECS services. For more information, see [speeding up Amazon ECS deployments](#) in the [Amazon ECS Best Practices Guide](#).

Note

Services configured to use Amazon ECS Service Connect have a limit of 1,000 tasks per service. This is due to the AWS Cloud Map service quota for the number of instances per service. For more information, see [AWS Cloud Map service quotas](#) in the [Amazon Web Services General Reference](#).

AWS Fargate service quotas

The following are Amazon ECS on AWS Fargate service quotas and are listed under the **AWS Fargate** service in the Service Quotas console.

New AWS accounts might have initial lower quotas that can increase over time. Fargate constantly monitors the account usage within each Region, and then automatically increases the quotas based

on your usage. You can also request a quota increase for values that are shown as adjustable, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Name	Default	Adjust	Description
Fargate On-Demand vCPU resource count	Each supported Region: 6	Yes	The number of Fargate vCPUs running concurrently as Fargate On-Demand in this account in the current Region.
Fargate Spot vCPU resource count	Each supported Region: 6	Yes	The number of Fargate vCPUs running concurrently as Fargate Spot in this account in the current Region.

Note

The default values are the initial quotas set by AWS, which are separate from the actual applied quota value and maximum possible service quota. For more information, see [Terminology in Service Quotas](#) in the *Service Quotas User Guide*.

Note

Fargate additionally enforces Amazon ECS tasks and Amazon EKS pods launch rate limits. For more information, see [Fargate throttling limits](#).

Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console

Amazon ECS has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see [What Is Service Quotas?](#) in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your Amazon ECS service quotas.

AWS Management Console

To view Amazon ECS and Fargate service quotas using the AWS Management Console

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Container Service (Amazon ECS) or AWS Fargate**.

In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

4. To view additional information about a service quota, such as the description, choose the quota name.
5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the AWS Management Console see the [Service Quotas User Guide](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

AWS CLI

To view Amazon ECS and Fargate service quotas using the AWS CLI

Run the following command to view the default Amazon ECS quotas.

```
aws service-quotas list-aws-default-service-quotas \
--query 'Quotas[*]'.
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode} ' \
--service-code ecs \
--output table
```

Run the following command to view the default Fargate quotas.

```
aws service-quotas list-aws-default-service-quotas \
--query 'Quotas[*]'.
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode} ' \
--service-code fargate \
--output table
```

Run the following command to view your applied Fargate quotas.

```
aws service-quotas list-service-quotas \
--service-code fargate
```

Note

Amazon ECS doesn't support applied quotas.

For more information about working with service quotas using the AWS CLI, see the [Service Quotas AWS CLI Command Reference](#). To request a quota increase, see the [request-service-quota-increase](#) command in the [AWS CLI Command Reference](#).

Supported Regions for Amazon ECS on AWS Fargate

Contents

- [Supported Regions for Linux containers on AWS Fargate \(p. 540\)](#)
- [Supported Regions for Windows containers on AWS Fargate \(p. 542\)](#)

Supported Regions for Linux containers on AWS Fargate

Amazon ECS Linux containers on AWS Fargate are supported in the following AWS Regions. The supported Availability Zone IDs are noted when applicable.

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1 (usw1-az1 & usw1-az3 only)

Region Name	Region
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Tokyo)	ap-northeast-1 (apne1-az1, apne1-az2, & apne1-az4 only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1 (cnn1-az1 & cnn1-az2 only)
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Israel (Tel Aviv)	il-central-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Supported Regions for Windows containers on AWS Fargate

Amazon ECS Windows containers on AWS Fargate are supported in the following AWS Regions. The supported Availability Zone IDs are noted when applicable.

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1 (use1-az1, use1-az2, use1-az4, use1-az5, & use1-az6only)
US West (N. California)	us-west-1 (usw1-az1 & usw1-az3 only)
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Tokyo)	ap-northeast-1 (apne1-az1, apne1-az2, & apne1-az4 only)
Canada (Central)	ca-central-1 (cac1-az1 & cac1-az2 only)
China (Beijing)	cn-north-1 (cnn1-az1 & cnn1-az2 only)
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1

Region Name	Region
South America (São Paulo)	sa-east-1
Israel (Tel Aviv)	il-central-1
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1

Amazon ECS usage reports

AWS provides a reporting tool called Cost Explorer that you can use to analyze the cost and usage of your Amazon ECS resources.

You can use Cost Explorer to view charts of your usage and costs. You can view data from the last 13 months, and forecast how much you're 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. For example, you can use it to 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 and Usage Report shows usage across all of your Amazon ECS tasks. The metering data includes CPU usage as vCPU-Hours and memory usage as GB-Hours for each task that was run. How that data is presented depends on the launch type of the task.

For tasks using the Fargate launch type, the lineItem/Operation column shows FargateTask and you will see the cost associated with each task.

For tasks that use the EC2 launch type, the lineItem/Operation column shows ECSTask-EC2 and the tasks don't have a direct cost associated with them. The metering data that's shown in the report, such as memory usage, represents the total resources that the task reserved over the billing period that you specify. You can use this data to determine the cost of your underlying cluster of Amazon EC2 instances. The cost and usage data for your Amazon EC2 instances are listed separately under the Amazon EC2 service.

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 \(p. 532\)](#).

Important

The metering data is only viewable for tasks that are launched on or after November 16, 2018. Tasks that are launched before this date don't show metering data.

The following is an example of some of the fields that you can use to sort cost allocation data in Cost Explorer.

- Cluster name
- Service name
- Resource tags
- Launch type
- AWS Region
- Usage type

For more information about creating an AWS Cost and Usage Report, see [AWS Cost and Usage Report](#) in the [AWS Billing User Guide](#).

Task-level Cost and Usage Reports

AWS Cost Management can provide CPU and memory usage data in the AWS Cost and Usage Report for each task on Amazon ECS, including tasks on Fargate and tasks on EC2. This data is called *Split Cost Allocation Data*. You can use this data to analyze costs and usage for applications. Additionally, you can split and allocate the costs to individual business units and teams with cost allocation tags and cost categories. For more information about *Split Cost Allocation Data*, see [Understanding split cost allocation data](#) in the *AWS Cost and Usage Report User Guide*.

You can opt in to task-level *Split Cost Allocation Data* for the account in the AWS Cost Management Console. If you have a management (payer) account, you can opt in from the payer account to apply this configuration to every linked account.

After you set up *Split Cost Allocation Data*, there will be additional columns under the **splitLineItem** header in the report. For more information see [Split line item details](#) in the *AWS Cost and Usage Report User Guide*.

For tasks on EC2, this data splits the cost of the EC2 instance based on the resource usage or reservations and the remaining resources on the instance.

Prerequisites for Task-level CURs

- To use *Split Cost Allocation Data*, you must create a report, and select **Split cost allocation data**. For more information, see [Creating Cost and Usage Reports](#) in the *AWS Cost and Usage Report User Guide.*
- The minimum Docker version for reliable metrics is Docker version v20.10.13 and newer, which is included in Amazon ECS-optimized AMI 20220607 and newer.
- Ensure that the ECS agent has the `ECS_DISABLE_METRICS` configuration set to `false`. When this setting is `false`, the ECS agent sends metrics to Amazon CloudWatch. On Linux, this setting is `false` by default and metrics are sent to CloudWatch. On Windows, this setting is `true` by default, so you must change the setting to `false` to send the metrics to CloudWatch for AWS Cost Management to use. For more information about ECS agent configuration, see [Amazon ECS container agent configuration \(p. 370\)](#).

Note

AWS Cost Management calculates the *Split Cost Allocation Data* with the task CPU and memory usage. AWS Cost Management can use the task CPU and memory reservation instead of the usage, if the usage is unavailable. If you see the CUR is using the reservations, check that your container instances meet the prerequisites and the task resource usage metrics appear in CloudWatch.

Setting up Task-level Cost and Usage Reports

You can turn on *Split Cost Allocation Data* for ECS in the Cost Management Console, AWS Command Line Interface, or the AWS SDKs.

There are two steps to use *Split Cost Allocation Data*. First, you opt in to *Split Cost Allocation Data*. Second, you include the data in a new or existing report. For the steps in the Cost Management Console, see [Enabling split cost allocation data](#) in the *AWS Cost and Usage Report User Guide*.

Then, you can view the report. You can use the Billing and Cost Management console or view the report files in Amazon Simple Storage Service.

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?

The metrics made available 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 own and need to monitor the EC2 instances that make up your underlying infrastructure. Additional CPU and memory reservation and utilization metrics are made available at the cluster, service, and task level.

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.

To establish a baseline you should, at a minimum, monitor the following items:

- The CPU and memory reservation and utilization metrics for your Amazon ECS clusters
- The CPU and memory utilization metrics for your Amazon ECS services

Topics

- [Monitoring tools \(p. 546\)](#)
- [Amazon ECS CloudWatch metrics \(p. 547\)](#)
- [Amazon ECS events and EventBridge \(p. 558\)](#)
- [Amazon ECS CloudWatch Container Insights \(p. 572\)](#)
- [Container instance health \(p. 576\)](#)
- [Collecting application trace data \(p. 576\)](#)
- [Collecting application metrics \(p. 579\)](#)
- [Logging Amazon ECS API calls with AWS CloudTrail \(p. 584\)](#)
- [AWS Compute Optimizer recommendations \(p. 586\)](#)

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 \(p. 547\)](#).

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.

For your container instances that were launched with the Amazon ECS-optimized Amazon Linux AMI, you can use CloudWatch Logs to view different logs from your container instances in one convenient location. You must install the CloudWatch agent on your container instances. For more information, see [Download and configure the CloudWatch agent using the command line](#) in the [Amazon CloudWatch User Guide](#). You must also add the ECS-CloudWatchLogs policy to the `ecsInstanceRole` role. For more information, see [Required permissions for monitoring container instances \(p. 641\)](#).

- 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. For more information, see [Using the awslogs log driver \(p. 161\)](#).

You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type. For more information, see [???](#) (p. 354).

- 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 events and EventBridge \(p. 558\)](#) in this guide and [What Is Amazon CloudWatch Events?](#) in the [Amazon CloudWatch Events 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 \(p. 584\)](#) 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](#).
- AWS Compute Optimizer is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads.

For more information, see [AWS Compute Optimizer recommendations \(p. 586\)](#).

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](#).

Amazon ECS collects metrics for clusters and services. You must turn on Amazon ECS CloudWatch Container Insights for per-task metrics, including CPU and memory utilization. For more information about Container Insights, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).

Topics

- [Using CloudWatch metrics \(p. 547\)](#)
- [Available metrics and dimensions \(p. 548\)](#)
- [Cluster reservation \(p. 554\)](#)
- [Cluster utilization \(p. 555\)](#)
- [Service utilization \(p. 556\)](#)
- [Service RUNNING task count \(p. 557\)](#)
- [Viewing Amazon ECS metrics \(p. 557\)](#)

Using CloudWatch metrics

Any Amazon ECS service using the Fargate launch type has CloudWatch CPU and memory utilization metrics automatically, so you don't need to take any manual steps.

For any Amazon ECS task or service using the EC2 launch type, your Amazon ECS container instances require version 1.4.0 or later (Linux) or 1.0.0 or later (Windows) of the container agent for CloudWatch metrics. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

The minimum Docker version for reliable metrics is Docker version v20.10.13 and newer, which is included in Amazon ECS-optimized AMI 20220607 and newer.

If you're starting your agent manually (for example, if you're not using the Amazon ECS-optimized AMI for your container instances), see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 368\)](#).

Your Amazon ECS container instances also require the `ecs:StartTelemetrySession` permission on the IAM role that you launch your container instances with. If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, you might need to add this permission. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see [Checking for the container instance \(ecsInstanceRole\) in the IAM console \(p. 639\)](#).

Note

You can disable CloudWatch metrics collection by setting `ECS_DISABLE_METRICS=true` in your Amazon ECS container agent configuration. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

Available metrics and dimensions

The following sections list the metrics and dimensions that Amazon ECS sends to Amazon CloudWatch.

Amazon ECS metrics

Amazon ECS provides metrics for you to monitor your resources. You can measure 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. For your GPU workloads, you can measure your GPU reservation across your cluster.

The metrics made available will depend on the launch type of the tasks and services in your clusters. If you're using the Fargate launch type for your services, CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the EC2 launch type, Amazon ECS provides CPU, memory, and GPU reservation and CPU and memory utilization metrics at the cluster and service level. You need to monitor the Amazon EC2 instances that make up your underlying infrastructure separately.

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.

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. Only container instances in ACTIVE or DRAINING status will affect CPU reservation metrics. 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 the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. Only container instances in ACTIVE or DRAINING status will affect CPU utilization metrics. 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. Only container instances in ACTIVE or DRAINING status will affect memory reservation metrics. 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.

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. Only container instances in ACTIVE or DRAINING status will affect memory utilization metrics. 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 EC2 launch types.

Valid dimensions: ClusterName, ServiceName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

GPUReservation

The percentage of total available GPUs that are reserved by running tasks in the cluster.

Cluster GPU reservation is measured as the number of GPUs reserved by Amazon ECS tasks on the cluster, divided by the total number of GPUs that was available on all of the container instances with GPUs in the cluster. Only container instances in ACTIVE or DRAINING status will affect GPU reservation metrics.

Valid dimensions: ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

ActiveConnectionCount

The total number of concurrent connections active from clients to the Amazon ECS Service Connect proxies that run in tasks that share the selected DiscoveryName.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: DiscoveryName and DiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

NewConnectionCount

The total number of new connections established from clients to the Amazon ECS Service Connect proxies that run in tasks that share the selected DiscoveryName.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: DiscoveryName and DiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

ProcessedBytes

The total number of bytes of inbound traffic processed by the Service Connect proxies.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: DiscoveryName and DiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Bytes.

RequestCount

The number of inbound traffic requests processed by the Service Connect proxies.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: DiscoveryName and DiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

GrpcRequestCount

The number of gRPC inbound traffic requests processed by the Service Connect proxies.

This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is GRPC in the port mapping in the task definition.

Valid dimensions: DiscoveryName and DiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

HTTPCode_Target_2XX_Count

The number of HTTP response codes with numbers 200 to 299 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly.

This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is HTTP or HTTP2 in the port mapping in the task definition.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

HTTPCode_Target_3XX_Count

The number of HTTP response codes with numbers 300 to 399 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly.

This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is HTTP or HTTP2 in the port mapping in the task definition.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

HTTPCode_Target_4XX_Count

The number of HTTP response codes with numbers 400 to 499 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly.

This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is HTTP or HTTP2 in the port mapping in the task definition.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

HTTPCode_Target_5XX_Count

The number of HTTP response codes with numbers 500 to 599 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly.

This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is HTTP or HTTP2 in the port mapping in the task definition.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

RequestCountPerTarget

The average number of requests received by each target that share the selected DiscoveryName.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Count.

TargetProcessedBytes

The total number of bytes processed by the Service Connect proxies.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Bytes.

TargetResponseTime

The latency of the application request processing. The time elapsed, in milliseconds, after the request reached the Service Connect proxy in the target task until a response from the target application is received back to the proxy.

This metric is only available if you have configured Amazon ECS Service Connect.

Valid dimensions: TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Milliseconds.

Note

If you're 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. For burstable performance instances (T3, T3a, and T2 instances), the CPU utilization metric may reflect different data compared to instance-level CPU metrics.

Dimensions for Amazon ECS metrics

Amazon ECS metrics use the AWS/ECS namespace and provide metrics for the following dimensions. Metrics for a dimension only reflect the resources with running tasks during a period. For example, if you have a cluster with one service in it but that service has no tasks in a RUNNING state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks would be sent.

`ClusterName`

This dimension filters the data that you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName`.

`ServiceName`

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

`DiscoveryName`

This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across all Amazon ECS clusters.

Note that a specific port in a running container can have multiple discovery names.

`DiscoveryName, ServiceName, ClusterName`

This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across tasks that have this discovery name and that are created by this service in this cluster.

Use this dimension to see the inbound traffic metrics for a specific service, if you have reused the same discovery name in multiple services in different namespaces.

Note that a specific port in a running container can have multiple discovery names.

`TargetDiscoveryName`

This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across all Amazon ECS clusters.

Different from `DiscoveryName`, these traffic metrics only measure inbound traffic to this `DiscoveryName` that come from other Amazon ECS tasks that have a Service Connect configuration in this namespace. This includes tasks made by services with either a client-only or client-server Service Connect configuration.

Note that a specific port in a running container can have multiple discovery names.

`TargetDiscoveryName, ServiceName, ClusterName`

This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name but only counts traffic from tasks created by this service in this cluster.

Use this dimension to see the inbound traffic metrics that come from a specific client in another service.

Different from `DiscoveryName`, `ServiceName`, `ClusterName`, these traffic metrics only measure inbound traffic to this `DiscoveryName` that come from other Amazon ECS tasks that have a Service Connect configuration in this namespace. This includes tasks made by services with either a client-only or client-server Service Connect configuration.

Note that a specific port in a running container can have multiple discovery names.

Cluster reservation

Cluster reservation metrics are measured as the percentage of CPU, memory, and GPUs that are reserved by all Amazon ECS tasks on a cluster when compared to the aggregate CPU, memory, and GPUs that were registered for each active container instance in the cluster. Only container instances in ACTIVE or DRAINING status will affect cluster reservation metrics. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.

$$\text{Cluster CPU reservation} = \frac{(\text{Total CPU units reserved by tasks in cluster}) \times 100}{(\text{Total CPU units registered by container instances in cluster})}$$

$$\text{Cluster memory reservation} = \frac{(\text{Total MiB of memory reserved by tasks in cluster} \times 100)}{(\text{Total MiB of memory registered by container instances in cluster})}$$

$$\text{Cluster GPU reservation} = \frac{(\text{Total GPUs reserved by tasks in cluster} \times 100)}{(\text{Total GPUs registered by container instances in cluster})}$$

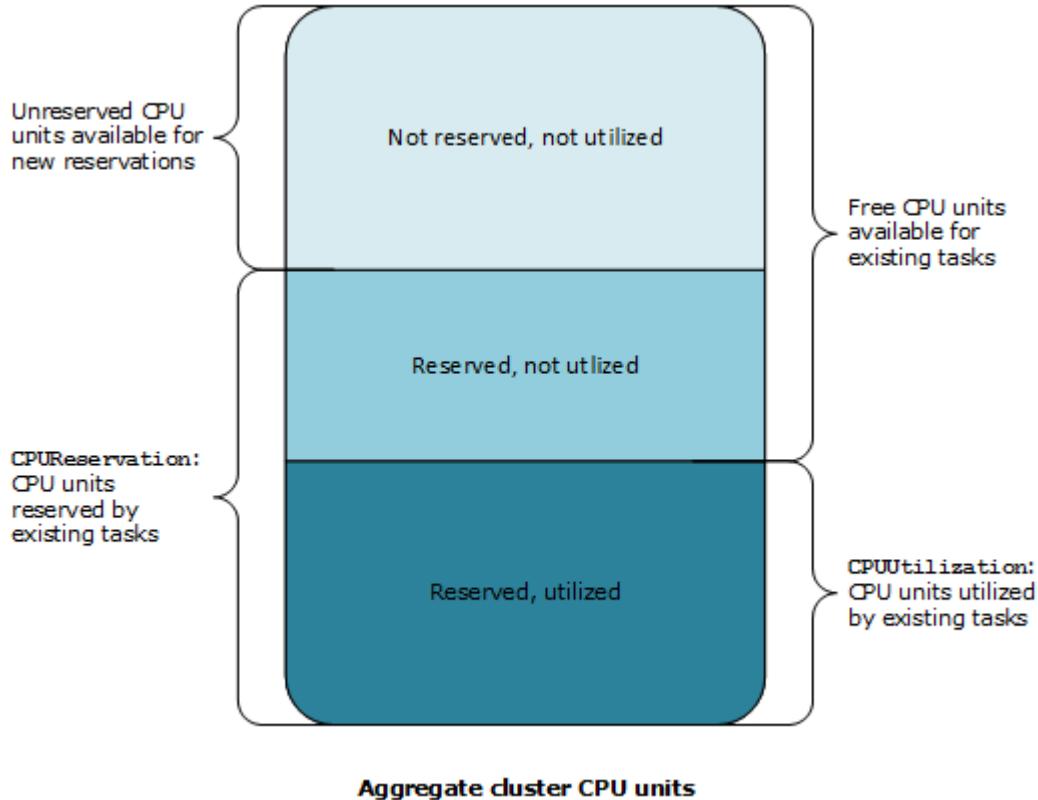
When you run a task in a cluster, Amazon ECS parses its task definition and reserves the aggregate CPU units, MiB of memory, and GPUs that are specified in its container definitions. Each minute, Amazon ECS calculates the number of CPU units, MiB of memory, and GPUs that are currently reserved for each task that is running in the cluster. The total amount of CPU, memory, and GPUs reserved for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster. If you specify a soft limit (`memoryReservation`), it's 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](#).

For example, a cluster has two active container instances registered: a `c4.4xlarge` instance and a `c4.1large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.1large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If a task definition reserves 1,024 CPU units and 2,048 MiB of memory, and ten tasks are started with this task definition on this cluster (and no other tasks are currently running), a total of 10,240 CPU units and 20,480 MiB of memory are reserved. This is reported to CloudWatch as 55% CPU reservation and 60% memory reservation for the cluster.

The following illustration shows the total registered CPU units in a cluster and what their reservation and utilization means to existing tasks and new task placement. The lower (Reserved, used) and center (Reserved, not used) blocks represent the total CPU units that are reserved for the existing tasks that are running on the cluster, or the `CPUReservation` CloudWatch metric. The lower block represents the

reserved CPU units that the running tasks are actually using on the cluster, or the `CPUUtilization` CloudWatch metric. The upper block represents CPU units that are not reserved by existing tasks; these CPU units are available for new task placement. Existing tasks can use these unreserved CPU units as well, if their need for CPU resources increases. For more information, see the [cpu \(p. 813\)](#) task definition parameter documentation.



Cluster utilization

Cluster utilization is measured as the percentage of CPU and memory that is used by all Amazon ECS tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active container instance in the cluster. Only container instances in ACTIVE or DRAINING status will affect cluster utilization metrics. A GPU utilization metric isn't supported because it's not possible to overcommit a GPU. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.

$$\text{Cluster CPU utilization} = \frac{(\text{Total CPU units used by tasks in cluster}) \times 100}{(\text{Total CPU units registered by container instances in cluster})}$$

$$\text{Cluster memory utilization} = \frac{(\text{Total MiB of memory used by tasks in cluster} \times 100)}{(\text{Total MiB of memory registered by container instances in cluster})}$$

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory

used for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster.

For example, a cluster has two active container instances registered, a c4.4xlarge instance and a c4.1.large instance. The c4.4xlarge instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The c4.1.large instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If ten tasks are running on this cluster and each task consumes 1,024 CPU units and 2,048 MiB of memory, a total of 10,240 CPU units and 20,480 MiB of memory are used on the cluster. This is reported to CloudWatch as 55% CPU utilization and 60% memory utilization for the cluster.

Service utilization

Service utilization is measured as the percentage of CPU and memory that is used by the Amazon ECS tasks that belong to a service on a cluster when compared to the CPU and memory that is specified in the service's task definition. This metric is supported for services with tasks using both the EC2 and Fargate launch types.

$$\text{Service CPU utilization} = \frac{(\text{Total CPU units used by tasks in service}) \times 100}{(\text{Total CPU units specified in task definition}) \times (\text{number of tasks in service})}$$

$$\text{Service memory utilization} = \frac{100}{(\text{Total MiB of memory used by tasks in service}) \times (\text{Total MiB of memory specified in task definition}) \times (\text{number of tasks in service})}$$

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks owned by the service that are running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total resources that are specified for the service in the service's task definition. If you specify a soft limit (`memoryReservation`), it's 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](#).

For example, the task definition for a service specifies a total of 512 CPU units and 1,024 MiB of memory (with the hard limit `memory` parameter) for all of its containers. The service has a desired count of 1 running task, the service is running on a cluster with 1 c4.1.large container instance (with 2,048 CPU units and 3,768 MiB of total memory), and there are no other tasks running on the cluster. Although the task specifies 512 CPU units, because it is the only running task on a container instance with 2,048 CPU units, it can use up to four times the specified amount (2,048 / 512). However, the specified memory of 1,024 MiB is a hard limit and it can't be exceeded, so in this case, service memory utilization can't exceed 100%.

If the previous example used the soft limit `memoryReservation` instead of the hard limit `memory` parameter, the service's tasks could use more than the specified 1,024 MiB of memory as needed. In this case, the service's memory utilization could exceed 100%.

If your application has a sudden spike in memory utilization for a short amount of time, you will not see the service memory utilization increasing because Amazon ECS collects multiple data points every minute, and then aggregates them to one data point that is sent to CloudWatch.

If this task is performing CPU-intensive work during a period and using all 2,048 of the available CPU units and 512 MiB of memory, the service reports 400% CPU utilization and 50% memory utilization. If the task is idle and using 128 CPU units and 128 MiB of memory, the service reports 25% CPU utilization and 12.5% memory utilization.

Note

In this example, the CPU utilization will only go above 100% when the CPU units are defined at the container level. If you define CPU units at the task level, the utilization will not go above the defined task-level limit.

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.

Service RUNNING task count in Amazon ECS CloudWatch Container Insights

A "Number of Running Tasks" (RunningTaskCount) metric is available per cluster and per service when you use Amazon ECS CloudWatch Container Insights. You can use Container Insights for all new clusters created by opting in to the `containerInsights` account setting, on individual clusters by turning on the cluster settings during cluster creation, or on existing clusters by using the `UpdateClusterSettings` API. Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [CloudWatch Pricing](#).

To view this metric, see [Amazon ECS Container Insights Metrics](#) in the *Amazon CloudWatch User Guide*.

Service RUNNING task count from Amazon ECS provided metrics

However Amazon ECS provides monitoring metrics at no additional cost. To use these metrics to count the running tasks, follow the steps below in the CloudWatch console.

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, All metrics**.
3. On the **Browse** tab, choose the **ECS** namespace.
4. Choose **ClusterName, ServiceName** and then choose any metric (either **CPUUtilization** or **MemoryUtilization**) that corresponds to the service to view running tasks in.
5. On the **Graphed metrics** tab, change **Period** to **1 Minute** and **Statistic** to **Sample Count**.

The value displayed in the graph indicates the number of RUNNING tasks in the service.

Viewing Amazon ECS metrics

After you have turned on CloudWatch metrics for Amazon ECS, you can view those metrics on the Amazon ECS and CloudWatch consoles. The Amazon ECS console provides a 24-hour maximum, minimum, and average view of your cluster and service metrics. The CloudWatch console provides a fine-grained and customizable display of your resources, as well as the number of running tasks in a service.

Topics

- [Viewing cluster metrics using the Amazon ECS console \(p. 558\)](#)

- [Viewing service metrics using the Amazon ECS console \(p. 558\)](#)
- [Viewing Amazon ECS metrics using the CloudWatch console \(p. 558\)](#)

Viewing cluster metrics using the Amazon ECS console

Cluster and service metrics are available on the Amazon ECS console. The view provided for cluster metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information about cluster metrics, see [Cluster reservation \(p. 554\)](#) and [Cluster utilization \(p. 555\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Select the cluster that you want to view metrics for.
3. On the Cluster: **cluster-name** page, choose Metrics.

Viewing service metrics using the Amazon ECS console

Amazon ECS service CPU and memory utilization metrics are available on 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 \(p. 556\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. Select the cluster that you want to view metrics for.
3. On the Cluster: **cluster-name** page, select the service.

The metrics are available under **Health and metrics**.

Viewing Amazon ECS metrics using the CloudWatch console

Amazon ECS cluster and service metrics can also be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs. You can view [Cluster reservation \(p. 554\)](#), [Cluster utilization \(p. 555\)](#), [Service utilization \(p. 556\)](#), and the [Service RUNNING task count \(p. 557\)](#). For information about how to view the metrics, see [View available metrics](#) the *Amazon CloudWatch User Guide*.

Amazon ECS events and EventBridge

Using Amazon EventBridge, you can automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule. The actions that can be automatically configured to include the following:

- Adding events to log groups in CloudWatch Logs
- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon Simple Queue Service (Amazon SQS) queue

For more information, see [Getting Started with Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the Fargate launch type, you can see the state of your tasks. If your tasks are using the EC2 launch type, you can see the state of both the container instances and the current state of all tasks running on those container instances. For services, you can see events related to the health of your service.

Using EventBridge, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters and monitoring 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 EventBridge target. Targets might include AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event is delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see [Handling events \(p. 571\)](#).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics

- [Amazon ECS events \(p. 559\)](#)
- [Handling events \(p. 571\)](#)

Amazon ECS events

Amazon ECS sends the following types of events to EventBridge: container instance state change events, task state change events, service action, and service deployment state change events. If these resources change, an event is generated. 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 deserializing event JSON data in code, make sure that your application is prepared to handle unknown properties to avoid issues if and when these additional properties are added.

In some cases, multiple events are generated for the same activity. For example, when a task is started on a container instance, a task state change event is generated for the new task. A container instance state change event is generated to account for the change in available resources, such as CPU, memory, and available ports, on the container instance. Likewise, if a container instance is terminated, events are generated for the container instance, the container agent connection status, and every task that was running on the container instance.

Container state change and task state change events contain two version fields: one in the main body of the event, and one in the detail object of the event. The following describes the differences between these two fields:

- The version field in the main body of the event is set to 0 on all events. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.
- The version field 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. Duplicate events have the same version in the detail object. If you are replicating your Amazon ECS container instance and task state with

EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in EventBridge for the resource (inside the `detail` object) to verify that the version in your event stream is current.

Service action events only contain the `version` field in the main body.

Examples are covered later in this topic. For additional information about how to integrate Amazon ECS and EventBridge, see [Integrating Amazon EventBridge and Amazon ECS](#).

Container instance state change events

The following scenarios cause container instance state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations, either directly or with the AWS Management Console or SDKs.

Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS service scheduler starts or stops a task.

Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation with a `STOPPED` status for a task with a desired status of `RUNNING`.

The Amazon ECS container agent monitors the state of tasks on your container instances, and it reports any state changes. If a task that is supposed to be `RUNNING` is transitioned to `STOPPED`, the agent releases the resources that were allocated to the stopped task, such as CPU, memory, and available ports.

You deregister the container instance with the `DeregisterContainerInstance` API operation, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent.

A task was stopped when an EC2 instance was stopped.

When you stop a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

The Amazon ECS container agent registers a container instance for the first time.

The first time the Amazon ECS container agent registers a container instance (at launch or when first run manually), this creates a state change event for the instance.

The Amazon ECS container agent connects or disconnects from Amazon ECS.

When the Amazon ECS container agent connects or disconnects from the Amazon ECS backend, it changes the `agentConnected` status of the container instance.

Note

The Amazon ECS container agent disconnects and reconnects several times per hour as a part of its normal operation, so agent connection events should be expected. These events are not an indication that there is an issue with the container agent or your container instance.

You upgrade the Amazon ECS container agent on an instance.

The container instance detail contains an object for the container agent version. If you upgrade the agent, this version information changes and generates an event.

Example Container instance state change event

Container instance state change events are delivered in the following format. The detail section below resembles the [ContainerInstance](#) object that is returned from a [DescribeContainerInstances](#) API operation in the *Amazon Elastic Container Service API Reference*. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

```
{  
    "version": "0",  
    "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",  
    "detail-type": "ECS Container Instance 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:container-instance/  
b54a2a04-046f-4331-9d74-3f6d7f6ca315"  
    ],  
    "detail": {  
        "agentConnected": true,  
        "attributes": [  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.syslog"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.json-file"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.privileged-container"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.ecr-auth"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.task-iam-role"  
            }  
        ]  
    }  
}
```

```

"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",
"ec2InstanceId": "i-f3a8506b",
"registeredResources": [
  {
    "name": "CPU",
    "type": "INTEGER",
    "integerValue": 2048
  },
  {
    "name": "MEMORY",
    "type": "INTEGER",
    "integerValue": 3767
  },
  {
    "name": "PORTS",
    "type": "STRINGSET",
    "stringSetValue": [
      "22",
      "2376",
      "2375",
      "51678",
      "51679"
    ]
  },
  {
    "name": "PORTS_UDP",
    "type": "STRINGSET",
    "stringSetValue": []
  }
],
"remainingResources": [
  {
    "name": "CPU",
    "type": "INTEGER",
    "integerValue": 1988
  },
  {
    "name": "MEMORY",
    "type": "INTEGER",
    "integerValue": 767
  },
  {
    "name": "PORTS",
    "type": "STRINGSET",
    "stringSetValue": [
      "22",
      "2376",
      "2375",
      "51678",
      "51679"
    ]
  },
  {
    "name": "PORTS_UDP",
    "type": "STRINGSET",
    "stringSetValue": []
  }
],
"status": "ACTIVE",
"version": 14801,
"versionInfo": {
  "agentHash": "aebcbca",
  "agentVersion": "1.13.0",
  "dockerVersion": "DockerVersion: 1.11.2"
}

```

```
    },
    "updatedAt": "2016-12-06T16:41:06.991Z"
}
```

Task state change events

The following scenarios cause 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 tasks on your container instances, and it reports any state changes. State changes might include changes from `PENDING` to `RUNNING` or from `RUNNING` to `STOPPED`.

You force deregistration of the underlying container instance with the `DeregisterContainerInstance` API operation and the `force` flag, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent. If tasks are running on the container instance, the `force` flag must be set to allow deregistration. This stops all tasks on the instance.

The underlying container instance is stopped or terminated.

When you stop or terminate a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

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 generates an event.

A task using the Fargate Spot capacity provider receives a termination notice.

When a task is using the `FARGATE_SPOT` capacity provider and is stopped due to a Spot interruption, a task state change event is generated.

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*. If your containers are using an image hosted with Amazon ECR, the `imageDigest` field is returned.

Note

The values for the `createdAt`, `connectivityAt`, `pullStartedAt`, `startedAt`, `pullStoppedAt`, and `updatedAt` fields are UNIX timestamps in the response of a `DescribeTasks` action whereas in the task state change event they are ISO string timestamps.

For more information about CloudWatch Events parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

```
{
```

```

    "version": "0",
    "id": "3317b2af-7005-947d-b652-f55e762e571a",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-01-23T17:57:58Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/
c13b4cb40f1f4fe4a2971f76ae5a47ad"
    ],
    "detail": {
        "attachments": [
            {
                "id": "1789bcae-ddfb-4d10-8ebe-8ac87ddba5b8",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-abcd1234"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-abcd1234"
                    },
                    {
                        "name": "macAddress",
                        "value": "0a:98:eb:a7:29:ba"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.0.139"
                    }
                ]
            }
        ],
        "availabilityZone": "us-west-2c",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/FargateCluster",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-west-2:111122223333:container/
cf159fd6-3e3f-4a9e-84f9-66cbe726af01",
                "lastStatus": "RUNNING",
                "name": "FargateApp",
                "image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/hello-
repository:latest",
                "imageDigest": "sha256:74b2c688c700ec95a93e478cdb959737c148df3fbf5ea706abe0318726e885e6",
                "runtimeId": "ad64cbc71c7fb31c55507ec24c9f77947132b03d48d9961115cf24f3b7307e1e",
                "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/
c13b4cb40f1f4fe4a2971f76ae5a47ad",
                "networkInterfaces": [
                    {
                        "attachmentId": "1789bcae-ddfb-4d10-8ebe-8ac87ddba5b8",
                        "privateIpv4Address": "10.0.0.139"
                    }
                ],
                "cpu": "0"
            }
        ],
        "createdAt": "2020-01-23T17:57:34.402Z",
        "launchType": "FARGATE",
        "cpu": "256",
        "memory": "512",
    }
}

```

```

    "desiredStatus": "RUNNING",
    "group": "family:sample-fargate",
    "lastStatus": "RUNNING",
    "overrides": [
        "containerOverrides": [
            {
                "name": "FargateApp"
            }
        ]
    ],
    "connectivity": "CONNECTED",
    "connectivityAt": "2020-01-23T17:57:38.453Z",
    "pullStartedAt": "2020-01-23T17:57:52.103Z",
    "startedAt": "2020-01-23T17:57:58.103Z",
    "pullStoppedAt": "2020-01-23T17:57:55.103Z",
    "updatedAt": "2020-01-23T17:57:58.103Z",
    "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/c13b4cb40f1f4fe4a2971f76ae5a47ad",
    "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/sample-fargate:1",
    "version": 4,
    "platformVersion": "1.3.0"
}
}

```

Service action events

Amazon ECS sends service action events with the detail type **ECS Service Action**. Unlike the container instance and task state change events, the service action events do not include a version number in the details response field. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service action events. For more information, see [Creating an EventBridge Rule](#) in the [Amazon EventBridge User Guide](#).

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Service Action"
    ]
}
```

Amazon ECS sends events with INFO, WARN, and ERROR event types. The following are the service action events.

Service action events with INFO event type

SERVICE_STEADY_STATE

The service is healthy and at the desired number of tasks, thus reaching a steady state. The service scheduler reports the status periodically, so you might receive this message multiple times.

TASKSET_STEADY_STATE

The task set is healthy and at the desired number of tasks, thus reaching a steady state.

CAPACITY_PROVIDER_STEADY_STATE

A capacity provider associated with a service reaches a steady state.

SERVICE_DESIRED_COUNT_UPDATED

When the service scheduler updates the computed desired count for a service or task set. This event is not sent when the desired count is manually updated by a user.

Service action events with WARN event type

SERVICE_TASK_START_IMPAIRED

The service is unable to consistently start tasks successfully.

SERVICE_DISCOVERY_INSTANCE_UNHEALTHY

A service using service discovery contains an unhealthy task. The service scheduler detects that a task within a service registry is unhealthy.

Service action events with ERROR event type

SERVICE_DAEMON_PLACEMENT_CONSTRAINT_VIOLATED

A task in a service using the DAEMON service scheduler strategy no longer meets the placement constraint strategy for the service.

ECS_OPERATION_THROTTLED

The service scheduler has been throttled due to the Amazon ECS API throttle limits.

SERVICE_DISCOVERY_OPERATION_THROTTLED

The service scheduler has been throttled due to the AWS Cloud Map API throttle limits. This can occur on services configured to use service discovery.

SERVICE_TASK_PLACEMENT_FAILURE

The service scheduler is unable to place a task. The cause will be described in the `reason` field.

A common cause for this service event being generated is because of a lack of resources in the cluster to place the task. For example, not enough CPU or memory capacity on the available container instances or no container instances being available. Another common cause is when the Amazon ECS container agent is disconnected on the container instance, causing the scheduler to be unable to place the task.

SERVICE_TASK_CONFIGURATION_FAILURE

The service scheduler is unable to place a task due to a configuration error. The cause will be described in the `reason` field.

A common cause of this service event being generated is because tags were being applied to the service but the user or role had not opted in to the new Amazon Resource Name (ARN) format in the Region. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 246\)](#). Another common cause is that Amazon ECS was unable to assume the task IAM role provided.

Example Service steady state event

Service steady state events are delivered in the following format. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

```
{  
    "version": "0",  
    "id": "af3c496d-f4a8-65d1-70f4-a69d52e9b584",  
    "detail-type": "ECS Service Action",  
    "source": "aws.ecs",  
    "account": "111122223333",  
    "time": "2019-11-19T19:27:22Z",  
    "region": "us-west-2",  
    "resources": [  
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"  
    ],  
}
```

```
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:27:22.695Z"
    }
}
```

Example Capacity provider steady state event

Capacity provider steady state events are delivered in the following format.

```
{
    "version": "0",
    "id": "b9baa007-2f33-0eb1-5760-0d02a572d81f",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:37:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "CAPACITY_PROVIDER_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/ASG-tutorial-capacity-provider"
        ],
        "createdAt": "2019-11-19T19:37:00.807Z"
    }
}
```

Example Service task start impaired event

Service task start impaired events are delivered in the following format.

```
{
    "version": "0",
    "id": "57c9506e-9d21-294c-d2fe-e8738da7e67d",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "WARN",
        "eventName": "SERVICE_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:55:38.725Z"
    }
}
```

Example Service task placement failure event

Service task placement failure events are delivered in the following format. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

In the following example, the task was attempting to use the FARGATE_SPOT capacity provider but the service scheduler was unable to acquire any Fargate Spot capacity.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/FARGATE_SPOT"
        ],
        "reason": "RESOURCE:FARGATE",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

In the following example for the EC2 launch type, the task was attempted to launch on the Container Instance 2dd1b186f39845a584488d2ef155c131 but the service scheduler was unable to place the task because of insufficient CPU.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "containerInstanceArns": [
            "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
        ],
        "reason": "RESOURCE:CPU",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

Service deployment state change events

Amazon ECS sends service deployment change state events with the detail type **ECS Deployment State Change**. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service deployment state change events. For more information, see [Creating an EventBridge Rule](#) in the *Amazon EventBridge User Guide*.

```
{
```

```

    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Deployment State Change"
    ]
}

```

Amazon ECS sends events with INFO and ERROR event types. The following are the service deployment state change events.

SERVICE_DEPLOYMENT_IN_PROGRESS

The service deployment is in progress. This event is sent for both initial deployments and rollback deployments.

SERVICE_DEPLOYMENT_COMPLETED

The service deployment has completed. This event is sent once a service reaches a steady state after a deployment.

SERVICE_DEPLOYMENT_FAILED

The service deployment has failed. This event is sent for services with deployment circuit breaker logic turned on.

Example service deployment in progress event

Service deployment in progress events are delivered when both an initial and a rollback deployment is started. The difference between the two is in the `reason` field. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

The following shows an example output for an initial deployment starting.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentId in progress."
    }
}
```

The following shows an example output for a rollback deployment starting. The `reason` field provides the ID of the deployment the service is rolling back to.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
```

```

"account": "111122223333",
"time": "2020-05-23T12:31:14Z",
"region": "us-west-2",
"resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
],
"detail": {
    "eventType": "INFO",
    "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
    "deploymentId": "ecs-svc/123",
    "updatedAt": "2020-05-23T11:11:11Z",
    "reason": "ECS deployment circuit breaker: rolling back to
deploymentId deploymentID."
}
}

```

Example service deployment completed event

Service deployment completed state events are delivered in the following format. For more information, see [Rolling update \(p. 473\)](#).

```

{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_COMPLETED",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentID completed."
    }
}

```

Example service deployment failed event

Service deployment failed state events are delivered in the following format. A service deployment failed state event will only be sent for services that have deployment circuit breaker logic turned on. For more information, see [Rolling update \(p. 473\)](#).

```

{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_DEPLOYMENT_FAILED",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentID failed due to
an error: errorReason."
    }
}

```

```
        "reason": "ECS deployment circuit breaker: task failed to start."
    }
}
```

Handling events

Amazon ECS sends events on an *at least once* basis. This means you may receive multiple copies of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred.

To order of events properly, the detail section of each event contains a `version` property. Each time a resource changes state, this version is incremented. Duplicate events have the same version in the detail object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in EventBridge for the resource to verify that the version in your event stream is current. Events with a higher version property number should be treated as occurring later than events with lower version numbers.

Example: Handling events in an AWS Lambda function

The following example shows a Lambda function written in Python 2.7 that captures both task and container instance state change events and saves them to one of two Amazon DynamoDB tables:

- `ECSCtrInstanceState` – Stores the latest state for a container instance. The table ID is the `containerInstanceArn` value of the container instance.
- `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
)
```

Amazon ECS CloudWatch Container Insights

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices.

Operational data is collected as performance log events. These are entries that use a structured JSON schema for high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates higher-level aggregated metrics at the cluster, service, and task level as CloudWatch metrics. The metrics include utilization for resources such as CPU, memory, disk, and network. The metrics are available in CloudWatch automatic dashboards. For information about the available metrics, see [Amazon ECS Container Insights metrics](#) in the [Amazon CloudWatch User Guide](#).

Important

Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [CloudWatch Pricing](#). Amazon ECS also provides monitoring metrics that are provided at no additional cost. For more information, see [Amazon ECS CloudWatch metrics \(p. 547\)](#).

Container Insights considerations

The following should be considered when using CloudWatch Container Insights.

- CloudWatch Container Insights metrics only reflect the resources with running tasks during the specified time range. For example, if you have a cluster with one service in it but that service has no

tasks in a RUNNING state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks will be sent.

- Network metrics are available for all tasks run on Fargate and tasks run on Amazon EC2 instances that use either the bridge or awsvpc network modes.

Setting up CloudWatch Container Insights for cluster and service level metrics

Container Insights can be turned on for all new clusters created by opting in to the containerInsights account setting, on individual clusters by turning it on during cluster creation, or on existing clusters by using the `UpdateClusterSettings` API.

Opting in to the containerInsights account setting can be done with both the Amazon ECS console and the AWS CLI. You must be running version 1.16.200 or later of the AWS CLI to use this feature. For more information on creating Amazon ECS clusters, see [Creating a cluster using the classic console \(p. 879\)](#).

Important

For clusters containing tasks or services using the EC2 launch type, your container instances must be running version 1.29.0 or later of the Amazon ECS agent. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).

To change the default for Container Insights for all users using the console

You can make all new clusters turn on Container Insights when they are created by all users and roles. These changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves. Any user on an account can use one of the following steps to modify the default account setting for all users or roles on your account. The following steps show how to set this default using the AWS Management Console.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. Under **CloudWatch Container Insights**, select **CloudWatch Container Insights**.

Important

You must give users the `ecs:PutAccountSetting` permission to perform this action.

6. Choose **Save changes**.
7. On the confirmation screen, choose **Confirm** to save the selection.

To change the default for Container Insights for all users using the command line

You can make all new clusters turn on Container Insights when they are created by all IAM users and roles. These changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves. Any user on an account can use one of the following steps to modify the default account setting for all users or roles on your account. The following steps show how to set this default using the AWS Command Line Interface.

1. [put-account-setting-default](#) (AWS CLI)

```
aws ecs put-account-setting-default --name containerInsights --value enabled --region us-east-1
```

2. [Write-ECSAccountSettingDefault](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name containerInsights -Value enabled -Region us-east-1 -Force
```

To change the default for Container Insights for a specific user using the command line

You can make all new clusters turn on Container Insights when they are created by a specific user or role. This is useful when a specific role is used by AWS CloudFormation to make all changes in a production account, for example. The account owner can use one of the following commands and specify the ARN of the principal user or container instance IAM role in the request to modify the account settings.

1. [put-account-setting](#) (AWS CLI)

The following example is for modifying the account setting of a specific user:

```
aws ecs put-account-setting --name containerInsights --value enabled --principal-arn arn:aws:iam::aws_account_id:user/userName --region us-east-1
```

2. [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

The following example is for modifying the account setting of a specific user:

```
Write-ECSAccountSetting -Name containerInsights -Value enabled -PrincipalArn arn:aws:iam::aws_account_id:user/userName -Region us-east-1 -Force
```

To turn on Container Insights for a specific cluster using the command line

Use one of the following commands to turn on Container Insights for a cluster.

- [update-cluster-settings](#) (AWS CLI)

```
aws ecs update-cluster-settings --cluster cluster_name_or_arn --settings name=containerInsights,value=enabled/disabled --region us-east-1
```

Use CloudWatch Container Insights to view Amazon ECS lifecycle events

You can view Amazon ECS task and service lifecycle events within the CloudWatch Container Insights console. This helps you correlate your container metrics, logs, and events in a single view to give you a more complete operational visibility.

The events that you can view are the ones that Amazon ECS sends to Amazon EventBridge. For more information, see [Amazon ECS events](#).

You can choose to configure performance metrics for clusters, tasks, or services. Depending on the resource you choose, the following events are reported:

- Container instance state change events
- Service action events
- Task state change events

You must configure the correct permissions, and then you can configure and view the events in the CloudWatch Container Insights console. For more information, see [Amazon ECS lifecycle events within Container Insights](#) in the *Amazon CloudWatch User Guide*.

Permissions required to configure Container Insights to view Amazon ECS lifecycle events

The following permissions are required to configure the lifecycle events:

- events:PutRule
- events:PutTargets
- logs>CreateLogGroup

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "events:PutRule",  
                "events:PutTargets",  
                "logs>CreateLogGroup"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Permissions required to view Amazon ECS lifecycle events in Container Insights

The following permissions are required to view the lifecycle events. Add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- events:DescribeRule
- events>ListTargetsByRule
- logs:DescribeLogGroups

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "events:DescribeRule",  
                "events>ListTargetsByRule",  
                "logs:DescribeLogGroups"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "events:DescribeRule",
            "events>ListTargetsByRule",
            "logs:DescribeLogGroups"
        ],
        "Resource": "*"
    }
}
```

Container instance health

Amazon ECS provides container instance health monitoring. You can quickly determine whether Amazon ECS has detected any problems that might prevent your container instances from running containers. Amazon ECS performs automated checks on every running container instance with agent version 1.57.0 or later to identify issues. For more information on verifying the agent version on a container instance, see [Updating the Amazon ECS container agent \(p. 364\)](#).

You must be using AWS CLI version 1.22.3 or later or AWS CLI version 2.3.6 or later. For information about how to update the AWS CLI, see [Installing or updating the latest version of the AWS CLI](#) in the [AWS Command Line Interface User Guide Version 2](#).

Status checks are performed about twice per minute, returning a pass or a fail status. If all checks pass, the overall status of the instance is OK. If one or more checks fail, the overall status is IMPAIRED. Status checks are built into Amazon ECS container agent, so they cannot be turned off or deleted. You can view the results of these status checks to identify specific and detectable problems. For more information, see [the section called "Health check" \(p. 811\)](#).

The container instance health status can be retrieved using the `DescribeContainerInstances` API. The following AWS CLI command retrieves the container instance health status.

```
aws ecs describe-container-instances \
--cluster cluster_name \
--container-instances 47279cd2cadb41cbaef2dcEXAMPLE \
--include CONTAINER_INSTANCE_HEALTH
```

The following is an example of the health status object in the output.

```
"healthStatus": {
    "overallStatus": "OK",
    "details": [
        {
            "type": "CONTAINER_RUNTIME",
            "status": "OK",
            "lastUpdated": "2021-11-10T03:30:26+00:00",
            "lastStatusChange": "2021-11-10T03:26:41+00:00"
        }
    ]
}
```

Collecting application trace data

Amazon ECS integrates with AWS Distro for OpenTelemetry to collect trace data from your application. Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route trace data to

AWS X-Ray. For more information, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon ECS](#).

For the AWS Distro for OpenTelemetry Collector to send trace data to AWS X-Ray, your application must be configured to create the trace data. For more information, see [Instrumenting your application for AWS X-Ray](#) in the *AWS X-Ray Developer Guide*.

Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task IAM role and specify the role in your task definition. We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually. For more information about creating a task execution IAM role, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Important

If you're also collecting application metrics using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application metrics \(p. 579\)](#).

To create a task IAM role for AWS Distro for OpenTelemetry integration

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**, **Create policy**.
3. On the **Create policy** page, switch to the **JSON** tab, copy and paste the following IAM policy JSON into the field, then choose **Next: Tags**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "xray:PutTraceSegments",  
                "xray:PutTelemetryRecords",  
                "xray:GetSamplingRules",  
                "xray:GetSamplingTargets",  
                "xray:GetSamplingStatisticSummaries"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

4. (Optional) Add one or more tags to the policy, then choose **Next: Review**.
5. For **Name**, specify **AWSDistroOpenTelemetryPolicyForXray**.
6. For **Description**, specify an optional description, then choose **Create policy**.
7. In the navigation pane, choose **Roles**, **Create role**.
8. In the **Select type of trusted entity** section, choose **AWS service**, **Elastic Container Service**.
9. For **Select your use case**, choose **Elastic Container Service Task**, then choose **Next: Permissions**.
10. In the **Attach permissions policy** section, search for **AWSDistroOpenTelemetryPolicyForXray**, select the policy, and then choose **Next: Tags**.

11. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
12. For **Role name**, specify `AmazonECS_OpenTelemetryXrayRole` and choose **Create role**.

Specifying the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use trace collection** option. For more information, see [Creating a task definition using the console \(p. 125\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition. The following task definition snippet shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration.

```
{  
  "family": "otel-using-xray",  
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS\_OpenTelemetryXrayRole",  
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",  
  "containerDefinitions": [  
    {  
      "name": "aws-otel-emitter",  
      "image": "application-image",  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-create-group": "true",  
          "awslogs-group": "/ecs/aws-otel-emitter",  
          "awslogs-region": "us-east-1",  
          "awslogs-stream-prefix": "ecs"  
        }  
      },  
      "dependsOn": [  
        {  
          "containerName": "aws-otel-collector",  
          "condition": "START"  
        }  
      ],  
      "name": "aws-otel-collector",  
      "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",  
      "essential": true,  
      "command": [  
        "--config=/etc/ecs/otel-instance-metrics-config.yaml"  
      ],  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-create-group": "True",  
          "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",  
          "awslogs-region": "us-east-1",  
          "awslogs-stream-prefix": "ecs"  
        }  
      }  
    },  
    {  
      "networkMode": "awsvpc",  
      "requiresCompatibilities": [  
        "FARGATE"  
      ],  
      "cpu": "1024",  
      "memory": "3072"  
    }  
  ]  
}
```

}

Collecting application metrics

Amazon ECS on Fargate supports collecting metrics from your applications running on Fargate and exporting them to either Amazon CloudWatch or Amazon Managed Service for Prometheus. Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route your application metrics to the destination. The new Amazon ECS console experience simplifies the process of adding this integration when creating your task definitions.

Topics

- [Exporting application metrics to Amazon CloudWatch \(p. 579\)](#)
- [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 582\)](#)

Exporting application metrics to Amazon CloudWatch

Amazon ECS on Fargate supports exporting your custom application metrics to Amazon CloudWatch as custom metrics. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The new Amazon ECS console experience simplifies this process by adding the [Use metric collection](#) option when creating a new task definition. For more information, see [Creating a task definition using the console \(p. 125\)](#).

The application metrics are exported to CloudWatch Logs with log group name /aws/ecs/application/metrics and the metrics can be viewed in the ECS/AWSOTel/Application namespace. Your application must be instrumented with the OpenTelemetry SDK. For more information, see [Introduction to AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon CloudWatch.

- This integration only sends your custom application metrics to CloudWatch. If you want task-level metrics, you can turn on Container Insights in the Amazon ECS cluster configuration. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
- The AWS Distro for OpenTelemetry integration is supported for Amazon ECS workloads hosted on Fargate and Amazon ECS workloads hosted on Amazon EC2 instances. External instances aren't currently supported.
- CloudWatch supports a maximum of 30 dimensions per metric. By default, Amazon ECS defaults to including the TaskARN, ClusterARN, LaunchType, TaskDefinitionFamily, and TaskDefinitionRevision dimensions to the metrics. The remaining 25 dimensions can be defined by your application. If more than 30 dimensions are configured, CloudWatch can't display them. When this occurs, the application metrics will appear in the ECS/AWSOTel/Application CloudWatch metric namespace but without any dimensions. You can instrument your application to add additional dimensions. For more information, see [Using CloudWatch metrics with AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task IAM role and specify the role in your task definition. We recommend that the AWS Distro for OpenTelemetry

sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually and added to your task definition. For more information about the task execution IAM role, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Important

If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application trace data \(p. 576\)](#).

To create a task IAM role for AWS Distro for OpenTelemetry integration with CloudWatch

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**, **Create policy**.
3. On the **Create policy** page, switch to the **JSON** tab, copy and paste the following IAM policy JSON into the field, then choose **Next: Tags**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:PutLogEvents",  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs:DescribeLogStreams",  
                "logs:DescribeLogGroups",  
                "cloudwatch:PutMetricData"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Note

If your application requires any additional permissions, you should add them to this policy. Each task definition may only specify one task IAM role. For example, if you are using a custom configuration file stored in Systems Manager, you should add the `ssm:GetParameters` permission to this IAM policy.

4. (Optional) Add one or more tags to the policy, then choose **Next: Review**.
5. For **Name**, specify `AWS-Distro-OpenTelemetry-Policy-For-CloudWatch`.
6. For **Description**, specify an optional description, then choose **Create policy**.
7. In the navigation pane, choose **Roles**, **Create role**.
8. In the **Select type of trusted entity** section, choose **AWS service**, **Elastic Container Service**.
9. For **Select your use case**, choose **Elastic Container Service Task**, then choose **Next: Permissions**.
10. In the **Attach permissions policy** section, search for `AWS-Distro-OpenTelemetry-Policy-For-CloudWatch`, select the policy, and then choose **Next: Tags**.
11. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
12. For **Role name**, specify `Amazon-ECS_OpenTelemetry-CloudWatch-Role` and choose **Create role**.

Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see [Creating a task definition using the console \(p. 125\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon CloudWatch integration.

```
{  
  "family": "otel-using-cloudwatch",  
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",  
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",  
  "containerDefinitions": [  
    {  
      "name": "aws-otel-emitter",  
      "image": "application-image",  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-create-group": "true",  
          "awslogs-group": "/ecs/aws-otel-emitter",  
          "awslogs-region": "us-east-1",  
          "awslogs-stream-prefix": "ecs"  
        }  
      },  
      "dependsOn": [{  
        "containerName": "aws-otel-collector",  
        "condition": "START"  
      }]  
    },  
    {  
      "name": "aws-otel-collector",  
      "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",  
      "essential": true,  
      "command": [  
        "--config=/etc/ecs/ecs-cloudwatch.yaml"  
      ],  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-create-group": "True",  
          "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",  
          "awslogs-region": "us-east-1",  
          "awslogs-stream-prefix": "ecs"  
        }  
      }  
    },  
    {  
      "networkMode": "awsvpc",  
      "requiresCompatibilities": [  
        "FARGATE"  
      ],  
      "cpu": "1024",  
      "memory": "3072"  
    }  
  ]  
}
```

Exporting application metrics to Amazon Managed Service for Prometheus

Amazon ECS supports exporting your task-level CPU, memory, network, and storage metrics and your custom application metrics to Amazon Managed Service for Prometheus. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The new Amazon ECS console experience simplifies this process by adding the **Use metric collection** option when creating a new task definition. For more information, see [Creating a task definition using the console \(p. 125\)](#).

The metrics are exported to Amazon Managed Service for Prometheus and can be viewed using the Amazon Managed Grafana dashboard. Your application must be instrumented with either Prometheus libraries or with the OpenTelemetry SDK. For more information about instrumenting your application with the OpenTelemetry SDK, see [Introduction to AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

When using the Prometheus libraries, your application must expose a `/metrics` endpoint which is used to scrape the metrics data. For more information about instrumenting your application with Prometheus libraries, see [Prometheus client libraries](#) in the Prometheus documentation.

Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon Managed Service for Prometheus.

- The AWS Distro for OpenTelemetry integration is supported for Amazon ECS workloads hosted on Fargate and Amazon ECS workloads hosted on Amazon EC2 instances. External instances aren't supported currently.
- By default, AWS Distro for OpenTelemetry includes all available task-level dimensions for your application metrics when exporting to Amazon Managed Service for Prometheus. You can also instrument your application to add additional dimensions. For more information, see [Getting Started with Prometheus Remote Write Exporter for Amazon Managed Service for Prometheus](#) in the AWS Distro for OpenTelemetry documentation.

Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus

The Amazon ECS integration with Amazon Managed Service for Prometheus using the AWS Distro for OpenTelemetry sidecar requires that you create a task IAM role and specify the role in your task definition. This task IAM role must be created manually using the steps below prior to registering your task definition.

We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually. For more information about creating a task execution IAM role, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Important

If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application trace data \(p. 576\)](#).

To create a task IAM role for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus

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 **AWS service**, **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 the **AmazonPrometheusRemoteWriteAccess** policy, select the policy, and then choose **Next: Tags**.
6. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
7. For **Role name**, specify **AmazonECS_OpenTelemetryPrometheusRole** and choose **Create role**.

Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see [Creating a task definition using the console \(p. 125\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon Managed Service for Prometheus integration.

```
{  
  "family": "otel-using-cloudwatch",  
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS\_OpenTelemetryCloudWatchRole",  
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",  
  "containerDefinitions": [  
    {  
      "name": "aws-otel-emitter",  
      "image": "application-image",  
      "logConfiguration": {  
        "logDriver": "awslogs",  
        "options": {  
          "awslogs-create-group": "true",  
          "awslogs-group": "/ecs/aws-otel-emitter",  
          "awslogs-region": "aws-region",  
          "awslogs-stream-prefix": "ecs"  
        }  
      },  
      "dependsOn": [  
        {  
          "containerName": "aws-otel-collector",  
          "condition": "START"  
        }  
      ],  
      {  
        "name": "aws-otel-collector",  
        "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",  
        "essential": true,  
        "command": [  
          "--config=/etc/ecs/ecs-amp.yaml"  
        ],  
        "environment": [{  
          "name": "AWS_PROMETHEUS_ENDPOINT",  
          "value": "https://aps-workspaces.aws-region.amazonaws.com/workspaces/  
ws-a1b2c3d4-5678-90ab-cdef-EXAMPLE1111/api/v1/remote_write"  
        ]},  
        "logConfiguration": {  
          "logDriver": "awslogs",  
          "options": {  
            "awslogs-create-group": "True",  
            "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",  
            "awslogs-region": "aws-region"  
          }  
        }  
      ]  
    }  
  ]  
}
```

```
        "awslogs-region": "aws-region",
        "awslogs-stream-prefix": "ecs"
    }
}
],
"networkMode": "awsvpc",
"requiresCompatibilities": [
    "FARGATE"
],
"cpu": "1024",
"memory": "3072"
}
```

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. To protect your VPC, requests that are denied by a VPC endpoint policy, but otherwise would have been allowed, are not recorded in CloudTrail.

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 turned on in 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 which CloudTrail uses 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 user or 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 allows the 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"
}
```

AWS Compute Optimizer recommendations

AWS Compute Optimizer generates recommendations for Amazon ECS task and container sizes. For more information, see [What is AWS Compute Optimizer?](#) in the *AWS Compute Optimizer User Guide*.

Task and container size recommendations for Amazon ECS services on AWS Fargate

AWS Compute Optimizer generates recommendations for Amazon ECS services on AWS Fargate. AWS Compute Optimizer recommends task CPU and task memory size and container CPU, container memory and container memory reservation sizes. These recommendations are displayed on the following pages of the Compute Optimizer console.

- **Recommendations for Amazon ECS services on Fargate** page
- **Amazon ECS services on Fargate details** page

For more information, see [Viewing recommendations for Amazon ECS services on Fargate](#) in the *AWS Compute Optimizer User Guide*.

Security in Amazon Elastic Container Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon Elastic Container Service, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon ECS. The following topics show you how to configure Amazon ECS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon ECS resources.

Topics

- [Identity and Access Management for Amazon Elastic Container Service \(p. 587\)](#)
- [Logging and Monitoring in Amazon Elastic Container Service \(p. 655\)](#)
- [Compliance validation for Amazon Elastic Container Service \(p. 656\)](#)
- [AWS Fargate Federal Information Processing Standard \(FIPS-140\) \(p. 657\)](#)
- [Infrastructure Security in Amazon Elastic Container Service \(p. 659\)](#)

Identity and Access Management for Amazon Elastic Container Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon ECS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience \(p. 588\)](#)
- [Authenticating with identities \(p. 588\)](#)
- [Managing access using policies \(p. 590\)](#)
- [How Amazon Elastic Container Service works with IAM \(p. 592\)](#)

- [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#)
- [AWS managed policies for Amazon Elastic Container Service \(p. 611\)](#)
- [Using service-linked roles for Amazon ECS \(p. 624\)](#)
- [Amazon ECS task execution IAM role \(p. 626\)](#)
- [Task IAM role \(p. 631\)](#)
- [Additional configuration for Windows IAM roles for tasks \(p. 637\)](#)
- [Amazon ECS container instance IAM role \(p. 638\)](#)
- [ECS Anywhere IAM role \(p. 642\)](#)
- [Amazon ECS CodeDeploy IAM Role \(p. 644\)](#)
- [Amazon ECS CloudWatch Events IAM Role \(p. 648\)](#)
- [Grant permission to tag resources on creation \(p. 651\)](#)
- [Troubleshooting Amazon Elastic Container Service identity and access \(p. 654\)](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon ECS.

Service user – If you use the Amazon ECS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon ECS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon ECS, see [Troubleshooting Amazon Elastic Container Service identity and access \(p. 654\)](#).

Service administrator – If you're in charge of Amazon ECS resources at your company, you probably have full access to Amazon ECS. It's your job to determine which Amazon ECS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon ECS, see [How Amazon Elastic Container Service works with IAM \(p. 592\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECS. To view example Amazon ECS identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the [AWS Sign-In User Guide](#).

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS

tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the *AWS account root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS Account Management Reference Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center (successor to AWS Single Sign-On). You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS

Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
 - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
 - **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request.

Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an

entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon Elastic Container Service works with IAM

Before you use IAM to manage access to Amazon ECS, learn what IAM features are available to use with Amazon ECS.

IAM features you can use with Amazon Elastic Container Service

IAM feature	Amazon ECS support
Identity-based policies (p. 593)	Yes
Resource-based policies (p. 593)	No
Policy actions (p. 593)	Yes
Policy resources (p. 594)	Partial
Policy condition keys (p. 595)	Yes
ACLs (p. 597)	No
ABAC (tags in policies) (p. 597)	Yes
Temporary credentials (p. 598)	Yes
Principal permissions (p. 598)	Yes
Service roles (p. 598)	Yes
Service-linked roles (p. 599)	Yes

To get a high-level view of how Amazon ECS and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for Amazon ECS

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for Amazon ECS

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).

Resource-based policies within Amazon ECS

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

Policy actions for Amazon ECS

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon ECS actions, see [Actions defined by Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

Policy actions in Amazon ECS use the following prefix before the action:

```
ecs
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "ecs:action1",  
    "ecs:action2"  
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action:

```
"Action": "ecs:Describe*"
```

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).

Policy resources for Amazon ECS

Supports policy resources	Partial
---------------------------	---------

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon ECS resource types and their ARNs, see [Resources defined by Amazon Elastic Container Service](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon Elastic Container Service](#).

Some Amazon ECS API actions support multiple resources. For example, multiple clusters can be referenced when calling the `DescribeClusters` API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "EXAMPLE-RESOURCE-1",  
    "EXAMPLE-RESOURCE-2"]
```

For example, the Amazon ECS cluster resource has the following ARN:

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

To specify `my-cluster-1` and `my-cluster-2` cluster in your statement, use the following ARNs:

```
"Resource": [  
    "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-1",  
    "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-2"]
```

To specify all clusters that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

For task definitions, you can specify the latest revision, or a specific revision.

To specify the latest task definition, use:

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/  
${TaskDefinitionFamilyName}"
```

To specify a specific task definition revision, use `${TaskDefinitionRevisionNumber}`:

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/  
${TaskDefinitionFamilyName}:${TaskDefinitionRevisionNumber}"
```

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).

Policy condition keys for Amazon ECS

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition block`) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

Amazon ECS supports the following service-specific condition keys that you can use to provide fine-grained filtering for your IAM policies:

Condition Key	Description	Evaluation Types
aws:RequestTag/ \${TagKey}	<p>The context key is formatted "aws :RequestTag/tag-key" :"tag-value" where tag-keyand tag-value are a tag key and value pair.</p> <p>Checks that the tag key–value pair is present in an AWS request. For example, you could check to see that the request includes the tag key "Dept" and that it has the value "Accounting".</p>	String
aws:ResourceTag/ \${TagKey}	<p>The context key is formatted "aws :ResourceTag/tag-key" :"tag-value" where tag-keyand tag-value are a tag key and value pair.</p> <p>Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.</p>	String
aws:TagKeys	<p>This context key is formatted "aws :TagKeys" :"tag-key" where tag-key is a list of tag keys without values (for example, ["Dept", "Cost-Center"]).</p> <p>Checks the tag keys that are present in an AWS request.</p>	String
ecs:ResourceTag/ \${TagKey}	<p>The context key is formatted "ecs :ResourceTag/tag-key" :"tag-value" where tag-keyand tag-value are a tag key and value pair.</p> <p>Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.</p>	String
ecs:cluster	The context key is formatted "ecs :cluster" :" cluster-arn " where cluster-arn is the ARN for the Amazon ECS cluster.	ARN, Null
ecs:container- instances	The context key is formatted "ecs :container- instances" :" container-instance-arns " where container-instance-arns is one or more container instance ARNs.	ARN, Null
ecs:container- name	The context key is formatted "ecs :container- name" :" container-name " where container- name - is the name of an Amazon ECS container which is defined in the ECS task definition.	String

Condition Key	Description	Evaluation Types
ecs:enable-execute-command	The context key is formatted "ecs:enable-execute-command": " <i>value</i> " where <i>value</i> - is "true" or "false".	String
ecs:enable-service-connect	The context key is formatted "ecs:enable-service-connect": " <i>value</i> " where <i>value</i> is "true" or "false".	String
ecs:namespace	The context key is formatted "ecs:namespace": " <i>namespace-arn</i> " where <i>namespace-arn</i> is the ARN for the AWS Cloud Map namespace.	ARN, Null
ecs:service	The context key is formatted "ecs:service": " <i>service-arn</i> " where <i>service-arn</i> is the ARN for the Amazon ECS service.	ARN, Null
ecs:task-definition	The context key is formatted "ecs:task-definition": " <i>task-definition-arn</i> " where <i>task-definition-arn</i> is the ARN for the Amazon ECS task definition.	ARN, Null

To see a list of Amazon ECS condition keys, see [Condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon Elastic Container Service](#).

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 599\)](#).

Access control lists (ACLs) in Amazon ECS

Supports ACLs	No
---------------	----

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon ECS

Supports ABAC (tags in policies)	Yes
----------------------------------	-----

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For more information about tagging Amazon ECS resources, see [Resources and tags \(p. 529\)](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Describing Amazon ECS services based on tags \(p. 610\)](#).

Using Temporary credentials with Amazon ECS

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for Amazon ECS

Supports principal permissions	Yes
--------------------------------	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

Service roles for Amazon ECS

Supports service roles	Yes
------------------------	-----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break Amazon ECS functionality. Edit service roles only when Amazon ECS provides guidance to do so.

Service-linked roles for Amazon ECS

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Amazon ECS service-linked roles, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

Identity-based policy examples for Amazon Elastic Container Service

By default, users and roles don't have permission to create or modify Amazon ECS resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Creating IAM policies](#) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon ECS, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices \(p. 599\)](#)
- [Allow users to view their own permissions \(p. 600\)](#)
- [Amazon ECS first-run wizard permissions \(p. 601\)](#)
- [Cluster examples \(p. 605\)](#)
- [Container instance examples \(p. 606\)](#)
- [Task definition examples \(p. 607\)](#)
- [Run Task Example \(p. 607\)](#)
- [Start task example \(p. 608\)](#)
- [List and describe task examples \(p. 608\)](#)
- [Create service example \(p. 609\)](#)
- [Update service example \(p. 610\)](#)
- [Describing Amazon ECS services based on tags \(p. 610\)](#)
- [Deny Service Connect Namespace Override Example \(p. 611\)](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon ECS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant

permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions**
 - IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ViewOwnUserInfo",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetUserPolicy",  
                "iam>ListGroupsForUser",  
                "iam>ListAttachedUserPolicies",  
                "iam>ListUserPolicies",  
                "iam GetUser"  
            ],  
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]  
        },  
        {  
            "Sid": "NavigateInConsole",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetGroupPolicy",  
                "iam:GetPolicyVersion",  
                "iam GetPolicy",  
                "iam>ListAttachedGroupPolicies",  
                "iam>ListGroupPolicies",  
                "iam>ListPolicyVersions",  
                "iam ListPolicy"  
            ]  
        }  
    ]  
}
```

```
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

Amazon ECS first-run wizard permissions

The Amazon ECS first-run wizard in the classic console 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 \(p. 612\)](#) 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",
                "appmesh>DescribeVirtualGateway",
                "appmesh>DescribeVirtualNode",
                "appmesh>ListMeshes",
                "appmesh>ListVirtualGateways",
                "appmesh>ListVirtualNodes",
                "autoscaling>CreateAutoScalingGroup",
                "autoscaling>CreateLaunchConfiguration",
                "autoscaling>DeleteAutoScalingGroup",
                "autoscaling>DeleteLaunchConfiguration",
                "autoscaling>Describe*",
                "autoscaling>UpdateAutoScalingGroup",
                "cloudformation>CreateStack",
                "cloudformation>DeleteStack",
                "cloudformation>DescribeStack*",
                "cloudformation>UpdateStack",
                "cloudwatch>DeleteAlarms",
                "cloudwatch>DescribeAlarms",
                "cloudwatch>GetMetricStatistics",
                "cloudwatch>PutMetricAlarm",
                "codedeploy>BatchGetApplicationRevisions",
                "codedeploy>BatchGetApplications",
                "codedeploy>BatchGetDeploymentGroups",
                "codedeploy>BatchGetDeployments",
                "codedeploy>ContinueDeployment",
                "codedeploy>CreateApplication",
                "codedeploy>CreateDeployment",
                "codedeploy>CreateDeploymentGroup",
                "codedeploy>GetApplication",
                "codedeploy>GetApplicationRevision",
                "codedeploy>GetDeployment",
                "codedeploy>GetDeploymentConfig",
                "codedeploy>GetDeploymentGroup",
                "codedeploy>GetDeploymentTarget",
                "codedeploy>ListApplicationRevisions",
                "codedeploy>ListApplications",
                "codedeploy>StartDeployment"
            ]
        }
    ]
}
```

```
"codedeploy>ListDeploymentConfigs",
"codedeploy>ListDeploymentGroups",
"codedeploy>ListDeployments",
"codedeploy>ListDeploymentTargets",
"codedeploy/RegisterApplicationRevision",
"codedeploy>StopDeployment",
"ec2:AssociateRouteTable",
"ec2:AttachInternetGateway",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CancelSpotFleetRequests",
"ec2>CreateInternetGateway",
"ec2>CreateLaunchTemplate",
"ec2>CreateRoute",
"ec2>CreateRouteTable",
"ec2>CreateSecurityGroup",
"ec2>CreateSubnet",
"ec2>CreateVpc",
"ec2>DeleteLaunchTemplate",
"ec2>DeleteSubnet",
"ec2>DeleteVpc",
"ec2:Describe*",
"ec2:DetachInternetGateway",
"ec2:DisassociateRouteTable",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVpcAttribute",
"ec2:RequestSpotFleet",
"ec2:RunInstances",
"ecs:*",
"elasticfilesystem:DescribeAccessPoints",
"elasticfilesystem:DescribeFileSystems",
"elasticloadbalancing>CreateListener",
"elasticloadbalancing>CreateLoadBalancer",
"elasticloadbalancing>CreateRule",
"elasticloadbalancing>CreateTargetGroup",
"elasticloadbalancing>DeleteListener",
"elasticloadbalancing>DeleteLoadBalancer",
"elasticloadbalancing>DeleteRule",
"elasticloadbalancing>DeleteTargetGroup",
"elasticloadbalancing>DescribeListeners",
"elasticloadbalancing>DescribeLoadBalancers",
"elasticloadbalancing>DescribeRules",
"elasticloadbalancing>DescribeTargetGroups",
"events>DeleteRule",
"events>DescribeRule",
"events>ListRuleNamesByTarget",
"events>ListTargetsByRule",
"events>PutRule",
"events>PutTargets",
"events>RemoveTargets",
"fsx:DescribeFileSystems",
"iam>ListAttachedRolePolicies",
"iam>ListInstanceProfiles",
"iam>ListRoles",
"lambda>ListFunctions",
"logs>CreateLogGroup",
"logs>DescribeLogGroups",
"logs>FilterLogEvents",
"route53>CreateHostedZone",
"route53>DeleteHostedZone",
"route53>GetHealthCheck",
"route53>GetHostedZone",
"route53>ListHostedZonesByName",
"servicediscovery>CreatePrivateDnsNamespace",
"servicediscovery>CreateService",
"servicediscovery>DeleteService",
"servicediscovery>GetNamespace",
```

```

        "servicediscovery:GetOperation",
        "servicediscovery:GetService",
        "servicediscovery>ListNamespaces",
        "servicediscovery>ListServices",
        "servicediscovery:UpdateService",
        "sns>ListTopics"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ssm:GetParameter",
        "ssm:GetParameters",
        "ssm:GetParametersByPath"
    ],
    "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": [
                "autoscaling.amazonaws.com",
                "ecs.amazonaws.com",
                "ecs.application-autoscaling.amazonaws.com",
                "spot.amazonaws.com",
                "spotfleet.amazonaws.com"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "elasticloadbalancing:AddTags"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "elasticloadbalancing:CreateAction": [
                "CreateTargetGroup",
                "CreateRule",
                "CreateListener",
                "CreateLoadBalancer"
            ]
        }
    }
}
]
}

```

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. The following must be true:

- Your user has administrator access. For more information, see [Set up to use Amazon ECS \(p. 10\)](#).
- 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:
 - [Using service-linked roles for Amazon ECS \(p. 624\)](#)
 - [Amazon ECS container instance IAM role \(p. 638\)](#)
 - [Amazon ECS task execution IAM role \(p. 626\)](#)

Cluster examples

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 `DescribeClusters` and `DeleteCluster` actions accept cluster ARNs as resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecs:DescribeClusters",  
                "ecs>DeleteCluster"  
            ],  
            "Resource": [  
                "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/<cluster_name>"  
            ]  
        }  
    ]  
}
```

The following IAM policy can be attached to a user or group that would only allow that user or group to perform operations on a specific cluster.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "ecs:Describe*",  
                "ecs>List*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        },  
        {  
            "Action": [  
                "ecs>DeleteCluster",  
                "ecs>DeregisterContainerInstance",  
                "ecs>ListContainerInstances",  
                "ecs:RegisterContainerInstance",  
                "ecs>SubmitContainerStateChange",  
            ]  
        }  
    ]  
}
```

```

        "ecs:SubmitTaskStateChange"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
},
{
    "Action": [
        "ecs:DescribeContainerInstances",
        "ecs:DescribeTasks",
        "ecs>ListTasks",
        "ecs:UpdateContainerAgent",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:RunTask"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "ArnEquals": {
            "ecs:cluster": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
        }
    }
}
]
}

```

Container instance examples

Container instance registration is handled by the Amazon ECS agent, but there may be times where you want to allow a user to deregister an instance manually from a cluster. Perhaps the container instance was accidentally registered to the wrong cluster, or the instance was terminated with tasks still running on it.

The following IAM policy allows a user to list and deregister container instances in a specified cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DeregisterContainerInstance",
                "ecs>ListContainerInstances"
            ],
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
            ]
        }
    ]
}
```

The following IAM policy allows a user to describe a specified container instance in a specified cluster. To open this permission up to all container instances in a cluster, you can replace the container instance UUID with *.

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

```

        ],
        "Condition": {
            "ArnEquals": {
                "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
            }
        },
        "Resource": [
            "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<cluster_name>/<container_instance_UUID>"
        ]
    }
}

```

Task definition examples

Task definition IAM policies do not support resource-level permissions, but the following IAM policy allows a user to register, list, and describe task definitions:

If you use the console, you must add CloudFormation: CreateStack as an Action.

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

Run Task Example

The resources for RunTask are task definitions. To limit which clusters a user can run task definitions on, you can specify them in the Condition block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to run any revision of a specific task definition on a specific cluster:

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

```

        "Resource": [
            "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*"
        ]
    }
}

```

Start task example

The resources for StartTask are task definitions. To limit which clusters and container instances a user can start task definitions on, you can specify them in the Condition block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to start any revision of a specific task definition on a specific cluster and specific container instance.

Note

For this example, when you call the StartTask API with the AWS CLI or another AWS SDK, you must specify the task definition revision so that the Resource mapping matches.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StartTask"
            ],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>",
                    "ecs:container-instances" : [
                        "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<cluster_name>/<container_instance_UUID>"
                    ]
                }
            },
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*"
            ]
        }
    ]
}

```

List and describe task examples

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:DescribeTasks"
            ],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
                }
            },
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:task/<cluster_name>/<task_UUID>"
            ]
        }
    ]
}
```

Create service example

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 service example

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": [
                "*"
            ]
        }
    ]
}
```

Describing Amazon ECS services based on tags

You can use conditions in your identity-based policy to control access to Amazon ECS resources based on tags. This example shows how you might create a policy that allows describing your services. However, permission is granted only if the service tag Owner has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeServices",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "*"
        },
        {
            "Sid": "ViewServiceIfOwner",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "arn:aws:ecs:*:service/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

```
        ]
    }
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to describe an Amazon ECS service, the service must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Deny Service Connect Namespace Override Example

The following IAM policy denies a user from overriding the default Service Connect namespace in a service configuration. The default namespace is set in the cluster. However, you can override it in a service configuration. For consistency, consider setting all your new services to use the same namespace. Use the following context keys to require services to use a specific namespace. Replace the `<region>`, `<aws_account_id>`, `<cluster_name>` and `<namespace_id>` with your own in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Condition": {
                "ARNEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>",
                    "ecs:namespace": "arn:aws:servicediscovery:<region>:<aws_account_id>:namespace/<namespace_id>"
                }
            },
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for Amazon Elastic Container Service

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

Amazon ECS and Amazon ECR provide several managed policies and trust relationships that you can attach to users, groups, roles, Amazon 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. For more information about the Amazon ECR managed policies, see [Amazon ECR managed policies](#).

AmazonECS_FullAccess

You can attach the AmazonECS_FullAccess policy to your IAM identities.

This policy grants administrative access to Amazon ECS resources and grants an IAM identity (such as a user, group, or role) access to the AWS services that Amazon ECS is integrated with to use all of Amazon ECS features. Using this policy allows access to all of Amazon ECS features that are available in the AWS Management Console.

Permissions details

The AmazonECS_FullAccess managed IAM policy includes the following permissions. Following the best practice of granting least privilege, you can use the AmazonECS_FullAccess managed policy as a template for creating your own custom policy. That way, you can take away or add permissions to and from the managed policy based on your specific requirements.

- **ecs** – Allows principals full access to all Amazon ECS APIs.
- **application-autoscaling** – Allows principals to create, describe, and manage Application Auto Scaling resources. This is required when enabling service auto scaling for your Amazon ECS services.
- **appmesh** – Allows principals to list App Mesh service meshes and virtual nodes and describe App Mesh virtual nodes. This is required when integrating your Amazon ECS services with App Mesh.
- **autoscaling** – Allows principals to create, manage, and describe Amazon EC2 Auto Scaling resources. This is required when managing Amazon EC2 auto scaling groups when using the cluster auto scaling feature.
- **cloudformation** – Allows principals to create and manage AWS CloudFormation stacks. This is required when creating Amazon ECS clusters using the AWS Management Console and the subsequent managing of those clusters.
- **cloudwatch** – Allows principals to create, manage, and describe Amazon CloudWatch alarms.
- **codedeploy** – Allows principals to create and manage application deployments as well as view their configurations, revisions, and deployment targets.
- **sns** – Allows principals to view a list of Amazon SNS topics.
- **lambda** – Allows principals to view a list of AWS Lambda functions and their version specific configurations.
- **ec2** – Allows principals run Amazon EC2 instances as well as create and manage routes, route tables, internet gateways, launch groups, security groups, virtual private clouds, spot fleets, and subnets.
- **elasticloadbalancing** – Allows principals to create, describe, and delete Elastic Load Balancing load balancers. Principals will also be able to add tags to newly created target groups, listeners, and listener rules for load balancers.
- **events** – Allows principals to create, manage, and delete Amazon EventBridge rules and their targets.
- **iam** – Allows principals to list IAM roles and their attached policies. Principals can also list instance profiles available to your Amazon EC2 instances.
- **logs** – Allows principals to create and describe Amazon CloudWatch Logs log groups. Principals can also list log events for these log groups.

- `route53` – Allows principals to create, manage, and delete Amazon Route 53 hosted zones. Principals can also view Amazon Route 53 health check configuration and information. For more information about hosted zones, see [Working with hosted zones](#).
 - `servicediscovery` – Allows principals to create, manage, and delete AWS Cloud Map services and create private DNS namespaces.

The following is an example AmazonECS_FullAccess policy.

```
{
  "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",
        "appmesh:DescribeVirtualGateway",
        "appmesh:DescribeVirtualNode",
        "appmesh>ListMeshes",
        "appmesh>ListVirtualGateways",
        "appmesh>ListVirtualNodes",
        "autoscaling>CreateAutoScalingGroup",
        "autoscaling>CreateLaunchConfiguration",
        "autoscaling>DeleteAutoScalingGroup",
        "autoscaling>DeleteLaunchConfiguration",
        "autoscaling>Describe*",
        "autoscaling>UpdateAutoScalingGroup",
        "cloudformation>CreateStack",
        "cloudformation>DeleteStack",
        "cloudformation>DescribeStack*",
        "cloudformation>UpdateStack",
        "cloudwatch>DeleteAlarms",
        "cloudwatch>DescribeAlarms",
        "cloudwatch>GetMetricStatistics",
        "cloudwatch>PutMetricAlarm",
        "codedeploy:BatchGetApplicationRevisions",
        "codedeploy:BatchGetApplications",
        "codedeploy:BatchGetDeploymentGroups",
        "codedeploy:BatchGetDeployments",
        "codedeploy:ContinueDeployment",
        "codedeploy>CreateApplication",
        "codedeploy>CreateDeployment",
        "codedeploy>CreateDeploymentGroup",
        "codedeploy:GetApplication",
        "codedeploy:getApplicationRevision",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:GetDeploymentTarget",
        "codedeploy>ListApplicationRevisions",
        "codedeploy>ListApplications",
        "codedeploy>ListDeploymentConfigs",
        "codedeploy>ListDeploymentGroups",
        "codedeploy>ListDeployments",
        "codedeploy>ListDeploymentTargets",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:StopDeployment",
        "ec2:AssociateRouteTable".
```

```
"ec2:AttachInternetGateway",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CancelSpotFleetRequests",
"ec2>CreateInternetGateway",
"ec2>CreateLaunchTemplate",
"ec2>CreateRoute",
"ec2>CreateRouteTable",
"ec2>CreateSecurityGroup",
"ec2>CreateSubnet",
"ec2>CreateVpc",
"ec2>DeleteLaunchTemplate",
"ec2>DeleteSubnet",
"ec2>DeleteVpc",
"ec2:Describe*",
"ec2:DetachInternetGateway",
"ec2:DisassociateRouteTable",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVpcAttribute",
"ec2:RequestSpotFleet",
"ec2:RunInstances",
"ecs:*",
"elasticfilesystem:DescribeAccessPoints",
"elasticfilesystem:DescribeFileSystems",
"elasticloadbalancing:CreateListener",
"elasticloadbalancing:CreateLoadBalancer",
"elasticloadbalancing:CreateRule",
"elasticloadbalancing:CreateTargetGroup",
"elasticloadbalancing:DeleteListener",
"elasticloadbalancing:DeleteLoadBalancer",
"elasticloadbalancing:DeleteRule",
"elasticloadbalancing:DeleteTargetGroup",
"elasticloadbalancing:DescribeListeners",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeRules",
"elasticloadbalancing:DescribeTargetGroups",
"events:DeleteRule",
"events:DescribeRule",
"events>ListRuleNamesByTarget",
"events>ListTargetsByRule",
"events:PutRule",
"events:PutTargets",
"events:RemoveTargets",
"fsx:DescribeFileSystems",
"iam>ListAttachedRolePolicies",
"iam>ListInstanceProfiles",
"iam>ListRoles",
"lambda>ListFunctions",
"logs>CreateLogGroup",
"logs:DescribeLogGroups",
"logs:FilterLogEvents",
"route53>CreateHostedZone",
"route53>DeleteHostedZone",
"route53:GetHealthCheck",
"route53:GetHostedZone",
"route53>ListHostedZonesByName",
"servicediscovery>CreatePrivateDnsNamespace",
"servicediscovery>CreateService",
"servicediscovery>DeleteService",
"servicediscovery:GetNamespace",
"servicediscovery:GetOperation",
"servicediscovery:GetService",
"servicediscovery>ListNamespaces",
"servicediscovery>ListServices",
"servicediscovery:UpdateService",
"sns>ListTopics"
],
```

```

        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:GetParameter",
            "ssm:GetParameters",
            "ssm:GetParametersByPath"
        ],
        "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": [
                "autoscaling.amazonaws.com",
                "ecs.amazonaws.com",
                "ecs.application-autoscaling.amazonaws.com",
                "spot.amazonaws.com",
                "spotfleet.amazonaws.com"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "elasticloadbalancing:AddTags"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "elasticloadbalancing:CreateAction": [
                "CreateTargetGroup",
                "CreateRule",
                "CreateListener",
                "CreateLoadBalancer"
            ]
        }
    }
}
]
```

AmazonEC2ContainerServiceforEC2Role

Amazon ECS attaches this policy to a service role that allows Amazon ECS to perform actions on your behalf against Amazon EC2 instances or external instances.

This policy grants administrative permissions that allow Amazon ECS container instances to make calls to AWS on your behalf. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

Considerations

You should consider the following recommendations and considerations when using the AmazonEC2ContainerServiceforEC2Role managed IAM policy.

- Following the standard security advice of granting least privilege, you can modify the AmazonEC2ContainerServiceforEC2Role managed policy to fit your specific needs. If any of the permissions granted in the managed policy aren't needed for your use case, create a custom policy and add only the permissions that you require. For example, the `UpdateContainerInstancesState` permission is provided for Spot Instance draining. If that permission isn't needed for your use case, exclude it using a custom policy. For more information, see [Permissions details \(p. 617\)](#).

- Containers that are running on your container instances have access to all of the permissions that are supplied to the container instance role through [instance metadata](#). We recommend that you limit the permissions in your container instance role to the minimal list of permissions that are provided in the managed AmazonEC2ContainerServiceforEC2Role policy. If the containers in your tasks need extra permissions that aren't listed, we recommend providing those tasks with their own IAM roles. For more information, see [Task IAM role \(p. 631\)](#).

You can prevent containers on the docker0 bridge from accessing the permissions supplied to the container instance role. You can do this while still allowing the permissions that are provided by [Task IAM role \(p. 631\)](#) by running the following **iptables** command on your container instances. Containers can't query instance metadata with this rule in effect. This command assumes the default Docker bridge configuration and it doesn't work with containers that use the host network mode. For more information, see [Network mode \(p. 800\)](#).

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
```

You must save this **iptables** rule on your container instance for it to survive a reboot. For the Amazon ECS-optimized AMI, use the following command. For other operating systems, consult the documentation for that OS.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo service iptables save
```

Permissions details

The AmazonEC2ContainerServiceforEC2Role managed IAM policy includes the following permissions. Following the standard security advice of granting least privilege, the AmazonEC2ContainerServiceforEC2Role managed policy can be used as a guide. If you don't need any of the permissions that are granted in the managed policy for your use case, create a custom policy and add only the permissions that you need.

- ec2:DescribeTags** – Allows a principal to describe the tags that are associated with an Amazon EC2 instance. This permission is used by the Amazon ECS container agent to support resource tag propagation. For more information, see [How resources are tagged \(p. 529\)](#).
- ecs:CreateCluster** – Allows a principal to create an Amazon ECS cluster. This permission is used by the Amazon ECS container agent to create a default cluster, if one doesn't already exist.
- ecs:DeregisterContainerInstance** – Allows a principal to deregister an Amazon ECS container instance from a cluster. The Amazon ECS container agent doesn't call this API, but this permission remains to ensure backwards compatibility.
- ecs:DiscoverPollEndpoint** – This action returns endpoints that the Amazon ECS container agent uses to poll for updates.
- ecs:Poll** – Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report task state changes.
- ecs:RegisterContainerInstance** – Allows a principal to register a container instance with a cluster. This permission is used by the Amazon ECS container agent to register the Amazon EC2 instance with a cluster as well as to support resource tag propagation.
- ecs:StartTelemetrySession** – Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report health information and metrics for each container and task.

- `ecs:TagResource` – Allows the Amazon ECS container agent to tag cluster on creation and to tag container instances when they are registered to a cluster.
- `ecs:UpdateContainerInstancesState` – Allows a principal to modify the status of an Amazon ECS container instance. This permission is used by the Amazon ECS container agent for Spot Instance draining. For more information, see [Spot Instance Draining \(p. 279\)](#).
- `ecs:Submit*` – This includes the `SubmitAttachmentStateChanges`, `SubmitContainerStateChange`, and `SubmitTaskStateChange` API actions. They're used by the Amazon ECS container agent to report state changes for each resource to the Amazon ECS control plane. The `SubmitContainerStateChange` permission is no longer used by the Amazon ECS container agent but remains to ensure backwards compatibility.
- `ecr:GetAuthorizationToken` – Allows a principal to retrieve an authorization token. The authorization token represents your IAM authentication credentials and can be used to access any Amazon ECR registry that the IAM principal has access to. The authorization token received is valid for 12 hours.
- `ecr:BatchCheckLayerAvailability` – When a container image is pushed to an Amazon ECR private repository, each image layer is checked to verify if it's already pushed. If it is, then the image layer is skipped.
- `ecr:GetDownloadUrlForLayer` – When a container image is pulled from an Amazon ECR private repository, this API is called once for each image layer that's not already cached.
- `ecr:BatchGetImage` – When a container image is pulled from an Amazon ECR private repository, this API is called once to retrieve the image manifest.
- `logs>CreateLogStream` – Allows a principal to create a CloudWatch Logs log stream for a specified log group.
- `logs:PutLogEvents` – Allows a principal to upload a batch of log events to a specified log stream.

The following is an example `AmazonEC2ContainerServiceforEC2Role` policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeTags",
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:UpdateContainerInstancesState",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs>CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ecs:TagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "AWS:Principal": "arn:aws:lambda:us-east-1:123456789012:lambdaFunctionName"
        }
      }
    }
  ]
}
```

```
    "ecs:CreateAction": [
        "CreateCluster",
        "RegisterContainerInstance"
    ]
}
]
}
}
```

AmazonEC2ContainerServiceEventsRole

This policy grants permissions that allow Amazon EventBridge (formerly CloudWatch Events) to run tasks on your behalf. This policy can be attached to the IAM role that's specified when you create scheduled tasks. For more information, see [Amazon ECS CloudWatch Events IAM Role \(p. 648\)](#).

Permissions details

This policy includes the following permissions.

- **ecs** – Allows a principal in a service to call the Amazon ECS RunTask API. Allows a principal in a service to add tags (TagResource) when they call the Amazon ECS RunTask API.
- **iam** – Allows passing any IAM service role to any Amazon ECS tasks.

The following is an example AmazonEC2ContainerServiceEventsRole policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ecs:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": [
                        "RunTask"
                    ]
                }
            }
        }
    ]
}
```

```
    }
  ]
}
```

AmazonECSTaskExecutionRolePolicy

The `AmazonECSTaskExecutionRolePolicy` managed IAM policy grants the permissions that are needed by the Amazon ECS container agent and AWS Fargate container agents to make AWS API calls on your behalf. This policy can be added to your task execution IAM role. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Permissions details

The `AmazonECSTaskExecutionRolePolicy` managed IAM policy includes the following permissions. Following the standard security advice of granting least privilege, the `AmazonECSTaskExecutionRolePolicy` managed policy can be used as a guide. If any of the permissions that are granted in the managed policy aren't needed for your use case, create a custom policy and add only the permissions that you require.

- `ecr:GetAuthorizationToken` – Allows a principal to retrieve an authorization token. The authorization token represents your IAM authentication credentials and can be used to access any Amazon ECR registry that the IAM principal has access to. The authorization token received is valid for 12 hours.
- `ecr:BatchCheckLayerAvailability` – When a container image is pushed to an Amazon ECR private repository, each image layer is checked to verify if it's already pushed. If it's pushed, then the image layer is skipped.
- `ecr:GetDownloadUrlForLayer` – When a container image is pulled from an Amazon ECR private repository, this API is called once for each image layer that's not already cached.
- `ecr:BatchGetImage` – When a container image is pulled from an Amazon ECR private repository, this API is called once to retrieve the image manifest.
- `logs>CreateLogStream` – Allows a principal to create a CloudWatch Logs log stream for a specified log group.
- `logs:PutLogEvents` – Allows a principal to upload a batch of log events to a specified log stream.

The following is an example `AmazonECSTaskExecutionRolePolicy` policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

AWSApplicationAutoscalingECSServicePolicy

You can't attach AWSApplicationAutoscalingECSServicePolicy to your IAM entities. This policy is attached to a service-linked role that allows Application Auto Scaling to perform actions on your behalf. For more information, see [Service-linked roles for Application Auto Scaling](#).

AWSCodeDeployRoleForECS

You can't attach AWSCodeDeployRoleForECS to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see [Create a service role for CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

AWSCodeDeployRoleForECSLimited

You can't attach AWSCodeDeployRoleForECSLimited to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see [Create a service role for CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

Amazon ECS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon ECS since this service started tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [Amazon ECS Document history page](#).

Change	Description	Date
Add permissions to AmazonEC2ContainerServiceEventsRole	The AmazonEC2ContainerServiceEventsRole policy was modified to add the <code>ecs:TagResource</code> permission which allows a principal in a service to add tags when they call the Amazon ECS RunTask API.	March 6, 2023
Add permissions to AmazonEC2ContainerServiceforEC2Role	The AmazonEC2ContainerServiceforEC2Role policy was modified to add the <code>ecs:TagResource</code> permission which includes a condition which limits the permission only to newly created clusters and registered container instances.	March 6, 2023
Add permissions to the section called "AmazonECS_FullAccess" (p. 612)	The <code>AmazonECS_FullAccess</code> policy was modified to add the <code>elasticloadbalancing:AddTags</code> permission which includes a condition which limits the permission only to newly created load balancers, target groups, rules, and listeners created. This permission doesn't allow tags to be added to any already created Elastic Load Balancing resources.	January 4, 2023

Change	Description	Date
Amazon ECS started tracking changes	Amazon ECS started tracking changes for its AWS managed policies.	June 8, 2021

Phased out AWS managed IAM policies for Amazon Elastic Container Service

The following AWS managed IAM policies are phased out. These policies are now replaced by the updated policies. We recommend that you update your users or roles to use the updated policies.

AmazonEC2ContainerServiceFullAccess

Important

The `AmazonEC2ContainerServiceFullAccess` managed IAM policy was phased out as of January 29, 2021, in response to a security finding with the `iam:passRole` permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new users or roles. Any users or roles that already have the policy attached can continue using it. However, we recommend that you update your users or roles to use the `AmazonECS_FullAccess` managed policy instead. For more information, see [Migrating to the `AmazonECS_FullAccess` managed policy \(p. 622\)](#).

AmazonEC2ContainerServiceRole

Important

The `AmazonEC2ContainerServiceRole` managed IAM policy is phased out. It's now replaced by the Amazon ECS service-linked role. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

AmazonEC2ContainerServiceAutoscaleRole

Important

The `AmazonEC2ContainerServiceAutoscaleRole` managed IAM policy is phased out. It's now replaced by the Application Auto Scaling service-linked role for Amazon ECS. For more information, see [Service-linked roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

Migrating to the `AmazonECS_FullAccess` managed policy

The `AmazonEC2ContainerServiceFullAccess` managed IAM policy was phased out on January 29, 2021, in response to a security finding with the `iam:passRole` permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new groups, users, or roles. Any groups, users, or roles that already have the policy attached can continue using it. However, we recommend that you update your groups, users, or roles to use the `AmazonECS_FullAccess` managed policy instead.

The permissions that are granted by the `AmazonECS_FullAccess` policy include the complete list of permissions that are necessary to use ECS as an administrator. If you currently use permissions that are granted by the `AmazonEC2ContainerServiceFullAccess` policy that aren't in the `AmazonECS_FullAccess` policy, you can add them to an in-line policy statement. For more information, see [AWS managed policies for Amazon Elastic Container Service \(p. 611\)](#).

Use the following steps to determine if you have any groups, users, or roles that are currently using the `AmazonEC2ContainerServiceFullAccess` managed IAM policy. Then, update them to detach the earlier policy and attach the `AmazonECS_FullAccess` policy.

To update a group, user, or role to use the **AmazonECS_FullAccess** policy (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** and search for and select the **AmazonEC2ContainerServiceFullAccess** policy.
3. Choose the **Policy usage** tab that displays any IAM role that's currently using this policy.
4. For each IAM role that's currently using the **AmazonEC2ContainerServiceFullAccess** policy, select the role and use the following steps to detach the deprecated policy and attach the **AmazonECS_FullAccess** policy.
 - a. On the **Permissions** tab, choose the X next to the **AmazonEC2ContainerServiceFullAccess** policy.
 - b. Choose **Add permissions**.
 - c. Choose **Attach existing policies directly**, search for and select the **AmazonECS_FullAccess** policy, and then choose **Next: Review**.
 - d. Review the changes and then choose **Add permissions**.
 - e. Repeat these steps for each group, user, or role that's using the **AmazonEC2ContainerServiceFullAccess** policy.

To update a group, user, or role to use the **AmazonECS_FullAccess** policy (AWS CLI)

1. Use the [generate-service-last-accessed-details](#) command to generate a report that includes details about when the deprecated policy was last used.

```
aws iam generate-service-last-accessed-details \
--arn arn:aws:iam::aws:policy/AmazonEC2ContainerServiceFullAccess
```

Example output:

```
{  
    "JobId": "32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE"  
}
```

2. Use the job ID from the previous output with the [get-service-last-accessed-details](#) command to retrieve the last accessed report of the service. This report displays the Amazon Resource Name (ARN) of the IAM entities that last used the deprecated policy.

```
aws iam get-service-last-accessed-details \
--job-id 32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE
```

3. Use one of the following commands to detach the **AmazonEC2ContainerServiceFullAccess** policy from a group, user, or role.
 - [detach-group-policy](#)
 - [detach-role-policy](#)
 - [detach-user-policy](#)
4. Use one of the following commands to attach the **AmazonECS_FullAccess** policy to a group, user, or role.
 - [attach-group-policy](#)
 - [attach-role-policy](#)
 - [attach-user-policy](#)

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. The service-linked role is predefined by Amazon ECS and includes 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.

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 roles** 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**.

The AWSServiceRoleForECS service-linked role trusts the following services to assume the role:

- `ecs.amazonaws.com`

The role permissions policy named `AmazonECSServiceRolePolicy` allows Amazon ECS to complete the following actions on the specified resources:

- Action: When using the `awsvpc` network mode for your Amazon ECS tasks, Amazon ECS manages the lifecycle of the elastic network interfaces associated with the task. This also includes tags that Amazon ECS adds to your elastic network interfaces.
- Action: When using a load balancer with your Amazon ECS service, Amazon ECS manages the registration and deregistration of resources with the load balancer.
- Action: When using Amazon ECS service discovery, Amazon ECS manages the required Route 53 and AWS Cloud Map resources for service discovery to work.
- Action: When using Amazon ECS service auto scaling, Amazon ECS manages the required Auto Scaling resources.
- Action: Amazon ECS creates and manages CloudWatch alarms and log streams that assist in the monitoring of your Amazon ECS resources.
- Action: When using Amazon ECS Exec, Amazon ECS manages the permissions needed to start Amazon ECS Exec sessions to your tasks.
- Action: When using Amazon ECS Service Connect, Amazon ECS manages the required AWS Cloud Map resources to use the feature.

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. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.

Creating a service-linked role for Amazon ECS

You don't need to manually create a service-linked role. When you create a cluster or create or update a service in the AWS Management Console, the AWS CLI, or the AWS API, Amazon ECS creates the service-linked role for you.

Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster or create or update a service, Amazon ECS creates the service-linked role for you again.

Editing a service-linked role for Amazon ECS

Amazon ECS doesn't 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. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Amazon ECS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note

If the Amazon ECS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

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.

1. Scale all Amazon ECS services down to a desired count of 0 in all regions, and then delete the services. For more information, see [Updating a service using the classic console \(p. 911\)](#) and [Deleting a service using the classic console \(p. 913\)](#).
2. Force deregister all container instances from all clusters in all regions. For more information, see [Deregister an Amazon EC2 backed container instance \(p. 335\)](#).
3. Delete all Amazon ECS clusters in all regions. For more information, see [Deleting a cluster using the classic console \(p. 886\)](#).

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForECS service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Supported regions for Amazon ECS service-linked roles

Amazon ECS supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS regions and endpoints](#).

Amazon ECS task execution IAM role

The task execution role grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. The task execution IAM role is required depending on the requirements of your task. You can have multiple task execution roles for different purposes and services associated with your account. For the IAM permissions that your application needs to run, see [Task IAM role \(p. 631\)](#).

The following are common use cases for a task execution IAM role:

- Your task is hosted on AWS Fargate or on an external instance and...
 - is pulling a container image from an Amazon ECR private repository.
 - is pulling a container image from an Amazon ECR private repository in a different account from the account that runs the task.
 - sends container logs to CloudWatch Logs using the awslogs log driver. For more information, see [Using the awslogs log driver \(p. 161\)](#).
- Your tasks are hosted on either AWS Fargate or Amazon EC2 instances and...
 - is using private registry authentication. For more information, see [Required IAM permissions for private registry authentication \(p. 628\)](#).
 - the task definition is referencing sensitive data using Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see [Required IAM permissions for Amazon ECS secrets \(p. 629\)](#).

Note

The task execution role is supported by Amazon ECS container agent version 1.16.0 and later.

Amazon ECS provides the managed policy named `AmazonECSTaskExecutionRolePolicy` which contains the permissions the common use cases described above require. It might be necessary to add inline policies to your task execution role for special use cases which are outlined below.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecr:GetAuthorizationToken",  
                "ecr:BatchCheckLayerAvailability",  
                "ecr:GetDownloadUrlForLayer",  
                "ecr:BatchGetImage",  
                "logs>CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

An Amazon ECS task execution role can be created for you in the Amazon ECS console; 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.

Checking for the task execution (ecsTaskExecutionRole) 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. In the search box, enter `ecsTaskExecutionRole`. If the role does exist, choose the role to view the attached policies.
4. On the **Permissions** tab, verify that the **AmazonECSTaskExecutionRolePolicy** is attached to the role.
 - a. Choose **Add Permissions, Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, enter **AmazonECSTaskExecutionRolePolicy**.
 - c. Check the box to the left of the **AmazonECSTaskExecutionRolePolicy** policy, and then choose **Attach policy**.
5. Choose **Trust relationships**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update policy**.

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

Creating the task execution (ecsTaskExecutionRole) role

If your account does not already have a task execution role, use the following steps to create the role.

To create a task execution IAM role (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles, Create role**.
3. In the **Trusted entity type** section, choose **AWS service, Elastic Container Service**.
4. For **Use case**, choose **Elastic Container Service Task**, and then choose **Next**.
5. In the **Attach permissions policy** section, do the following:
 - a. Search for **AmazonECSTaskExecutionRolePolicy**, then select the policy.
 - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
 - c. Choose **Next**.
6. Under **Role details**, do the following:
 - a. For **Role name**, type `ecsTaskExecutionRole`.

- b. For **Add tags (optional)**, specify any custom tags to associate with the policy .
7. Choose **Create role**.

To create a task execution IAM role (AWS CLI)

1. Create a file named `ecs-tasks-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

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

2. Create an IAM role named `ecsTaskExecutionRole` using the trust policy created in the previous step.

```
aws iam create-role \  
  --role-name ecsTaskExecutionRole \  
  --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. Attach the AWS managed `AmazonECSTaskExecutionRolePolicy` policy to the `ecsTaskExecutionRole` role. This policy provides

```
aws iam attach-role-policy \  
  --role-name ecsTaskExecutionRole \  
  --policy-arn arn:aws:iam::aws:policy/service-role/  
AmazonECSTaskExecutionRolePolicy
```

Required IAM permissions for private registry authentication

The Amazon ECS task execution role is required to use this feature. This allows the container agent to pull the container image.

To provide access to the secrets that you create, 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 Amazon Resource Name (ARN) for your custom key must be added as a resource.

The following is an example inline policy that adds the permissions.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "secretsmanager:GetSecretValue",  
      "Resource": "arn:aws:kms:us-east-1:123456789012:alias/MyCustomKey"  
    }  
  ]  
}
```

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

Required IAM permissions for Amazon ECS secrets

To use the Amazon ECS secrets feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager or Secrets Manager resources. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

Using Secrets Manager

To provide access to the Secrets Manager secrets that you create, manually add the following permission to the task execution role. For information about how to manage permissions, see [Adding and Removing IAM identity permissions](#) in the *IAM User Guide*.

- `secretsmanager:GetSecretValue`— Required if you are referencing a Secrets Manager secret. Adds the permission to retrieve the secret from Secrets Manager.

The following example policy adds the required permissions.

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

Using Systems Manager

Important

For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

To provide access to the Systems Manager Parameter Store parameters that you create, manually add the following permissions as a policy to the task execution role. For information about how to manage permissions, see [Adding and Removing IAM identity permissions](#) in the *IAM User Guide*.

- `ssm:GetParameters` — Required if you are referencing a Systems Manager Parameter Store parameter in a task definition. Adds the permission to retrieve Systems Manager parameters.

- `secretsmanager:GetSecretValue` — Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition. Adds the permission to retrieve the secret from Secrets Manager.
- `kms:Decrypt` — Required only if your secret uses a customer managed key and not the default key. The ARN for your custom key should be added as a resource. Adds the permission to decrypt the customer managed key .

The following example policy adds the required permissions:

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

Optional IAM permissions for Fargate tasks pulling Amazon ECR images over interface endpoints

When launching tasks that use the Fargate launch type that pull images from Amazon ECR when Amazon ECR is configured to use an interface VPC endpoint, you can restrict the tasks access to a specific VPC or VPC endpoint. Do this by creating a task execution role for the tasks to use that use IAM condition keys.

Use the following IAM global condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see [AWS Global Condition Context Keys](#).

- `aws:SourceVpc`—Restricts access to a specific VPC.
- `aws:SourceVpce`—Restricts access to a specific VPC endpoint.

The following task execution role policy provides an example for adding condition keys:

Important

The `ecr:GetAuthorizationToken` API action cannot have the `aws:sourceVpc` or `aws:sourceVpce` condition keys applied to it because the `GetAuthorizationToken` API call goes through the elastic network interface owned by AWS Fargate rather than the elastic network interface of the task.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "logs>CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Condition": {
        "aws:SourceVpc": "source_vpc_id" // Add this condition key if you want to restrict access to a specific VPC
      }
    }
  ]
}
```

```

        "logs:PutLogEvents"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:sourceVpce": "vpce-xxxxxx",
            "aws:sourceVpc": "vpc-xxxxx"
        }
    }
}
]
}

```

Task IAM role

Your Amazon ECS tasks can have an IAM role associated with them. The permissions granted in the IAM role are assumed by the containers running in the task. For the IAM permissions that Amazon ECS needs to pull container images and run the task, see [Amazon ECS task execution IAM role \(p. 626\)](#).

If your containerized applications need to call AWS APIs, they must sign their AWS API requests with AWS credentials, and a task IAM role provides a strategy for managing credentials for your applications to use, similar to the way that an Amazon EC2 instance profile provides credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with an Amazon ECS task definition or RunTask API operation. Your containers can then use the AWS SDK or AWS CLI to make API requests to authorized AWS services.

The following explain the benefits of using IAM roles with your 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.

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 Amazon EC2 or external instance they are running on.

You can specify a task IAM role 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`.

From inside the container, you can query the credential endpoint 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"  
}
```

If your Amazon EC2 instance is using at least version 1.11.0 of the container agent and a supported version of the AWS CLI or SDKs, then the SDK client will see that the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` variable is available, and it will use the provided credentials to make calls to the AWS APIs. For more information, see [Using task IAM roles on your Amazon EC2 or external instances \(p. 633\)](#).

Each time the credential provider is used, the request is logged locally on the host container instance at `/var/log/ecs/audit.log.YYYY-MM-DD-HH`. For more information, see [IAM Roles for Tasks Credential Audit Log \(p. 784\)](#).

Important

When creating your task IAM role, it is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys in either the trust relationship or the IAM policy associated with the role to prevent the confused deputy security issue. Using the `aws:SourceArn` condition key to specify a specific cluster is not currently supported, you should use the wildcard to specify all clusters. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

Considerations for tasks hosted on Amazon EC2 instances

When using an IAM role with your tasks that are running on Amazon EC2 instances, the containers aren't prevented from accessing the credentials that are supplied to the Amazon EC2 instance profile (through the Amazon EC2 instance metadata server). We recommend that you limit the permissions in your container instance role to the minimal list of permissions used in the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

The following should also be considered when using a task IAM role for tasks hosted on Amazon EC2 instances.

- To prevent containers run by tasks that use the `awsvpc` network mode from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role, set the `ECS_AWSVPC_BLOCK_IMDS` agent configuration variable to `true` in the agent configuration file and restart the agent. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).
- To prevent containers run by tasks that use the `bridge` network mode from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role, by running the following `iptables` command on your Amazon EC2 instances. This command doesn't affect containers in tasks that use the `host` or `awsvpc` network modes. For more information, see [Network mode \(p. 800\)](#).

```
sudo yum install -y iptables-services; sudo iptables --insert DOCKER-USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
```

You must save this **iptables** rule on your Amazon EC2 instance for it to survive a reboot. When using the Amazon ECS-optimized AMI, you can use the following command. For other operating systems, consult the documentation for that operating system.

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
```

Using task IAM roles on your Amazon EC2 or external instances

Your Amazon EC2 or external instances require at least version 1.11.0 of the container agent to use task IAM roles; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#). If you are using an Amazon ECS-optimized AMI, your instance needs at least 1.11.0-1 of the `ecs-init` package. If your instances are using the latest Amazon ECS-optimized AMI, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

If you are not using the Amazon ECS-optimized AMI for your container instances, be sure to add the `--net=host` option to your `docker run` command that starts the agent and the following agent configuration variables for your desired configuration (for more information, see [Amazon ECS container agent configuration \(p. 370\)](#)):

`ECS_ENABLE_TASK_IAM_ROLE=true`

Uses IAM roles for tasks for containers with the bridge and default network modes.

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true`

Uses IAM roles for tasks for containers with the host network mode. This variable is only supported on agent versions 1.12.0 and later.

For an example run command, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 368\)](#). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

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

You must save these **iptables** rules on your container instance for them to survive a reboot. You can use the `iptables-save` and `iptables-restore` commands to save your **iptables** rules and restore them at boot. For more information, consult your specific operating system documentation.

Creating an IAM role and policy for your tasks

When creating an IAM policy for your tasks to use, the policy should include the permissions that you would like the containers in your tasks to assume. You can use an existing AWS managed policy or you can create a custom policy from scratch that meets your specific needs. For more information, see [Creating IAM policies in the IAM User Guide](#).

Important

For Amazon ECS tasks (for all launch types), we recommend that you use the IAM policy and role for your tasks. These credentials allow your task to make AWS API requests without calling `sts:AssumeRole` to assume the same role that is already associated with the task. If your task requires that a role assumes itself, you must create a trust policy that explicitly allows that role to assume itself. For more information, see [Modifying a role trust policy](#) in the *IAM User Guide*.

After the IAM policy is created, you can create an IAM role which includes that policy which you reference in your Amazon ECS task definition. You can create the role using the **Elastic Container Service Task** use case 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.

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.

For information about the service endpoint for your Region, see [Service endpoints](#) in the *Amazon Web Services General Reference Reference Guide*.

The IAM task role must have a trust policy that specifies the `ecs-tasks.amazonaws.com` service. The `sts:AssumeRole` permission allows your tasks to assume an IAM role that's different from the one that the Amazon EC2 instance uses. This way, your task doesn't inherit the role associated with the Amazon EC2 instance. It is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys to scope the permissions further to prevent the confused deputy security issue. These condition keys can be specified in the trust relationship or in the IAM policy associated with the role. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

The following is an example trust policy. You should replace the Region identifier and specify the AWS account number that you use when launching tasks.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "ecs-tasks.amazonaws.com"  
                ]  
            },  
            "Action": "sts:AssumeRole",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:ecs:us-west-2:111122223333:*"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "111122223333"  
                }  
            }  
        }  
    ]  
}
```

Important

When creating your task IAM role, it is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys in either the trust relationship or the IAM policy associated with the role to prevent the confused deputy security issue. Using the `aws:SourceArn` condition key to specify a specific cluster is not currently supported, you should use the wildcard to specify all clusters. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

To create an IAM policy for your tasks (AWS Management Console)

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 Amazon Resource Name (ARN) of your Amazon S3 bucket.
4. (Optional) For **Request conditions**, select **Add condition**. This is recommended to prevent the confused deputy security issue. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.
 - a. For **Condition key**, select either **aws:SourceAccount** or **aws:SourceArn**. For more information about these global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.
 - b. For **Operator**, select **StringEquals** if you specified the **aws:SourceAccount** condition key or **ArnLike** if you specified the **aws:SourceArn** condition key.
 - c. For **Value**, specify your AWS account ID if you specified the **aws:SourceAccount** condition key or the Amazon Resource Name (ARN) of your Amazon ECS task if you specified the **aws:SourceArn** condition key. You may use wildcards, for example `aws:ecs:/*:accountId:*` which will work for all tasks in your account.
 - d. Choose **Add** to save the condition key. Repeat these steps for each condition key you want to add to the policy.
5. Choose **Next: Tags** and add any resource tags to the policy to help you organize them and then choose **Next: Review**.
6. On the **Review policy** page, for **Name** type your own unique name, such as `AmazonECSTaskS3BucketPolicy`. You may specify an optional description for the policy as well.
7. When the policy is complete, 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. It includes a condition statement, which you can use to specify either a specific task using its Amazon Resource Name (ARN) or a specific account ID. This provides a way to further scope the permission for additional security. This is recommended to prevent the confused deputy security issue. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

The following is an example permissions policy. You can modify the policy to suit your specific needs. You should replace the Region identifier and specify the AWS account number that you use when launching tasks.

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-task-secrets-bucket/*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:ecs:us-west-2:111122223333:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        }
    ]
}

```

2. Choose **Next: Tags** and add any resource tags to the policy to help you organize them and then choose **Next: Review**.
3. On the **Review policy** page, for **Name** type your own unique name, such as **AmazonECSTaskS3BucketPolicy**. You may specify an optional description for the policy as well.
4. When the policy is complete, choose **Create policy** to finish.

To create an IAM role for your tasks (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select trusted entity** section, choose **AWS service**.
4. For **Use case**, using the drop down menu, select **Elastic Container Service** and then the **Elastic Container Service Task** use case and then choose **Next**.
5. For **Add permissions**, search for and select the policy to use for your tasks (in this example **AmazonECSTaskS3BucketPolicy**), and then choose **Next**.
6. On **Step 3: Name, review, and create**, do the following:
 - a. For **Role name**, enter a name for your role. For this example, type **AmazonECSTaskS3BucketRole** to name the role.
 - b. (Optional) For **Description**, specify a description for this IAM role.
 - c. Review the trusted entity and permissions policy for the role.
 - d. For **Add tags (Optional)**, enter any metadata tags you want to associate with the IAM role, and then choose **Create role**.

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 classic console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS

CLI or SDKs, specify the Amazon Resource Name (ARN) of your task role using the `taskRoleArn` parameter. For more information, see [Creating a task definition using the console \(p. 125\)](#).

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 classic 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 [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#).

Note

In addition to the standard Amazon ECS permissions required to run tasks and services, users also require `iam:PassRole` permissions to use IAM roles for tasks.

Additional configuration for Windows IAM roles for tasks

Important

For Windows containers on Fargate that use task roles, no further action is necessary. For Windows containers on EC2 that use task roles, follow these steps.

The IAM roles for tasks with Windows features requires additional configuration on EC2, but much of this configuration is similar to configuring IAM roles for tasks on Linux container instances. The following requirements must be met to configure IAM roles for tasks for Windows containers.

- When you launch your container instances, you must set the `-EnableTaskIAMRole` option in the container instances user data script. The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks. For example:

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
</powershell>
```

- You must bootstrap your container with the networking commands that are provided in [IAM roles for task container bootstrap script \(p. 638\)](#).
- You must create an IAM role and policy for your tasks. For more information, see [Creating an IAM role and policy for your tasks \(p. 633\)](#).
- You must specify the IAM role you created for your tasks when you register the task definition, or as an override when you run the task. For more information, see [Specifying an IAM role for your tasks \(p. 636\)](#).
- The IAM roles for the task credential provider use port 80 on the container instance. Therefore, if you configure IAM roles for tasks on your container instance, your containers can't use port 80 for the host port in any port mappings. To expose your containers on port 80, we recommend configuring a service for them that uses load balancing. You can use port 80 on the load balancer. By doing so, traffic can be routed to another host port on your container instances. For more information, see [Service load balancing \(p. 486\)](#).
- If your Windows instance is restarted, you must delete the proxy interface and initialize the Amazon ECS container agent again to bring the credential proxy back up.

IAM roles for task container bootstrap script

Before containers can access the credential proxy on the container instance to get credentials, the container must be bootstrapped with the required networking commands. The following code example script should be run on your containers when they start.

Note

You do not need to run this script when you use `awsvpc` network mode on Windows.

If you run Windows containers which include Powershell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
    RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-Object
    | Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop $gateway
    -PolicyStore ActiveStore # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
    $gateway -PolicyStore ActiveStore # metadata API
```

If you run Windows containers that only have the Command shell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

for /f "tokens=1" %i in ('netsh interface ipv4 show interfaces ^| findstr /x /r
    ".*vEthernet.*"') do set interface=%i
for /f "tokens=3" %i in ('netsh interface ipv4 show addresses %interface% ^| findstr /x /r
    ".*Default.Gateway.*"') do set gateway=%i
netsh interface ipv4 add route prefix=169.254.170.2/32 interface="%interface%" 
    nexthop="%gateway%" store=active # credentials API
netsh interface ipv4 add route prefix=169.254.169.254/32 interface="%interface%" 
    nexthop="%gateway%" store=active # metadata API
```

Amazon ECS container instance IAM role

Amazon ECS container instances, including both Amazon EC2 and external instances, run the Amazon ECS container agent and require an IAM role for the service to know that the agent belongs to you.

Before you launch container instances and register them to a cluster, you must create an IAM role for your container instances to use. The role is created in the account that you use to log into the console or run the AWS CLI commands.

Important

If you are registering external instances to your cluster, the IAM role you use requires Systems Manager permissions as well. For more information, see [Required IAM permissions for external instances \(p. 340\)](#).

Amazon ECS provides the **AmazonEC2ContainerServiceforEC2Role** managed IAM policy which contains the permissions needed to use the full Amazon ECS feature set. This managed policy can be attached to an IAM role and associated with your container instances. Alternatively, you can use the managed policy as a guide when creating a custom policy to use. The container instance role provides permissions needed for the Amazon ECS container agent and Docker daemon to call AWS APIs on your behalf. For more information on the managed policy, see [AmazonEC2ContainerServiceforEC2Role \(p. 616\)](#).

Checking for the container instance (`ecsInstanceRole`) in the IAM console

You can manually create the role and attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. Use the following procedure to check and see if your account already has the Amazon ECS container instance IAM role and to attach the managed IAM policy if needed.

Important

The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. In the search box, enter `ecsInstanceRole`. If the role does exist, choose the role to view the attached policies.
4. On the **Permissions** tab, verify that the **AmazonEC2ContainerServiceforEC2Role** is attached to the role.
 - a. Choose **Add Permissions, Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, enter **AmazonEC2ContainerServiceforEC2Role**.
 - c. Check the box to the left of the **AmazonEC2ContainerServiceforEC2Role** policy, and then choose **Attach policy**.
5. Choose **Trust relationships**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update policy**.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "ec2.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

```
        ]  
    }
```

Creating the container instance (`ecsInstanceRole`) role

Important

If you are registering external instances to your cluster, see [Required IAM permissions for external instances \(p. 340\)](#).

You can manually create the role and attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. Use the following procedure to check and see if your account already has the Amazon ECS container instance IAM role and to attach the managed IAM policy if needed.

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

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then under **Use cases for other AWS services**, choose **Elastic Container Service**.
4. Choose the **EC2 Role for Elastic Container Service** use case, and then choose **Next: Permissions**.
5. In the **Permissions policies** section, verify the **AmazonEC2ContainerServiceforEC2Role** policy is selected, and then choose **Next**.

Important

The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

6. For **Role name**, enter `ecsInstanceRole` and optionally you can enter a description.
7. For **Add tags (optional)**, enter any custom tags to associate with the policy, and then choose **Next: Review**.
8. Review your role information and then choose **Create role** to finish.

To create the `ecsInstanceRole` role (AWS CLI)

1. Create an instance profile named `ecsInstanceRole-profile` using the [create-instance-profile](#) command.

```
aws iam create-instance-profile --instance-profile-name ecsInstanceRole-profile
```

Example response

```
{  
    "InstanceProfile": {  
        "InstanceProfileId": "AIPAJTLBPJLEGREXAMPLE",  
        "Roles": [],  
        "CreateDate": "2022-04-12T23:53:34.093Z",  
        "InstanceProfileName": "ecsInstanceRole-profile",  
        "Path": "/",  
        "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole-profile"  
    }  
}
```

2. Add the `ecsInstanceRole` role to the `ecsInstanceRole-profile` instance profile.

```
aws iam add-role-to-instance-profile \
--instance-profile-name ecsInstanceRole-profile \
--role-name ecsInstanceRole
```

Adding Amazon S3 read-only access to your container instance (ecsInstanceRole) role

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

For more information about creating an `ecs.config` file, storing it in Amazon S3, and launching instances with this configuration, see [Storing container instance configuration in Amazon S3 \(p. 370\)](#).

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

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. In the **Filter policies** search box, enter **AmazonS3ReadOnlyAccess**, and then choose the policy.

Note

This policy allows read-only access to all Amazon S3 resources. For more restrictive bucket policy examples, see [Bucket Policy Examples](#) in the Amazon Simple Storage Service User Guide.

4. Choose **Attach**.
5. In the **Filter roles** search box, enter **ecsInstanceRole**.
6. Check the box to the left of the **ecsInstanceRole** role, and then choose **Attach policy**.

Required permissions for monitoring container instances

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

To create the ECS-CloudWatchLogs IAM policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy, JSON**.
4. Enter the following policy:

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

```
        ],
        "Resource": [
            "arn:aws:logs:*:*:*"
        ]
    ]
}
```

5. Choose **Review policy**.
6. On the **Review policy** page, enter ECS-CloudWatchLogs for the **Name** and choose **Create policy**.

To attach the ECS-CloudWatchLogs policy to ecsInstanceRole

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose **ecsInstanceRole**. If the role does not exist, follow the procedures in [Amazon ECS container instance IAM role \(p. 638\)](#) to create the role.
4. In the navigation pane, choose **Policies**.
5. Choose **ECS-CloudWatchLogs**.
6. Choose **Policy actions, Attach**.
7. To narrow the available policies to attach, for **Filter**, type **ecsInstance**.
8. Select the **ecsInstance** role and choose **Attach policy**.

ECS Anywhere IAM role

When registering an on-premise server or virtual machine (VM) to your cluster, the server or VM requires an IAM role to communicate with AWS APIs. You only need to create this IAM role once per AWS account.

Checking for the ECS Anywhere (ecsAnywhereRole) in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. In the search box, enter **ecsAnywhereRole**. If the role does exist, choose the role to view the attached policies.
4. On the **Permissions** tab, verify that the **AmazonEC2ContainerServiceforEC2Role** and **AmazonSSMManagedInstanceCore** is attached to the role.
 - a. Choose **Add Permissions, Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, enter **AmazonEC2ContainerServiceforEC2Role** and **AmazonSSMManagedInstanceCore**.
 - c. Check the box to the left of the **AmazonEC2ContainerServiceforEC2Role** and **AmazonSSMManagedInstanceCore** policy, and then choose **Attach policy**.
5. Choose **Trust relationships**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "sts:AssumeRole"
        }
    ]
}
```

```
        "Sid": "",  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "ssm.amazonaws.com"  
        },  
        "Action": "sts:AssumeRole"  
    }  
]  
}
```

Creating the ECS Anywhere (ecsAnywhereRole) role

To create the ecsAnywhereRole (AWS Management Console)

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. Choose the **AWS service** role type, and then choose **Elastic Container Service**.
4. Choose the **EC2 Role for Elastic Container Service** use case and then **Next: Permissions**.
5. In the **Attached permissions policy** section, select **AmazonEC2ContainerServiceforEC2Role** and then choose **Next: Review**.
6. For **Role name**, enter **ecsAnywhereRole** and optionally you can enter a description, for example **Allows on-premises servers or virtual machine in an ECS cluster to access ECS**.
7. Review your role information and then choose **Create role** to finish.
8. Choose the **ecsAnywhereRole** role you just created.
9. On the **Permissions** tab, choose **Attach policies**.
10. In the **Filter** box, enter **AmazonSSMManagedInstanceCore** to narrow the available policies to attach.
11. Check the box to the left of the **AmazonSSMManagedInstanceCore** policy and choose **Attach policy**.
12. On the **Trust relationships** tab, choose **Edit trust relationship**.
13. Change the trust relationship so that it contains the following policy and then choose **Update Trust Policy**.

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

To create the ecsAnywhereRole role (AWS CLI)

1. Create a local file named **ssm-trust-policy.json** with the following contents.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "ssm.amazonaws.com"  
        },  
        "Action": "sts:AssumeRole"  
    }  
}
```

```

        "Principal": {"Service": [
            "ssm.amazonaws.com"
        ]},
        "Action": "sts:AssumeRole"
    }
}

```

2. Create the role.

```
aws iam create-role --role-name ecsAnywhereRole --assume-role-policy-document file://ssm-trust-policy.json
```

3. Attach the AWS managed policies.

```

aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role

```

Amazon ECS CodeDeploy IAM Role

Before you can use the CodeDeploy blue/green deployment type with Amazon ECS, the CodeDeploy service needs permissions to update your Amazon ECS service on your behalf. These permissions are provided by the CodeDeploy IAM role (`ecsCodeDeployRole`).

Note

Users also require permissions to use CodeDeploy; these permissions are described in [Blue/green deployment required IAM permissions \(p. 480\)](#).

There are two managed policies provided. The `AWSCodeDeployRoleForECS` policy, shown below, gives CodeDeploy permission to update any resource using the associated action.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DescribeServices",
                "ecs>CreateTaskSet",
                "ecs:UpdateServicePrimaryTaskSet",
                "ecs>DeleteTaskSet",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:ModifyRule",
                "lambda:InvokeFunction",
                "cloudwatch:DescribeAlarms",
                "sns:Publish",
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "AWS:CloudWatchLogsLogGroupArn": "arn:aws:logs:::log-group:"
                }
            }
        }
    ]
}
```

```
        "Condition": {
            "Stringlike": {
                "iam:PassedToService": [
                    "ecs-tasks.amazonaws.com"
                ]
            }
        }
    }
}
```

The `AWSCodeDeployRoleForECSLimited` policy, shown below, gives CodeDeploy more limited permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DescribeServices",
                "ecs>CreateTaskSet",
                "ecs:UpdateServicePrimaryTaskSet",
                "ecs:DeleteTaskSet",
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:CodeDeployTopic_*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:ModifyRule"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:*:*:function:CodeDeployHook_*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        }
    ]
}
```

```
        "Effect": "Allow"
    },
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::*:role/ecsTaskExecutionRole",
        "arn:aws:iam::*:role/ECSTaskExecution*"
    ],
    "Condition": {
        "Stringlike": {
            "iam:PassedToService": [
                "ecs-tasks.amazonaws.com"
            ]
        }
    }
}
]
```

Creating the CodeDeploy AWSCodeDeployRoleForECS role

To create an IAM role for CodeDeploy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select type of trusted entity** section, choose **AWS service**.
4. For **Choose the service that will use this role**, choose **CodeDeploy**.
5. For **Select your use case**, choose **CodeDeploy - ECS**, **Next**.
6. In the **Attach permissions policy** section, do the following:
 - a. Search for **AWSCodeDeployRoleForECS**, then select the policy.
 - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
 - c. Choose **Next**.
7. Under **Role details**, do the following:
 - a. For **Role name**, enter **ecsCodeDeployRole**, and enter an optional description.
 - b. For **Add tags (optional)**, enter any custom tags to associate with the policy .
8. Choose **Create role**.

To add the required permissions to the Amazon ECS CodeDeploy IAM role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Search the list of roles for **ecsCodeDeployRole**. If the role does not exist, use the procedure above to create the role. If the role does exist, select the role to view the attached policies.
3. In the **Permissions policies** section, verify that either the **AWSCodeDeployRoleForECS** or **AWSCodeDeployRoleForECSLimited** managed policy is attached to the role. If the policy is attached, your Amazon ECS CodeDeploy service role is properly configured. If not, follow the substeps below to attach the policy.
 - a. Choose **Add Permissions, Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, type **AWSCodeDeployRoleForECS** or **AWSCodeDeployRoleForECSLimited**.

- c. Check the box to the left of the AWS managed policy, and then choose **Attach policy**.
4. Choose **Trust relationships**.
5. 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, choose **Edit trust policy**, copy the policy into the **Policy Document** window, and then choose **Update policy**.

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

Adding permissions for blue/green deployments

If the tasks in your Amazon ECS service using the blue/green deployment type require the use of the task execution role or a task role override, then you must add the `iam:PassRole` permission for each task execution role or task role override to the CodeDeploy IAM role as a policy. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#) and [Task IAM role \(p. 631\)](#).

Use the following procedure to create the policy

To use the JSON policy editor to create a policy

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 on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": [  
                "arn:aws:iam::<aws_account_id>:role/  
                <ecsTaskExecutionRole_or_TaskRole_name>"  
            ]  
        }  
    ]  
}
```

6. Choose **Next**.

Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

After you create the policy, attach the policy to the `AWSCodeDeployRoleForECS` or `AWSCodeDeployRoleForECSLimited` role. For information about how to attach the policy to the role, see [Modifying a role permission policy](#) in the *AWS Identity and Access Management User Guide*.

Amazon ECS 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 \(p. 445\)](#).

The `AmazonEC2ContainerServiceEventsRole` policy is shown below.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "ecs:RunTask"  
      ],  
      "Resource": [  
        "*"  
      ]  
    },  
    {  
      "Effect": "Allow",  
      "Action": "iam:PassRole",  
      "Resource": [  
        "*"  
      ],  
      "Condition": {  
        "StringLike": {  
          "iam:PassedToService": "ecs-tasks.amazonaws.com"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "ecs:TagResource",  
      "Resource": "*",  
      "Condition": {  
        "StringEquals": {  
          "ecs>CreateAction": [  
            "RunTask"  
          ]  
        }  
      }  
    }  
  ]  
}
```

```
    ]  
}
```

If your scheduled tasks require the use of the task execution role, a task role, or a task role override, then you must add `iam:PassRole` permissions for each task execution role, task role, or task role override to the CloudWatch Events IAM role. For more information about the task execution role, see [Amazon ECS task execution IAM role \(p. 626\)](#).

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_or_TaskRole_name>"  
            ]  
        }  
    ]  
}
```

Checking for the Amazon ECS CloudWatch Events (ecsEventsRole) in the IAM console

You can manually create the role and attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. Use the following procedure to check and see if your account already has the Amazon ECS container instance IAM role and to attach the managed IAM policy if needed.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. In the search box, enter `ecsEventsRole`. If the role does exist, choose the role to view the attached policies.
4. On the **Permissions** tab, verify that the **AmazonEC2ContainerServiceEventsRole** is attached to the role.
 - a. Choose **Add Permissions, Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, enter **AmazonEC2ContainerServiceEventsRole**.
 - c. Check the box to the left of the **AmazonEC2ContainerServiceEventsRole** policy, and then choose **Attach policy**.
5. Choose **Trust relationships**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update policy**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": "  
            "Action": "sts:AssumeRole",  
            "Resource": "  
        }  
    ]  
}
```

```
    "Principal": {  
        "Service": "events.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
}  
]  
}
```

Creating the Amazon ECS CloudWatch Events (ecsEventsRole) role

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**, **Create role**.
3. In the **Trusted entity type** section, choose **AWS service, Elastic Container Service**.
4. For **Use case**, choose **Elastic Container Service Task**, and then choose **Next**.
5. In the **Attach permissions policy** section, do the following:
 - a. In the search box, enter **AmazonEC2ContainerServiceEventsRole**, and then select the policy.
 - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
 - c. Choose **Next**.
6. Under **Role details**, do the following:
 - a. For **Role name**, enter **ecsEventsRole**.
 - b. For **Add tags (optional)**, enter any custom tags to associate with the policy.
7. Choose **Create role**.
8. Search the list of roles for **ecsEventsRole** and select the role.
9. Replace the existing trust relationship with the following text. On the **Trust relationships** tab, choose **Edit trust policy**, copy the policy into the **Policy Document** window, and then choose **Update policy**.

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

Attaching a policy to the ecsEventsRole 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_or_TaskRole_name>"  
            ]  
        }  
    ]  
}
```

4. For **Name**, enter **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 then select the role to view the attached policies.
7. Choose **Attach policy**.
8. In the **Attach policy** section, select the **AmazonECSEventsTaskExecutionRole** policy, and then choose **Attach policy**.

Grant permission to tag resources on creation

The following tag-on create Amazon ECS API actions allow you to specify tags when you create the resource. If tags are specified in the resource-creating action, AWS performs additional authorization to verify that the correct permissions are assigned to create tags.

- `CreateCapacityProvider`
- `CreateCluster`
- `CreateService`
- `CreateTaskSet`
- `RegisterContainerInstance`
- `RegisterTaskDefinition`
- `RunTask`
- `StartTask`

You can use resource tags to implement attribute-based control (ABAC). For more information, see [the section called “Control access to Amazon ECS resources using resource tags” \(p. 653\)](#) and [the section called “Tagging your resources” \(p. 529\)](#).

To allow tagging on creation, create or modify a policy to include both the permissions to use the action that creates the resource, such as `ecs:CreateCluster` or `ecs:RunTask` and the `ecs:TagResource` action.

The following example demonstrates a policy that allows users to create clusters and run tasks, but can only add tags during the cluster creation. Users are not permitted to tag any existing resources (they cannot call the `ecs:TagResource` action directly).

```
{
```

```

"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ecs:CreateCluster",
            "ecs:RunTask"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ecs:TagResource"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "ecs:CreateAction" : "CreateCluster"
            }
        }
    }
]
}

```

The `ecs:TagResource` action is only evaluated if tags are applied during the resource-creating action. Therefore, a user that has permissions to create a resource (assuming there are no tagging conditions) does not require permissions to use the `ecs:TagResource` action if no tags are specified in the request. However, if the user attempts to create a resource with tags, the request fails if the user does not have permissions to use the `ecs:TagResource` action.

Control access to specific tags

You can use additional conditions in the `Condition` element of your IAM policies to control the tag keys and values that can be applied to resources.

The following condition keys can be used with the examples in the preceding section:

- `aws:RequestTag`: To indicate that a particular tag key or tag key and value must be present in a request. Other tags can also be specified in the request.
- Use with the `StringEquals` condition operator to enforce a specific tag key and value combination, for example, to enforce the tag `cost-center=cc123`:

```
"StringEquals": { "aws:RequestTag/cost-center": "cc123" }
```

- Use with the `StringLike` condition operator to enforce a specific tag key in the request; for example, to enforce the tag key `purpose`:

```
"StringLike": { "aws:RequestTag/purpose": "*" }
```

- `aws:TagKeys`: To enforce the tag keys that are used in the request.
- Use with the `ForAllValues` modifier to enforce specific tag keys if they are provided in the request (if tags are specified in the request, only specific tag keys are allowed; no other tags are allowed). For example, the tag keys `environment` or `cost-center` are allowed:

```
"ForAllValues:StringEquals": { "aws:TagKeys": ["environment", "cost-center"] }
```

- Use with the `ForAnyValue` modifier to enforce the presence of at least one of the specified tag keys in the request. For example, at least one of the tag keys `environment` or `webserver` must be present in the request:

```
"ForAnyValue:StringEquals": { "aws:TagKeys": ["environment", "webserver"] }
```

These condition keys can be applied to resource-creating actions that support tagging, as well as the `ecs:TagResource` action. To learn whether an Amazon ECS API action supports tagging, see [Actions, resources, and condition keys for Amazon ECS](#).

To force users to specify tags when they create a resource, you must use the `aws:RequestTag` condition key or the `aws:TagKeys` condition key with the `ForAnyValue` modifier on the resource-creating action. The `ecs:TagResource` action is not evaluated if a user does not specify tags for the resource-creating action.

For conditions, the condition key is not case-sensitive and the condition value is case-sensitive. Therefore, to enforce the case-sensitivity of a tag key, use the `aws:TagKeys` condition key, where the tag key is specified as a value in the condition.

For more information about multi-value conditions, see [Creating a Condition That Tests Multiple Key Values](#) in the *IAM User Guide*.

Control access to Amazon ECS resources using resource tags

When you create an IAM policy that grants users permission to use Amazon ECS resources, you can include tag information in the `Condition` element of the policy to control access based on tags. This is known as attribute-based access control (ABAC). ABAC provides better control over which resources a user can modify, use, or delete. For more information, see [What is ABAC for AWS?](#)

For example, you can create a policy that allows users to delete a cluster, but denies the action if the cluster has the tag `environment=production`. To do this, you use the `aws:ResourceTag` condition key to allow or deny access to the resource based on the tags that are attached to the resource.

```
"StringEquals": { "aws:ResourceTag/environment": "production" }
```

To learn whether an Amazon ECS API action supports controlling access using the `aws:ResourceTag` condition key, see [Actions, resources, and condition keys for Amazon ECS](#). Note that the `Describe` actions do not support resource-level permissions, so you must specify them in a separate statement without conditions.

For example IAM policies, see [Example policies \(p. 653\)](#).

If you allow or deny users access to resources based on tags, you must consider explicitly denying users the ability to add those tags to or remove them from the same resources. Otherwise, it's possible for a user to circumvent your restrictions and gain access to a resource by modifying its tags.

Example policies

You can use IAM policies to grant users permissions to view and work with specific resources in the Amazon ECS console. You can use the example policies in the previous section; however, they are designed for requests that are made with the AWS CLI or an AWS SDK.

Example: Allow users to delete a cluster based on tags

The following policy allows users to delete clusters when the tag has a key/value pair of "Purpose/Testing".

```
{  
  "Version": "2012-10-17",
```

```
"Statement": [  
    {  
        "Action": [  
            "ecs:DeleteCluster"  
        ],  
        "Effect": "Allow",  
        "Resource": "arn:aws:ecs:region:account-id:cluster/*",  
        "Condition": {  
            "StringEquals": {  
                "aws:ResourceTag/Purpose": "Testing"  
            }  
        }  
    }  
]
```

Troubleshooting Amazon Elastic Container Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon ECS and IAM.

Topics

- [I am not authorized to perform an action in Amazon ECS \(p. 654\)](#)
- [I am not authorized to perform iam:PassRole \(p. 654\)](#)
- [I want to allow people outside of my AWS account to access my Amazon ECS resources \(p. 655\)](#)

I am not authorized to perform an action in Amazon ECS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional *my-example-widget* resource but doesn't have the fictional `ecs:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
ecs:GetWidget on resource: my-example-widget
```

In this case, the policy for the mateojackson user must be updated to allow access to the *my-example-widget* resource by using the `ecs:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon ECS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named *marymajor* tries to use the console to perform an action in Amazon ECS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon ECS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon ECS supports these features, see [How Amazon Elastic Container Service works with IAM \(p. 592\)](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

Logging and Monitoring in Amazon Elastic Container Service

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Elastic Container Service 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. AWS provides several tools for monitoring your Amazon ECS resources and responding to potential incidents:

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 \(p. 547\)](#).

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.

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. For more information, see [Using the awslogs log driver \(p. 161\)](#).

You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type. For more information, see [???](#) (p. 354).

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 events and EventBridge \(p. 558\)](#) in this guide and [What Is Amazon CloudWatch Events?](#) in the *Amazon CloudWatch Events User Guide*.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. 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 [Logging Amazon ECS API calls with AWS CloudTrail \(p. 584\)](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

For more information, see [AWS Trusted Advisor](#) in the *AWS Support User Guide*.

AWS Compute Optimizer

AWS Compute Optimizer is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads.

For more information, see [AWS Compute Optimizer recommendations \(p. 586\)](#).

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.

Compliance validation for Amazon Elastic Container Service

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.

- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

Note

Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](#).

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

AWS Fargate Federal Information Processing Standard (FIPS-140)

Federal Information Processing Standard (FIPS). FIPS-140 is a U.S. and Canadian government standard that specifies the security requirements for cryptographic modules that protect sensitive information. FIPS-140 defines a set of validated cryptography functions that can be used to encrypt data in transit and data at rest.

When you turn on FIPS-140 compliance, you can run workloads on Fargate in a manner that is compliant with FIPS-140. For more information about FIPS-140 compliance, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Considerations

Consider the following when using FIPS-140 compliance on Fargate:

- FIPS-140 compliance is only available in the AWS GovCloud (US) Regions.
- FIPS-140 compliance is turned off by default. You must turn it on.
- Your tasks must use the following configuration for FIPS-140 compliance:
 - The `operatingSystemFamily` must be `LINUX`.
 - The `cpuArchitecture` must be `X86_64`.
 - The Fargate platform version must be `1.4.0` or later.

Use FIPS on Fargate

Use the following procedure to use FIPS-140 compliance on Fargate.

1. Turn on FIPS-140 compliance. For more information, see [the section called “AWS Fargate Federal Information Processing Standard \(FIPS-140\) compliance” \(p. 247\)](#).
2. You can optionally use ECS Exec to run the following command to verify the FIPS-140 compliance status for a cluster.

Replace `my-cluster` with the name of your cluster.

A return value of "1" indicates that you are using FIPS.

```
aws ecs execute-command --cluster cluster-name \  
    --interactive \  
    --command "cat /proc/sys/crypto/fips_enabled"
```

Use CloudTrail for auditing

CloudTrail is turned on in your AWS account when you create the account. When API and console 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 which CloudTrail uses 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 [the section called "Logging Amazon ECS API calls with AWS CloudTrail" \(p. 584\)](#).

The following example shows a CloudTrail log entry that demonstrates the PutAccountSettingDefault API action:

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AIDAI5AJI5LXF5EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/jdoe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAIPWI0FC3EXAMPLE",  
    },  
    "eventTime": "2023-03-01T21:45:18Z",  
    "eventSource": "ecs.amazonaws.com",  
    "eventName": "PutAccountSettingDefault",  
    "awsRegion": "us-gov-east-1",  
    "sourceIPAddress": "52.94.133.131",  
    "userAgent": "aws-cli/2.9.8 Python/3.9.11 Windows/10 exe/AMD64 prompt/off command/  
ecs.put-account-setting",  
    "requestParameters": {  
        "name": "awsvpcTrunking",  
        "value": "enabled"  
    },  
    "responseElements": {  
        "setting": {  
            "name": "fargateFIPSMode",  
            "value": "enabled",  
            "principalArn": "arn:aws:iam::123456789012:user/jdoe"  
        }  
    },  
    "requestID": "acdc731e-e506-447c-965d-f5f75EXAMPLE",  
    "eventID": "6afced68-75cd-4d44-8076-0beEXAMPLE",  
    "readOnly": false,  
    "eventType": "AwsApiCall",  
    "managementEvent": true,  
    "recipientAccountId": "123456789012",  
    "eventCategory": "Management",  
    "tlsDetails": {  
        "tlsVersion": "TLSv1.2",  
    }  
}
```

```
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "ecs-fips.us-gov-east-1.amazonaws.com"
    }
```

Infrastructure Security in Amazon Elastic Container Service

As a managed service, Amazon Elastic Container Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection in Security Pillar AWS Well-Architected Framework](#).

You use AWS published API calls to access Amazon ECS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location. Amazon ECS supports resource-based access policies, which can include restrictions based on the source IP address, so make sure that the policies account for the IP address for the network location. You can also use Amazon ECS policies to control access from specific Amazon Virtual Private Cloud endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon ECS resource from only the specific VPC within the AWS network. For more information, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\)](#) (p. 659).

Amazon ECS interface VPC endpoints (AWS PrivateLink)

You can improve the security posture of your VPC by configuring Amazon ECS to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that allows you to privately access Amazon ECS APIs by using private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway.

For more information about AWS PrivateLink and VPC endpoints, see [VPC Endpoints](#) in the [Amazon VPC User Guide](#).

Considerations for Amazon ECS VPC endpoints

Considerations for Amazon ECS VPC endpoints for the Fargate launch type

Before you set up interface VPC endpoints for Amazon ECS, be aware of the following considerations:

- Tasks using the Fargate launch type don't require the interface VPC endpoints for Amazon ECS, but you might need interface VPC endpoints for Amazon ECR, Secrets Manager, or Amazon CloudWatch Logs described in the following points.

- To allow your tasks to pull private images from Amazon ECR, you must create the interface VPC endpoints for Amazon ECR. For more information, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

If your VPC doesn't have an internet gateway, you must create the gateway endpoint for Amazon S3. For more information, see [Create the Amazon S3 gateway endpoint](#) in the *Amazon Elastic Container Registry User Guide*. The interface endpoints for Amazon S3 can't be used with Amazon ECR.

Important

If you configure Amazon ECR to use an interface VPC endpoint, you can create a task execution role that includes condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see [Optional IAM permissions for Fargate tasks pulling Amazon ECR images over interface endpoints \(p. 630\)](#).

- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the awslogs log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#) in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to Amazon ECS. For example, assume that you want to run tasks in US East (N. Virginia). Then, you must create the Amazon ECS VPC endpoint in US East (N. Virginia). An Amazon ECS VPC endpoint created in any other region can't run tasks in US East (N. Virginia).
- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see [DHCP Options Sets](#) in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on TCP port 443 from the private subnet of the VPC.
- Service Connect management of the Envoy proxy uses the com.amazonaws.region.ecs-agent VPC endpoint. When you don't use the VPC endpoints, Service Connect management of the Envoy proxy uses the ecs-sc endpoint in that Region. For a list of the Amazon ECS endpoints in each Region, see [Amazon ECS endpoints and quotas](#).

Considerations for Amazon ECS VPC endpoints for the EC2 launch type

Before you set up interface VPC endpoints for Amazon ECS, be aware of the following considerations:

- Tasks using the EC2 launch type require that the container instances that they're launched on to run version 1.25.1 or later of the Amazon ECS container agent. For more information, see [Amazon ECS Linux container agent versions \(p. 362\)](#).
- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the awslogs log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#) in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to Amazon ECS. For example, assume that you want to run tasks in US East (N. Virginia). Then, you must create the Amazon ECS VPC endpoint in US East (N. Virginia). An Amazon ECS VPC endpoint created in any other region can't run tasks in US East (N. Virginia).

- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see [DHCP Options Sets](#) in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on TCP port 443 from the private subnet of the VPC.

Creating the VPC Endpoints for Amazon ECS

To create the VPC endpoint for the Amazon ECS service, use the [Creating an Interface Endpoint](#) procedure in the *Amazon VPC User Guide* to create the following endpoints. If you have existing container instances within your VPC, you should create the endpoints in the order that they're listed. If you plan on creating your container instances after your VPC endpoint is created, the order doesn't matter.

- com.amazonaws.*region*.ecs-agent
- com.amazonaws.*region*.ecs-telemetry
- com.amazonaws.*region*.ecs

Note

region represents the Region identifier for an AWS Region supported by Amazon ECS, such as us-east-2 for the US East (Ohio) Region.

If you have existing tasks that are using the EC2 launch type, after you have created the VPC endpoints, each container instance needs to pick up the new configuration. For this to happen, you must either reboot each container instance or restart the Amazon ECS container agent on each container instance. To restart the container agent, do the following.

To restart the Amazon ECS container agent

1. Log in to your container instance via SSH.
2. Stop the container agent.

```
sudo docker stop ecs-agent
```

3. Start the container agent.

```
sudo docker start ecs-agent
```

After you have created the VPC endpoints and restarted the Amazon ECS container agent on each container instance, all newly launched tasks pick up the new configuration.

Creating a VPC endpoint policy for Amazon ECS

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon ECS. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon ECS actions

The following is an example of an endpoint policy for Amazon ECS. When attached to an endpoint, this policy grants access to 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.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecs:CreateCluster",  
                "ecs>ListClusters"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

AWS services integrated with Amazon ECS

Amazon ECS works with other AWS services to provide additional solutions for your business challenges. This topic identifies services that either use Amazon ECS to add functionality, or services that Amazon ECS uses to perform tasks.

Contents

- [Using Amazon ECR with Amazon ECS \(p. 663\)](#)
 - [Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts \(p. 664\)](#)
 - [Amazon Elastic Container Service on AWS Outposts \(p. 665\)](#)
 - [Use App Mesh with Amazon ECS \(p. 668\)](#)
 - [AWS Deep Learning Containers on Amazon ECS \(p. 668\)](#)
 - [Using AWS User Notifications with Amazon ECS \(p. 669\)](#)

Using Amazon ECR with Amazon ECS

Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. Amazon ECR provides a secure, scalable, and reliable registry. Amazon ECR supports private Docker repositories with resource-based permissions using AWS IAM so that specific users or Amazon EC2 instances can access repositories and images. Developers can use the Docker CLI to author and manage images.

For more information on how to create repositories, push and pull images from Amazon ECR, and set access controls on your repositories, see the [Amazon Elastic Container Registry User Guide](#).

Using Amazon ECR Images with Amazon ECS

You can use your ECR images with Amazon ECS, but you need to satisfy the following prerequisites.

- Your container instances must be using at least version 1.7.0 of the Amazon ECS container agent. The latest version of the Amazon ECS-optimized AMI supports ECR images in task definitions. For more information, including the latest Amazon ECS-optimized AMI IDs, see [Amazon ECS Container Agent Versions](#) in the *Amazon Elastic Container Service Developer Guide*.
 - The Amazon ECS container instance role (`ecsInstanceRole`) that you use with your container instances must possess the following IAM policy permissions for Amazon ECR.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        }
    ]
}
```

```
        "ecr:GetAuthorizationToken"
    ],
    "Resource": "*"
}
]
```

If you use the `AmazonEC2ContainerServiceforEC2Role` managed policy for your container instances, then your role has the proper permissions. To check that your role supports Amazon ECR, see [Amazon ECS Container Instance IAM Role](#) in the *Amazon Elastic Container Service Developer Guide*.

- In your ECS task definitions, make sure that you are using the full `registry/repository:tag` naming for your ECR images. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`.

Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts

Amazon ECS supports workloads that use Local Zones, Wavelength Zones, and AWS Outposts for when low latency or local data processing is a requirement.

- You can use Local Zones are an extension of an AWS Region to place resources in multiple locations closer to your end users.
- You can use Wavelength Zones to build applications that deliver ultra-low latencies to 5G devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks.
- AWS Outposts brings native AWS services, infrastructure, and operating models to virtually any data center, co-location space, or on-premises facility.

Important

Amazon ECS on AWS Fargate workloads aren't supported in Local Zones, Wavelength Zones, or on AWS Outposts at this time.

For information about the differences between Local Zones, Wavelength Zones, and AWS Outposts , see [How should I think about when to use AWS Wavelength, AWS Local Zones, or AWS Outposts for applications requiring low latency or local data processing](#) in the AWS Wavelength FAQs.

Local Zones

A *Local Zone* is an extension of an AWS Region in close geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect. Resources that are created in a Local Zone can serve local users with low-latency communications. For more information, see [AWS Local Zones](#).

A Local Zone is represented by a Region code followed by an identifier that indicates the location (for example, `us-west-2-lax-1a`).

To use a Local Zone, you must opt in to the zone. After you opt in, you must create an Amazon VPC and subnet in the Local Zone.

You can launch Amazon EC2 instances, Amazon FSx file servers, and Application Load Balancers to use for your Amazon ECS clusters and tasks.

For more information, see [Local Zones](#) in the *Amazon EC2 User Guide for Linux Instances*.

Wavelength Zones

You can use AWS *Wavelength* to build applications that deliver ultra-low latency to mobile devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks. You can extend an Amazon Virtual Private Cloud to one or more Wavelength Zones. Then, you can use AWS resources such as Amazon EC2 instances to run applications that require ultra-low latency and a connection to AWS services in the Region.

A Wavelength Zone is an isolated Zone in the carrier location where the Wavelength infrastructure is deployed. Wavelength Zones are tied to an AWS Region. A Wavelength Zone is a logical extension of a Region, and is managed by the control plane in the Region.

A Wavelength Zone is represented by a Region code followed by an identifier that indicates the Wavelength Zone (for example, us-east-1-wl1-bos-wlz-1).

To use a Wavelength Zone, you must opt in to the Zone. After you opt in, you must create an Amazon VPC and subnet in the Wavelength Zone. Then, you can launch your Amazon EC2 instances in the Zone to use for your Amazon ECS clusters and tasks.

For more information, see [Get started with AWS Wavelength](#) in the *AWS Wavelength Developer Guide*.

Wavelength Zones aren't available in all AWS Regions. For information about the Regions that support Wavelength Zones, see [Available Wavelength Zones](#) in the *AWS Wavelength Developer Guide*.

AWS Outposts

AWS Outposts uses native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon ECS on AWS Outposts is suitable for low-latency workloads that require to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see [Amazon Elastic Container Service on AWS Outposts \(p. 665\)](#).

Amazon Elastic Container Service on AWS Outposts

AWS Outposts allows native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon ECS on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the [AWS Outposts User Guide](#).

Prerequisites

The following are the prerequisites for using Amazon ECS on AWS Outposts:

- You must have installed and configured an AWS Outposts in your on-premises data center.
- You must have a reliable network connection between your AWS Outposts and its AWS Region.
- You must have sufficient capacity of instance types available in your AWS Outposts.
- All Amazon ECS container instances must have Amazon ECS container agent 1.33.0 or later.

Limitations

The following are the limitations of using Amazon ECS on AWS Outposts:

- Amazon Elastic Container Registry, AWS Identity and Access Management, and Network Load Balancer run in the AWS Region, not on AWS Outposts. This will increase latencies between these services and the containers.
- AWS Fargate is not available on AWS Outposts.

Network Connectivity Considerations

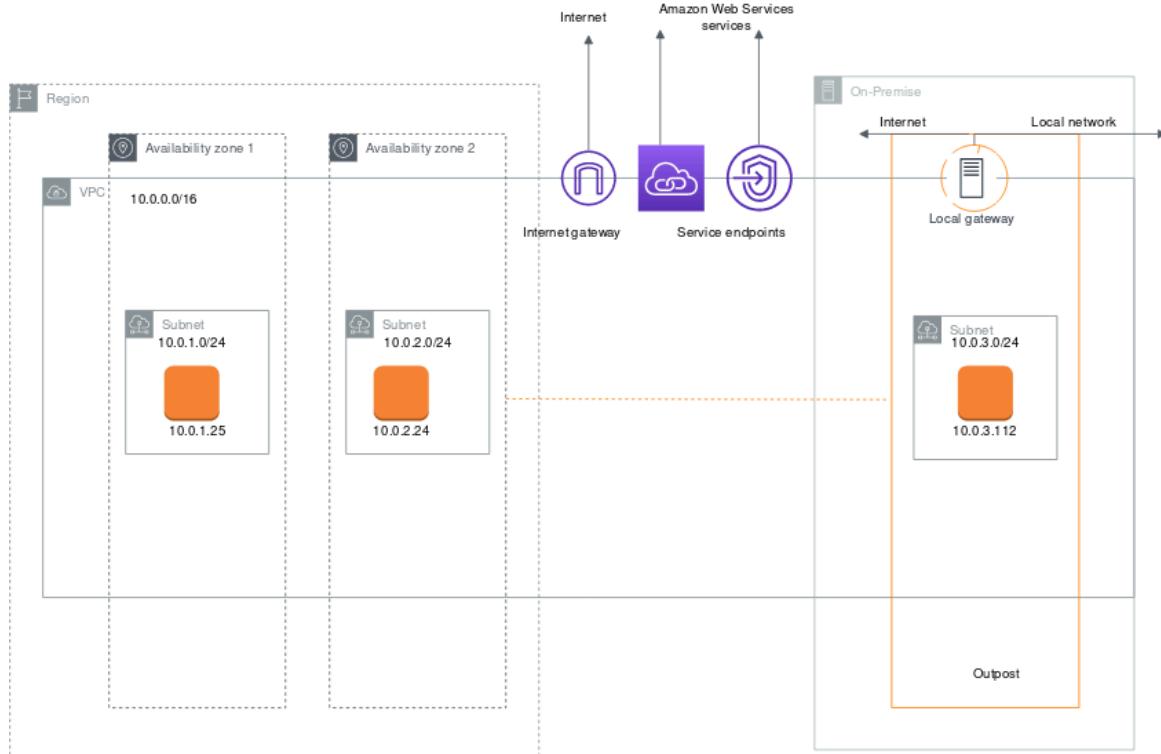
The following are network connectivity considerations for AWS Outposts:

- If network connectivity between your AWS Outposts and its AWS Region is lost, your clusters will continue to run. However, you cannot create new clusters or take new actions on existing clusters until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The CloudWatch Logs agent will be unable to update logs and event data.
- We recommend that you provide reliable, highly available, and low latency connectivity between your AWS Outposts and its AWS Region.

Creating an Amazon ECS Cluster on an AWS Outposts

Creating an Amazon ECS cluster on an AWS Outposts is similar to creating an Amazon ECS cluster in the AWS Cloud. When you create an Amazon ECS cluster on an AWS Outposts, you must specify a subnet associated with your AWS Outposts.

An AWS Outposts is an extension of an AWS Region, and you can extend an Amazon VPC in an account to span multiple Availability Zones and any associated AWS Outposts. When you configure your AWS Outposts, you associate a subnet with it to extend your Regional VPC environment to your on-premises facility. Instances on an AWS Outposts appear as part of your Regional VPC, similar to an Availability Zone with associated subnets.



AWS CLI

To create an Amazon ECS cluster on an AWS Outposts with the AWS CLI, specify a security group and a subnet to associate with your AWS Outposts.

To create a subnet associated with your AWS Outposts.

```
aws ec2 create-subnet \
--cidr-block 10.0.3.0/24 \
--vpc-id vpc-xxxxxxxx \
--outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-xxxxxxxxxxxxxx \
--availability-zone-id usw2-az1
```

The following example creates an Amazon ECS cluster on an AWS Outposts.

1. Create a role and policy with rights on AWS Outposts.

The `role-policy.json` file is the policy document that contains the effect and actions for resources. For information about the file format, see [PutRolePolicy](#) in the *IAM API Reference*

```
aws iam create-role --role-name ecsRole \
--assume-role-policy-document file://ecs-policy.json
aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
--policy-document file://role-policy.json
```

2. Create an IAM instance profile with rights on AWS Outposts.

```
aws iam create-instance-profile --instance-profile-name outpost
aws iam add-role-to-instance-profile --instance-profile-name outpost \
--role-name ecsRole
```

3. Create a VPC.

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

4. Create a security group for the container instances, specifying the proper CIDR range for the AWS Outposts. (This step is different for AWS Outposts.)

```
aws ec2 create-security-group --group-name MyOutpostSG
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
--port 22 --cidr 10.0.3.0/24
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
--port 80 --cidr 10.0.3.0/24
```

5. Create the Cluster.

6. Define the Amazon ECS container agent environment variables to launch the instance into the cluster created in the previous step and define any tags you want to add to help identify the cluster (for example, `Outpost` to indicate that the cluster is for an Outpost).

```
#!/bin/bash
cat << 'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={"environment": "Outpost"}
EOF
```

Note

In order to avoid delays caused by pulling container images from Amazon ECR in the Region, use image caches. To do this, each time a task is run, configure the Amazon ECS agent to default to using the cached image on the instance itself by setting `ECS_IMAGE_PULL_BEHAVIOR` to `prefer-cached`.

7. Create the container instance, specifying the VPC and subnet for the AWS Outposts where this instance should run and an instance type that is available on the AWS Outposts. (This step is different for AWS Outposts.)

The `userdata.txt` file contains the user data the instance can use to perform common automated configuration tasks and even run scripts after the instance starts. For information about the file for API calls, see [Run commands on your Linux instance at launch](#) in the *Amazon EC2 User Guide for Linux Instances*.

```
aws ec2 run-instances --count 1 --image-id ami-xxxxxxxxx --instance-type c5.large \
--key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxx \
--iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
--associate-public-ip-address --user-data file://userdata.txt
```

Note

This command is also used when adding additional instances to the cluster. Any containers deployed in the cluster will be placed on that specific AWS Outposts.

8. Register your task definition. Use the following command and substitute `ecs-task.json` with the name of your task definition.

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

9. Run the task or create the service.

Run the task

```
aws ecs run-task --cluster mycluster --count 1 --task-definition outpost-app:1
```

Create the service

```
aws ecs create-service --cluster mycluster --service-name outpost-service \
--task-definition outpost-app:1 --desired-count 1
```

Use App Mesh with Amazon ECS

App Mesh is a service mesh that makes it easy to monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility and helping to ensure high availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application. You can get started using App Mesh with Amazon ECS by completing the [Getting started with AWS App Mesh and Amazon ECS](#) tutorial in the AWS App Mesh User Guide. The tutorial recommends that you have existing services deployed to Amazon ECS that you want to use App Mesh with.

Note

This feature is not available for Window containers on Fargate.

AWS Deep Learning Containers on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for training and serving models in TensorFlow and Apache MXNet (Incubating) on Amazon ECS. Deep Learning Containers enable optimized environments with TensorFlow, NVIDIA CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries. Container images for Deep Learning Containers are available in Amazon ECR to reference in

Amazon ECS task definitions. You can use Deep Learning Containers along with Amazon Elastic Inference to lower your inference costs.

To get started using Deep Learning Containers without Elastic Inference on Amazon ECS, see [Deep Learning Containers on Amazon ECS](#) in the *AWS Deep Learning AMI Developer Guide*.

Deep Learning Containers with Elastic Inference on Amazon ECS

Note

Starting April 15, 2023, AWS will not onboard new customers to Amazon Elastic Inference (EI), and will help current customers migrate their workloads to options that offer better price and performance. After April 15, 2023, new customers will not be able to launch instances with Amazon EI accelerators in Amazon SageMaker, Amazon ECS, or Amazon EC2. However, customers who have used Amazon EI at least once during the past 30-day period are considered current customers and will be able to continue using the service.

AWS Deep Learning Containers provide a set of Docker images for serving models in TensorFlow and Apache MXNet (Incubating) that take advantage of Amazon Elastic Inference accelerators. Amazon ECS provides task definition parameters to attach Elastic Inference accelerators to your containers. When you specify an Elastic Inference accelerator type in your task definition, Amazon ECS manages the lifecycle of, and configuration for, the accelerator. The Amazon ECS service-linked role is required when using this feature. For more information about Elastic Inference accelerators, see [Amazon Elastic Inference Basics](#).

Important

Your Amazon ECS container instances require at least version 1.30.0 of the container agent. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

To get started using Deep Learning Containers with Elastic Inference on Amazon ECS, see [Deep Learning Containers with Elastic Inference on Amazon ECS](#) in the *Amazon Elastic Inference Developer Guide*.

Using AWS User Notifications with Amazon ECS

You can use [AWS User Notifications](#) to set up delivery channels to get notified about Amazon ECS events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, [AWS Chatbot](#) chat notifications, or [AWS Console Mobile Application](#) push notifications. You can also see notifications in the [Console Notifications Center](#). User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

Example

The following event pattern matches a task state change on the cluster named default.

```
{  
  "source": ["aws.ecs"],  
  "detail-type": ["ECS Task State Change"]  
  "detail": {  
    "clusterArn": [  
      "default"  
    ]  
  }  
}
```

Tutorials for Amazon ECS

The following tutorials show you how to perform common tasks when using Amazon ECS.

Topics

- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI \(p. 670\)](#)
- [Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI \(p. 678\)](#)
- [Tutorial: Creating a cluster with an EC2 task using the AWS CLI \(p. 683\)](#)
- [Tutorial: Using cluster auto scaling with the AWS Management Console and the Amazon ECS console \(p. 690\)](#)
- [Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets \(p. 693\)](#)
- [Tutorial: Creating a service using Service Discovery \(p. 697\)](#)
- [Tutorial: Creating a service using a blue/green deployment \(p. 704\)](#)
- [Tutorial: Listening for Amazon ECS CloudWatch Events \(p. 713\)](#)
- [Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events \(p. 715\)](#)
- [Tutorial: Using Amazon EFS file systems with Amazon ECS using the console \(p. 718\)](#)
- [Tutorial: Using FSx for Windows File Server file systems with Amazon ECS \(p. 723\)](#)
- [Tutorial: Deploying Fluent Bit on Amazon ECS for Windows containers \(p. 731\)](#)
- [Fargate AWS CLI capacity provider examples \(p. 741\)](#)
- [Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI \(p. 743\)](#)

Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Linux 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](#).

Topics

- [Prerequisites \(p. 670\)](#)
- [Step 1: Create a Cluster \(p. 671\)](#)
- [Step 2: Register a Linux Task Definition \(p. 671\)](#)
- [Step 3: List Task Definitions \(p. 672\)](#)
- [Step 4: Create a Service \(p. 673\)](#)
- [Step 5: List Services \(p. 673\)](#)
- [Step 6: Describe the Running Service \(p. 673\)](#)
- [Step 7: Test \(p. 675\)](#)
- [Step 8: Clean Up \(p. 678\)](#)

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 [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Amazon ECR Public so your task must have internet access. To give your task a route to the internet, use one of the following options.
 - Use a private subnet with a NAT gateway that has an elastic IP address.
 - Use a public subnet and assign a public IP address to the task.

For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).

For information about security groups and rules, see, [Default security groups for your VPCs](#) and [Example rules](#) in the *Amazon Virtual Private Cloud User Guide*.

- If you follow this tutorial using a private subnet, you can use Amazon ECS Exec to directly interact with your container and test the deployment. You will need to create a task IAM role to use ECS Exec. For more information on the task IAM role and other prerequisites, see [Using Amazon ECS Exec for debugging](#).
- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

Step 1: 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 Linux 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 using the httpd container image hosted on Docker Hub. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 83\)](#). For this tutorial, the taskRoleArn is only needed if you are deploying the task in a private subnet and wish to test the deployment. Replace the taskRoleArn with the IAM task role you created to use ECS Exec as mentioned in [Prerequisites \(p. 670\)](#).

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "taskRoleArn": "arn:aws:iam::aws_account_id:role/execCommandRole",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "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"
}
```

Save the task definition JSON as a file and pass it with the --cli-input-json file://[path_to_file.json](#) option.

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** command returns a description of the task definition after it completes its registration.

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"  
    ]  
}
```

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 with one instance of the sample-fargate:1 task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet. The `--enable-execute-command` option is needed to use Amazon ECS Exec.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --  
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-  
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-  
abcd1234]}" --enable-execute-command
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --  
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-  
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-  
abcd1234],assignPublicIp=ENABLED}"
```

The `create-service` command returns a description of the task definition after it completes its registration.

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-cluster/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
```

If successful, this will return a description of the service failures and services. For example, in the `services` section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and timestamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see [Service Event Messages](#). For more information about the service description, see [Describe Services](#).

```
{  
    "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,  
                    "lastUpdated": 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-b744699e6cfcc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
    "createdAt": 1510811364.691
}
],
"runningCount": 0,
"placementStrategy": []
},
],
"failures": []
}
]
}

```

Step 7: Test

Testing task deployed using public subnet

Describe the task in the service so that you can get the Elastic Network Interface (ENI) for the task.

First, get the task ARN.

```
aws ecs list-tasks --cluster fargate-cluster --service fargate-service
```

The output contains the task ARN.

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE"
    ]
}
```

Describe the task and locate the ENI ID. Use the task ARN for the tasks parameter.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-
east-1:123456789012:task/service/EXAMPLE
```

The attachment information is listed in the output.

```
{
    "tasks": [
        {

```

```
"attachments": [
  {
    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
    "type": "ElasticNetworkInterface",
    "status": "ATTACHED",
    "details": [
      {
        "name": "subnetId",
        "value": "subnetabcd1234"
      },
      {
        "name": "networkInterfaceId",
        "value": "eni-0fa40520aeEXAMPLE"
      }
    ]
  }
...]
```

Describe the ENI to get the public IP address.

```
aws ec2 describe-network-interfaces --network-interface-id eni-0fa40520aeEXAMPLE
```

The public IP address is in the output.

```
{
  "NetworkInterfaces": [
    {
      "Association": {
        "IpOwnerId": "amazon",
        "PublicDnsName": "ec2-34-229-42-222.compute-1.amazonaws.com",
        "PublicIp": "198.51.100.2"
      }
    }
...]
```

Enter the public IP address in your web browser and you should see a webpage that displays the **Amazon ECS** sample application.

Testing task deployed using private subnet

Describe the task and locate managedAgents to verify that the ExecuteCommandAgent is running. Note the privateIPv4Address for later use.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE
```

The managed agent information is listed in the output.

```
{
  "tasks": [
    {
      "attachments": [
        {
          "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
          "type": "ElasticNetworkInterface",
          "status": "ATTACHED",
          "details": [
            {
              "name": "subnetId",
```

```
        "value": "subnetabcd1234"
    },
{
    "name": "networkInterfaceId",
    "value": "eni-0fa40520aeEXAMPLE"
},
{
    "name": "privateIPv4Address",
    "value": "10.0.143.156"
}
],
],
...
"containers": [
{
...
"managedAgents": [
{
    "lastStartedAt": "2023-08-01T16:10:13.002000+00:00",
    "name": "ExecuteCommandAgent",
    "lastStatus": "RUNNING"
}
],
...
}
]
```

After verifying that the `ExecuteCommandAgent` is running, you can run the following command to run an interactive shell on the container in the task.

```
aws ecs execute-command --cluster fargate-cluster \
--task arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE \
--container fargate-app \
--interactive \
--command "/bin/sh"
```

After the interactive shell is running, run the following commands to install cURL.

```
apt update
```

```
apt install curl
```

After installing cURL, run the following command using the private IP address you obtained earlier.

```
curl 10.0.143.156
```

You should see the HTML equivalent of the **Amazon ECS** sample application webpage.

```
<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>
```

Step 8: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Windows 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](#).

Topics

- [Prerequisites \(p. 678\)](#)
- [Step 1: Create a Cluster \(p. 679\)](#)
- [Step 2: Register a Windows Task Definition \(p. 679\)](#)
- [Step 3: List task definitions \(p. 680\)](#)
- [Step 4: Create a service \(p. 680\)](#)
- [Step 5: List services \(p. 681\)](#)
- [Step 6: Describe the Running Service \(p. 681\)](#)
- [Step 7: Clean Up \(p. 683\)](#)

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 [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
 - Use a private subnet with a NAT gateway that has an elastic IP address.
 - Use a public subnet and assign a public IP address to the task.

For more information, see [the section called “Create a virtual private cloud” \(p. 12\)](#).

For information about security groups and rules, see, [Default security groups for your VPCs](#) and [Example rules](#) in the *Amazon Virtual Private Cloud User Guide*.

- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

Step 1: 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 Windows Task Definition

Before you can run a Windows task on your Amazon 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 web app. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 83\)](#).

```
{  
  "containerDefinitions": [  
    {  
      "command": [  
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<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>'; C:\\ServiceMonitor.exe w3svc"  
      ],  
      "entryPoint": [  
        "powershell",  
        "-Command"  
      ],  
      "essential": true,  
      "cpu": 2048,  
      "image": "microsoft/windows-server-2012r2-nano:latest",  
      "memory": 1024,  
      "portMappings": [  
        {  
          "hostPort": 80,  
          "containerPort": 80  
        }  
      ]  
    }  
  ]  
}
```

```

    "memory": 4096,
    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
    "name": "sample_windows_app",
    "portMappings": [
        {
            "hostPort": 80,
            "containerPort": 80,
            "protocol": "tcp"
        }
    ]
},
"memory": "4096",
"cpu": "2048",
"networkMode": "awsvpc",
"family": "windows-simple-iis-2019-core",
"executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
"runtimePlatform": {
    "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
},
"requiresCompatibilities": [
    "FARGATE"
]
}

```

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.

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` command returns a description of the task definition after it completes its registration.

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-windows:1"
    ]
}
```

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 with one instance of the `sample-fargate:1` task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --  
task-definition sample-fargate-windows:1 --desired-count 1 --launch-type "FARGATE" --  
network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234]}"
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --  
task-definition sample-fargate-windows:1 --desired-count 1 --launch-type "FARGATE" --  
network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

The **create-service** command returns a description of the task definition after it completes its registration.

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
```

If successful, this will return a description of the service failures and services. For example, in services section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see [Service Event Messages](#). For more information about the service description, see [Describe Services](#).

```
{  
    "services": [  
        {  
            "status": "ACTIVE",  
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-  
fargate-windows: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-windows: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-b744699e6fc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",

```

```
        "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
        "createdAt": 1510811364.691
    },
    "runningCount": 0,
    "placementStrategy": []
},
],
"failures": []
}
```

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Tutorial: Creating a cluster with an EC2 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](#).

Topics

- [Prerequisites \(p. 683\)](#)
- [Step 1: Create a Cluster \(p. 684\)](#)
- [Step 2: Launch an Instance with the Amazon ECS AMI \(p. 684\)](#)
- [Step 3: List Container Instances \(p. 684\)](#)
- [Step 4: Describe your Container Instance \(p. 685\)](#)
- [Step 5: Register a Task Definition \(p. 687\)](#)
- [Step 6: List Task Definitions \(p. 688\)](#)
- [Step 7: Run a Task \(p. 688\)](#)
- [Step 8: List Tasks \(p. 689\)](#)
- [Step 9: Describe the Running Task \(p. 689\)](#)

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 [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).
- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

Step 1: Create a Cluster

By default, your account receives a default cluster when you launch your first container instance.

Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` `cluster_name` option in the 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 MyCluster
```

Output:

```
{  
    "cluster": {  
        "clusterName": "MyCluster",  
        "status": "ACTIVE",  
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"  
    }  
}
```

Step 2: Launch an Instance with the Amazon ECS AMI

You must have an Amazon ECS container instance in your cluster before you can run tasks on it. If you do not have any container instances in your cluster, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#) for more information.

Step 3: List Container Instances

Within a few minutes of launching your container instance, the Amazon ECS agent registers the instance with your default cluster. You can list the container instances in a cluster by running the following command:

```
aws ecs list-container-instances --cluster default
```

Output:

```
{  
    "containerInstanceArns": [  
        "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"  
    ]  
}
```

Step 4: Describe your Container Instance

After you have the ARN or ID of a container instance, you can use the **describe-container-instances** command to get valuable information on the instance, such as remaining and registered CPU and memory resources.

```
aws ecs describe-container-instances --cluster default --container-instances container_instance_ID
```

Output:

```
{  
    "failures": [],  
    "containerInstances": [  
        {  
            "status": "ACTIVE",  
            "registeredResources": [  
                {  
                    "integerValue": 1024,  
                    "longValue": 0,  
                    "type": "INTEGER",  
                    "name": "CPU",  
                    "doubleValue": 0.0  
                },  
                {  
                    "integerValue": 995,  
                    "longValue": 0,  
                    "type": "INTEGER",  
                    "name": "MEMORY",  
                    "doubleValue": 0.0  
                },  
                {  
                    "name": "PORTS",  
                    "longValue": 0,  
                    "doubleValue": 0.0,  
                    "stringSetValue": [  
                        "22",  
                        "2376",  
                        "2375",  
                        "51678"  
                    ],  
                    "type": "STRINGSET",  
                    "integerValue": 0  
                },  
                {  
                    "name": "PORTS_UDP",  
                    "longValue": 0,  
                    "doubleValue": 0.0,  
                    "stringSetValue": [],  
                    "type": "STRINGSET",  
                    "integerValue": 0  
                }  
            ],  
            "ec2InstanceId": "instance_id",  
            "agentConnected": true,  
            "containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-instance/container_instance_ID",  
            "pendingTasksCount": 0,  
            "remainingResources": [  
                {  
                    "integerValue": 1024,  
                    "longValue": 0,  
                    "type": "INTEGER",  
                    "name": "CPU",  
                    "doubleValue": 0.0  
                },  
                {  
                    "integerValue": 995,  
                    "longValue": 0,  
                    "type": "INTEGER",  
                    "name": "MEMORY",  
                    "doubleValue": 0.0  
                }  
            ]  
        }  
    ]  
}
```

```

        "type": "INTEGER",
        "name": "CPU",
        "doubleValue": 0.0
    },
    {
        "integerValue": 995,
        "longValue": 0,
        "type": "INTEGER",
        "name": "MEMORY",
        "doubleValue": 0.0
    },
    {
        "name": "PORTS",
        "longValue": 0,
        "doubleValue": 0.0,
        "stringSetValue": [
            "22",
            "2376",
            "2375",
            "51678"
        ],
        "type": "STRINGSET",
        "integerValue": 0
    },
    {
        "name": "PORTS_UDP",
        "longValue": 0,
        "doubleValue": 0.0,
        "stringSetValue": [],
        "type": "STRINGSET",
        "integerValue": 0
    }
],
"runningTasksCount": 0,
"attributes": [
    {
        "name": "com.amazonaws.ecs.capability.privileged-container"
    },
    {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
    },
    {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
    },
    {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
    },
    {
        "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
    },
    {
        "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
    }
],
"versionInfo": {
    "agentVersion": "1.5.0",
    "agentHash": "b197edd",
    "dockerVersion": "DockerVersion: 1.7.1"
}
}
]
}

```

You can also find the Amazon EC2 instance ID that you can use to monitor the instance in the Amazon EC2 console or with the **aws ec2 describe-instances --instance-id *instance_id*** command.

Step 5: Register a Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses a busybox image from Docker Hub and simply sleeps for 360 seconds. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 83\)](#).

```
{  
    "containerDefinitions": [  
        {  
            "name": "sleep",  
            "image": "busybox",  
            "cpu": 10,  
            "command": [  
                "sleep",  
                "360"  
            ],  
            "memory": 10,  
            "essential": true  
        }  
    ],  
    "family": "sleep360"  
}
```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json` `file://path_to_file.json` option. Or, you can escape the quotation marks in the JSON and pass the JSON container definitions on the command line as in the below example. If you choose to pass the container definitions on the command line, your command additionally requires a `--family` parameter that is used to keep multiple versions of your task definition associated with each other.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/sleep360.json
```

To use a JSON string for container definitions:

```
aws ecs register-task-definition --family sleep360 --container-definitions "[{\\"name\\":  
    \\"sleep\\\", \\"image\\": \\"busybox\\\", \\"cpu\\": 10, \\"command\\": [\\"sleep\\\", \\"360\\\"], \\"memory\\": 10,  
    \\"essential\\": true}]"
```

The **register-task-definition** returns a description of the task definition after it completes its registration.

```
{  
    "taskDefinition": {  
        "volumes": [],  
        "taskDefinitionArn": "arn:aws:ec2:us-east-1:aws_account_id:task-definition/  
sleep360:1",  
        "containerDefinitions": [  
            {  
                "environment": [],  
                "name": "sleep",  
                "mountPoints": [],  
                "image": "busybox",  
                "cpu": 10,  
                "portMappings": [],  
                "command": [  
                    "sleep",  
                    "360"  
                ]  
            }  
        ]  
    }  
}
```

```

        "360"
    ],
    "memory": 10,
    "essential": true,
    "volumesFrom": []
}
],
"family": "sleep360",
"revision": 1
}
}

```

Step 6: List Task Definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the family and revision values that you can use together when calling **run-task** or **start-task**.

```
aws ecs list-task-definitions
```

Output:

```
{
  "taskDefinitionArns": [
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep300:1",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep300:2",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
  ]
}
```

Step 7: Run a Task

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can run the registered task in your cluster. For this example, you place a single instance of the `sleep360:1` task definition in your default cluster.

```
aws ecs run-task --cluster default --task-definition sleep360:1 --count 1
```

Output:

```
{
  "tasks": [
    {
      "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
      "overrides": {
        "containerOverrides": [
          {
            "name": "sleep"
          }
        ]
      },
      "lastStatus": "PENDING",
      "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID",
    }
  ]
}
```

```
"clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",
"desiredStatus": "RUNNING",
"taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
sleep360:1",
"containers": [
{
    "containerArn": "arn:aws:ecs:us-
east-1:aws_account_id:container/container_ID",
    "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
    "lastStatus": "PENDING",
    "name": "sleep"
}
]
}
```

Step 8: List Tasks

List the tasks for your cluster. You should see the task that you ran in the previous section. You can take the task ID or the full ARN that is returned from this command and use it to describe the task later.

```
aws ecs list-tasks --cluster default
```

Output:

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
    ]
}
```

Step 9: Describe the Running Task

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

```
aws ecs describe-tasks --cluster default --task task_ID
```

Output:

```
{
    "failures": [],
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
            "overrides": {
                "containerOverrides": [
                    {
                        "name": "sleep"
                    }
                ]
            },
            "lastStatus": "RUNNING",
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/container_instance_ID",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",
            "desiredStatus": "RUNNING",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
sleep360:1",
        }
    ]
}
```

```
"containers": [
    {
        "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/container_ID",
        "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
        "lastStatus": "RUNNING",
        "name": "sleep",
        "networkBindings": []
    }
]
```

Tutorial: Using cluster auto scaling with the AWS Management Console and the Amazon ECS console

This tutorial walks you through creating the resources for cluster auto scaling using the AWS Management Console. Where resources require a name, we will use the prefix `ConsoleTutorial` to ensure they all have unique names and to make them easy to locate.

Topics

- [Prerequisites \(p. 690\)](#)
- [Step 1: Create an Amazon ECS cluster \(p. 690\)](#)
- [Step 2: Register a task definition \(p. 691\)](#)
- [Step 3: Run a task \(p. 692\)](#)
- [Step 4: Verify \(p. 692\)](#)
- [Step 5: Clean up \(p. 693\)](#)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- The Amazon ECS container instance IAM role is created. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
- The Amazon ECS service-linked IAM role is created. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).
- The Auto Scaling service-linked IAM role is created. For more information, see [Service-Linked Roles for Amazon EC2 Auto Scaling](#) in the [Amazon EC2 Auto Scaling User Guide](#).
- You have a VPC and security group created to use. For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster.

Amazon ECS creates an Amazon EC2 Auto Scaling launch template and Auto Scaling group on your behalf as part of the AWS CloudFormation stack.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter **ConsoleTutorial-cluster**.
6. Under **Infrastructure**, select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider.
 - Under **Auto Scaling group (ASG)** . Select **Create new ASG**, and then provide the following details about the group:
 - For **Operating system/Architecture**, choose **Amazon Linux 2**.
 - For **EC2 instance type**, choose **t3.nano**.
 - For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group.
7. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

[Add a tag] Choose **Add tag** and do the following:

 - For **Key**, enter the key name.
 - For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
8. Choose **Create**.

Step 2: Register a task definition

Before you can run a task on your cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `amazonlinux` image from Docker Hub and simply sleeps. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 83\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the **JSON editor** box, paste the following contents.

```
{  
    "family": "ConsoleTutorial-taskdef",  
    "containerDefinitions": [  
        {  
            "name": "sleep",  
            "image": "amazonlinux:2",  
            "memory": 20,  
            "essential": true,  
            "command": [  
                "sh",  
                "-c",  
                "sleep infinity"  
            ]  
        }  
    ]  
}
```

```
        },
      ],
      "requiresCompatibilities": [
        "EC2"
      ]
    }
```

5. Choose **Create**.

Step 3: Run a task

After you have registered a task definition for your account, you can run a task in the cluster. For this tutorial, you run five instances of the `ConsoleTutorial-taskdef` task definition in your `ConsoleTutorial-cluster` cluster.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, choose `ConsoleTutorial-cluster`.
3. Under **Tasks**, choose **Run new task**.
4. In the **Environment** section, under **Compute options**, choose **Capacity provider strategy**.
5. Under **Deployment configuration**, for **Application type**, choose **Task**.
6. Choose `ConsoleTutorial-taskdef` from the **Family** dropdown list.
7. Under **Desired tasks**, enter 5.
8. Choose **Create**.

Step 4: Verify

At this point in the tutorial, you should have a cluster with five tasks running and an Auto Scaling group with a capacity provider. The capacity provider has Amazon ECS managed scaling enabled.

We can verify that everything is working properly by viewing the CloudWatch metrics, the Auto Scaling group settings, and finally the Amazon ECS cluster task count.

To view the CloudWatch metrics for your cluster

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. On the navigation bar at the top of the screen, select the Region.
3. On the navigation pane, under **Metrics**, choose **All metrics**.
4. On the **All metrics** page, under the **Browse** tab, choose AWS/ECS/ManagedScaling.
5. Choose **CapacityProviderName**, `ClusterName`.
6. Select the check box that corresponds to the `ConsoleTutorial-cluster ClusterName`.
7. Under the **Graphed metrics** tab, change **Period** to **30 seconds** and **Statistic** to **Maximum**.

The value displayed in the graph shows the target capacity value for the capacity provider. It should begin at 100, which was the target capacity percent we set. You should see it scale up to 200, which will trigger an alarm for the target tracking scaling policy. The alarm will then trigger the Auto Scaling group to scale out.

Use the following steps to view your Auto Scaling group details to confirm that the scale-out action occurred.

To verify the Auto Scaling group scaled out

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

2. On the navigation bar at the top of the screen, select the Region.
3. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
4. Choose the `ConsoleTutorial-cluster` Auto Scaling group created in this tutorial. View the value under **Desired capacity** and view the instances under the **Instance management** tab to confirm your group scaled out to two instances.

Use the following steps to view your Amazon ECS cluster to confirm that the Amazon EC2 instances were registered with the cluster and your tasks transitioned to a RUNNING status.

To verify the instances in the Auto Scaling group

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the navigation bar at the top of the screen, select the Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
5. On the **Tasks** tab, confirm you see five tasks in RUNNING status.

Step 5: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using. Deleting capacity providers and task definitions are not supported, but there is no cost associated with these resources.

To clean up the tutorial resources

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose `ConsoleTutorial-cluster`.
4. On the `ConsoleTutorial-cluster` page, choose the **Tasks** tab, and then choose **Stop, Stop all**.
5. In the navigation pane, choose **Clusters**.
6. On the **Clusters** page, choose `ConsoleTutorial-cluster`.
7. In the upper-right of the page, choose **Delete cluster**.
8. In the confirmation box, enter **delete ConsoleTutorial-cluster** and choose **Delete**.
9. Delete the Auto Scaling groups using the following steps.
 - a. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
 - b. On the navigation bar at the top of the screen, select the Region.
 - c. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
 - d. Select the `ConsoleTutorial-cluster` Auto Scaling group, then choose **Actions**.
 - e. From the **Actions** menu, choose **Delete**. Enter **delete** in the confirmation box and then choose **Delete**.

Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your container definition. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

The following tutorial shows how to create an Secrets Manager secret, reference the secret in an Amazon ECS task definition, and then verify it worked by querying the environment variable inside a container showing the contents of the secret.

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required IAM permissions to create the Secrets Manager and Amazon ECS resources described.

Step 1: Create an Secrets Manager Secret

You can use the Secrets Manager console to create a secret for your sensitive data. In this tutorial we will be creating a basic secret for storing a username and password to reference later in a container. For more information, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.

The **key/value pairs to be stored in this secret** is the environment variable value in your container at the end of the tutorial.

Save the **Secret ARN** to reference in your task execution IAM policy and task definition in later steps.

Step 2: Update Your Task Execution IAM Role

In order for Amazon ECS to retrieve the sensitive data from your Secrets Manager secret, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary Secrets Manager resources. If you have not already created your task execution IAM role, see [Amazon ECS task execution IAM role \(p. 626\)](#).

The following steps assume you already have the task execution IAM role created and properly configured.

To update your task execution IAM role

Use the IAM console to update your task execution role with the required permissions.

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` and select it.
4. Choose **Permissions, Add inline policy**.
5. Choose the **JSON** tab and specify the following JSON text, ensuring that you specify the full ARN of the Secrets Manager secret you created in step 1.

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

```
        ]
    }
}
```

6. Choose **Review policy**. For **Name** specify ECSSecretsTutorial, then choose **Create policy**.

Step 3: Create an Amazon ECS Task Definition

You can use the Amazon ECS console to create a task definition that references a Secrets Manager secret.

To create a task definition that specifies a secret

Use the IAM console to update your task execution role with the required permissions.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, edit your JSON file,
5. On the **Select launch type compatibility** page, choose **EC2** and choose **Next step**.
6. Choose **Configure via JSON** and enter the following task definition JSON text, ensuring that you specify the full ARN of the Secrets Manager secret you created in step 1 and the task execution IAM role you updated in step 2. Choose **Save**.

```
{
    "executionRoleArn": "arn:aws:iam::aws\_account\_id:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "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\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
"arn:aws:secretsmanager:region:aws\_account\_id:secret:username\_value",
                    "name": "username_value"
                }
            ],
            "memory": 300,
            "image": "httpd:2.4",
            "essential": true,
            "name": "ecs-secrets-container"
        }
    ]
}
```

```
        }
    ],
    "family": "ecs-secrets-tutorial"
}
```

7. Review the settings and then choose **Create**.

Step 4: Create an Amazon ECS Cluster

You can use the Amazon ECS console to create a cluster containing a container instance to run the task on. If you have an existing cluster with at least one container instance registered to it with the available resources to run one instance of the task definition created for this tutorial you can skip to the next step.

For this tutorial we will be creating a cluster with one `t2.micro` container instance using the Amazon ECS-optimized Amazon Linux 2 AMI.

For information about how to create a cluster for the EC2 launch type, see [the section called “Creating a cluster for the Amazon EC2 launch type using the console” \(p. 237\)](#).

Step 5: Run an Amazon ECS Task

You can use the Amazon ECS console to run a task using the task definition you created. For this tutorial we will be running a task using the EC2 launch type, using the cluster we created in the previous step.

For information about how to run a task, see [the section called “Running a standalone task using the Amazon ECS console” \(p. 430\)](#).

Step 6: Verify

You can verify all of the steps were completed successfully and the environment variable was created properly in your container using the following steps.

To verify that the environment variable was created

1. Find the public IP or DNS address for your container instance.
 - a. Open the console at <https://console.aws.amazon.com/ecs/v2>.
 - b. In the navigation pane, choose **Clusters**, and then choose the cluster you created.
 - c. Choose **Infrastructure**, and then choose the container instance.
 - d. Record the **Public IP** or **Public DNS** for your instance.
2. If you are using a macOS or Linux computer, connect to your instance with the following command, substituting the path to your private key and the public address for your instance:

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

For more information about using a Windows computer, see [Connecting to Your Linux Instance from Windows Using PuTTY](#) in the *Amazon EC2 User Guide for Linux Instances*.

Important

For more information about any issues while connecting to your instance, see [Troubleshooting Connecting to Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

3. List the containers running on the instance. Note the container ID for `ecs-secrets-tutorial` container.

```
docker ps
```

4. Connect to the `ecs-secrets-tutorial` container using the container ID from the output of the previous step.

```
docker exec -it container_ID /bin/bash
```

5. Use the `echo` command to print the value of the environment variable.

```
echo $username_value
```

If the tutorial was successful, you should see the following output:

```
password_value
```

Note

Alternatively, you can list all environment variables in your container using the `env` (or `printenv`) command.

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

To clean up the resources

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, select the cluster to delete.
4. In the upper-right of the page, choose **Delete Cluster**.

A message is displayed when you did not delete all the resources associated with the cluster.

5. In the confirmation box, enter **delete *cluster name***.
6. Open the IAM console at <https://console.aws.amazon.com/iam/>.
7. In the navigation pane, choose **Roles**.
8. Search the list of roles for `ecsTaskExecutionRole` and select it.
9. Choose **Permissions**, then choose the X next to **ECSecretsTutorial**. Choose **Remove**.
10. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
11. Select the `username_value` secret you created and choose **Actions, Delete secret**.

Tutorial: Creating a service using Service Discovery

Service discovery is now integrated into the Create Service wizard in the Amazon ECS console. For more information, see [Creating an Amazon ECS service in the classic console \(p. 899\)](#).

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

For a list of AWS Regions that support service discovery, see [Service discovery \(p. 522\)](#).

For information about the Regions that support Fargate, see [the section called “AWS Fargate Regions” \(p. 540\)](#).

Prerequisites

Before you start this tutorial, make sure that the following prerequisites are met:

- The latest version of the AWS CLI is installed and configured. For more information, see [Installing the AWS Command Line Interface](#).
- The steps described in [Set up to use Amazon ECS \(p. 10\)](#) are complete.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have created at least one VPC and one security group. For more information, see [the section called “Create a virtual private cloud” \(p. 12\)](#).

Step 1: Create the Service Discovery resources in AWS Cloud Map

Follow these steps to create your service discovery namespace and service discovery service:

1. Create a private Cloud Map service discovery namespace. This example creates a namespace that's called `tutorial`. Replace `vpc-abcd1234` with the ID of one of your existing VPCs.

```
aws servicediscovery create-private-dns-namespace \
    --name tutorial \
    --vpc vpc-abcd1234
```

The output of this command is as follows.

```
{  
    "OperationId": "h2qe3s6dxfvv7riu6lfy2f6c3jlf4-je6chs2e"  
}
```

2. Using the `OperationId` from the output of the previous step, verify that the private namespace was created successfully. Make note of the namespace ID because you use it in subsequent commands.

```
aws servicediscovery get-operation \
    --operation-id h2qe3s6dxfvv7riu6lfy2f6c3jlf4-je6chs2e
```

The output is as follows.

```
{  
    "Operation": {  
        "Id": "h2qe3s6dxfvv7riu6lfy2f6c3jlf4-je6chs2e",  
        "Type": "CREATE_NAMESPACE",  
        "Status": "SUCCESS",  
        "CreateDate": 1519777852.502,  
        "UpdateDate": 1519777856.086,  
        "Targets": {  
            "NAMESPACE": "ns-uejictsjen2i4eeg"  
        }  
    }  
}
```

3. Using the NAMESPACE ID from the output of the previous step, create a service discovery service. This example creates a service named myapplication. Make note of the service ID and ARN because you use them 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
```

The output is as follows.

```
{
    "Service": {
        "Id": "srv-utcrrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

Step 2: Create the Amazon ECS resources

Follow these steps to create your Amazon ECS cluster, task definition, and service:

1. Create an Amazon ECS cluster. This example creates a cluster that's named tutorial.

```
aws ecs create-cluster \
    --cluster-name tutorial
```

2. Register a task definition that's compatible with Fargate and uses the awsvpc network mode. Follow these steps:

- a. Create a file that's 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"
}
}

```

- b. Register the task definition using `fargate-task.json`.

```
aws ecs register-task-definition \
--cli-input-json file://fargate-task.json
```

3. Create an ECS service by following these steps:

- a. Create a file that's named `ecs-service-discovery.json` with the contents of the ECS service that you're creating. This example uses the task definition that was created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

When you create the ECS service, specify the Fargate launch type, and the LATEST platform version that supports service discovery. When the service discovery service is created in AWS Cloud Map, `registryArn` is the ARN returned. The `securityGroups` and `subnets` must belong to the VPC that's used to create the Cloud Map namespace. You can obtain the security group and subnet IDs from the Amazon VPC Console.

```

{
    "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": "LATEST",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [ "sg-abcd1234" ],
            "subnets": [ "subnet-abcd1234" ]
        }
    },
}

```

```
        "desiredCount": 1  
    }
```

- b. Create your ECS service using `ecs-service-discovery.json`.

```
aws ecs create-service \  
    --cli-input-json file://ecs-service-discovery.json
```

Step 3: Verify Service Discovery in AWS Cloud Map

You can verify that everything is created properly by querying your service discovery information. After service discovery is configured, you can either use AWS Cloud Map API operations, or call dig from an instance within your VPC. Follow these steps:

1. Using the service discovery service ID, list the service discovery instances. Make note of the instance ID (marked in bold) for resource cleanup.

```
aws servicediscovery list-instances \  
    --service-id srv-utcxh6wavdkggqtk
```

The output is as follows.

```
{  
    "Instances": [  
        {  
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",  
            "Attributes": {  
                "AWS_INSTANCE_IPV4": "172.31.87.2"  
                "AWS_INSTANCE_PORT": "80",  
                "AVAILABILITY_ZONE": "us-east-1a",  
                "REGION": "us-east-1",  
                "ECS_SERVICE_NAME": "ecs-service-discovery",  
                "ECS_CLUSTER_NAME": "tutorial",  
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"  
            }  
        }  
    ]  
}
```

2. Use the service discovery namespace, service, and additional parameters such as ECS cluster name to query details about the service discovery instances.

```
aws servicediscovery discover-instances \  
    --namespace-name tutorial \  
    --service-name myapplication \  
    --query-parameters ECS_CLUSTER_NAME=tutorial
```

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

- a. Using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID.

```
aws servicediscovery \  
    get-namespace --id ns-uejectsjen2i4eeg
```

The output is as follows.

```
{
    "Namespace": {
        "Id": "ns-uejectsjen2i4eeg",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-uejectsjen2i4eeg",
        "Name": "tutorial",
        "Type": "DNS_PRIVATE",
        "Properties": {
            "DnsProperties": {
                "HostedZoneId": "Z35JQ4ZFDRYPLV"
            }
        },
        "CreateDate": 1519777852.502,
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
    }
}
```

- b. Using the Route 53 hosted zone ID from the previous step (see the text in bold), get the resource record set for the hosted zone.

```
aws route53 list-resource-record-sets \
--hosted-zone-id Z35JQ4ZFDRYPLV
```

4. You can also query the DNS from an instance within your VPC using dig.

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

Step 4: Clean up

When you're finished with this tutorial, clean up the associated resources to avoid incurring charges for unused resources. Follow these steps:

1. Deregister the service discovery service instances using the service ID and instance ID that you noted previously.

```
aws servicediscovery deregister-instance \
--service-id srv-utcrh6wavdkggqtk \
--instance-id 16becc26-8558-4af1-9fdb-f81be062a266
```

The output is as follows.

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

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

```
aws servicediscovery get-operation \
--operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv
```

```
{
    "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-utc1h6wavdkggqtk"
        ]
    }
}

```

3. Delete the service discovery service using the service ID.

```
aws servicediscovery delete-service \  
--id srv-utc1h6wavdkggqtk
```

4. Delete the service discovery namespace using the namespace ID.

```
aws servicediscovery delete-namespace \  
--id ns-uejictsjen2i4eeg
```

The output is as follows.

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

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

```
aws servicediscovery get-operation \  
--operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj
```

The output is as follows.

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": [
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW30"
        ]
    }
}
```

6. Update the desired count for the Amazon ECS service to *0*. You must do this to delete the service in the next step.

```
aws ecs update-service \  
--cluster tutorial \  
--service ecs-service-discovery \  
--desired-count 0
```

7. Delete the Amazon ECS service.

```
aws ecs delete-service \  

```

```
--cluster tutorial \
--service ecs-service-discovery
```

8. Delete the Amazon ECS cluster.

```
aws ecs delete-cluster \
--cluster tutorial
```

Tutorial: Creating a service using a blue/green deployment

Amazon ECS has integrated blue/green deployments into the Create Service wizard on the classic Amazon ECS console. For more information, see [Creating an Amazon ECS service in the classic console \(p. 899\)](#).

The following tutorial shows how to create an Amazon ECS service containing a Fargate task that uses the blue/green deployment type with the AWS CLI.

Note

Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see [Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation](#) in the *AWS CloudFormation User Guide*.

Prerequisites

This tutorial assumes that you have completed the following prerequisites:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading the AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [the section called “Create a virtual private cloud” \(p. 12\)](#).
- The Amazon ECS CodeDeploy IAM role is created. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 644\)](#).

Step 1: Create an Application Load Balancer

Amazon ECS services using the blue/green deployment type require the use of either an Application Load Balancer or a Network Load Balancer. This tutorial uses an Application Load Balancer.

To create an Application Load Balancer

1. Use the [create-load-balancer](#) command to create an Application Load Balancer. Specify two subnets that aren't from the same Availability Zone as well as a security group.

```
aws elbv2 create-load-balancer \
--name blugreeng-alb \
--subnets subnet-abcd1234 subnet-abcd5678 \
```

```
--security-groups sg-abcd1234 \
--region us-east-1
```

The output includes the Amazon Resource Name (ARN) of the load balancer, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642
```

2. Use the [create-target-group](#) command to create a target group. This target group will route traffic to the original task set in your service.

```
aws elbv2 create-target-group \
--name bluegreentarget1 \
--protocol HTTP \
--port 80 \
--target-type ip \
--vpc-id vpc-abcd1234 \
--region us-east-1
```

The output includes the ARN of the target group, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4
```

3. Use the [create-listener](#) command to create a load balancer listener with a default rule that forwards requests to the target group.

```
aws elbv2 create-listener \
--load-balancer-arn
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
--protocol HTTP \
--port 80 \
--default-actions
Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4 \
--region us-east-1
```

The output includes the ARN of the listener, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

Step 2: Create an Amazon ECS cluster

Use the [create-cluster](#) command to create a cluster named `tutorial-bluegreen-cluster` to use.

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

The output includes the ARN of the cluster, with the following format:

```
arn:aws:ecs:region:aws_account_id:cluster/tutorial-bluegreen-cluster
```

Step 3: Register a task definition

Use the [register-task-definition](#) command to 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 following contents. Ensure that you use the ARN for your task execution role. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

```
{  
    "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",  
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"  
}
```

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

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

Step 4: Create an Amazon ECS service

Use the [create-service](#) command to create a service.

First, create a file named service-bluegreen.json with the following contents.

```
{
```

```

"cluster": "tutorial-bluegreen-cluster",
"serviceName": "service-bluegreen",
"taskDefinition": "tutorial-task-def",
"loadBalancers": [
    {
        "targetGroupArn":
            "arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
            bluegreentarget1/209a844cd01825a4",
        "containerName": "sample-app",
        "containerPort": 80
    }
],
"launchType": "FARGATE",
"schedulingStrategy": "REPLICA",
"deploymentController": {
    "type": "CODE_DEPLOY"
},
"platformVersion": "LATEST",
"networkConfiguration": {
    "awsvpcConfiguration": {
        "assignPublicIp": "ENABLED",
        "securityGroups": [ "sg-abcd1234" ],
        "subnets": [ "subnet-abcd1234", "subnet-abcd5678" ]
    }
},
"desiredCount": 1
}

```

Then create your service using the `service-bluegreen.json` file that you created.

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

The output includes the ARN of the service, with the following format:

```
arn:aws:ecs:region:aws_account_id:service/service-bluegreen
```

Step 5: Create the AWS CodeDeploy resources

Use the following steps to create your CodeDeploy application, the Application Load Balancer target group for the CodeDeploy deployment group, and the CodeDeploy deployment group.

To create CodeDeploy resources

1. Use the [create-application](#) command to create a CodeDeploy application. Specify the ECS compute platform.

```
aws deploy create-application \
    --application-name tutorial-bluegreen-app \
    --compute-platform ECS \
    --region us-east-1
```

The output includes the application ID, with the following format:

```
{
    "applicationId": "b8e9c1ef-3048-424e-9174-885d7dc9dc11"
}
```

2. Use the [create-target-group](#) command to create a second Application Load Balancer target group, which will be used when creating your CodeDeploy deployment group.

```
aws elbv2 create-target-group \
--name bluegreentarget2 \
--protocol HTTP \
--port 80 \
--target-type ip \
--vpc-id "vpc-0b6dd82c67d8012a1" \
--region us-east-1
```

The output includes the ARN for the target group, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc
```

3. Use the [create-deployment-group](#) command to create a CodeDeploy deployment group.

First, create a file named `tutorial-deployment-group.json` with the following contents. This example uses the resource that you created. For the `serviceRoleArn`, specify the ARN of your Amazon ECS CodeDeploy IAM role. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 644\)](#).

```
{
    "applicationName": "tutorial-bluegreen-app",
    "autoRollbackConfiguration": {
        "enabled": true,
        "events": [ "DEPLOYMENT_FAILURE" ]
    },
    "blueGreenDeploymentConfiguration": {
        "deploymentReadyOption": {
            "actionOnTimeout": "CONTINUE_DEPLOYMENT",
            "waitTimeInMinutes": 0
        },
        "terminateBlueInstancesOnDeploymentSuccess": {
            "action": "TERMINATE",
            "terminationWaitTimeInMinutes": 5
        }
    },
    "deploymentGroupName": "tutorial-bluegreen-dg",
    "deploymentStyle": {
        "deploymentOption": "WITH_TRAFFIC_CONTROL",
        "deploymentType": "BLUE_GREEN"
    },
    "loadBalancerInfo": {
        "targetGroupPairInfoList": [
            {
                "targetGroups": [
                    {
                        "name": "bluegreentarget1"
                    },
                    {
                        "name": "bluegreentarget2"
                    }
                ],
                "prodTrafficRoute": {
                    "listenerArns": [
                        "arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/e5ba62739c16e642/665750bec1b03bd4"
                    ]
                }
            }
        ]
    }
}
```

```

        ],
    "serviceRoleArn": "arn:aws:iam::aws_account_id:role/ecsCodeDeployRole",
    "ecsServices": [
        {
            "serviceName": "service-bluegreen",
            "clusterName": "tutorial-bluegreen-cluster"
        }
    ]
}

```

Then create the CodeDeploy deployment group.

```
aws deploy create-deployment-group \
--cli-input-json file://tutorial-deployment-group.json \
--region us-east-1
```

The output includes the deployment group ID, with the following format:

```
{
    "deploymentGroupId": "6fd9bdc6-dc51-4af5-ba5a-0a4a72431c88"
}
```

Step 6: Create and monitor a CodeDeploy deployment

Use the following steps to create and upload an application specification file (AppSpec file) and an CodeDeploy deployment.

To create and monitor an CodeDeploy deployment

1. Create and upload an AppSpec file using the following steps.
 - a. Create a file named `appspec.yaml` with the contents of the CodeDeploy deployment group. This example uses the resources that you created earlier in the tutorial.

```

version: 0.0
Resources:
- TargetService:
  Type: AWS::ECS::Service
  Properties:
    TaskDefinition: "arn:aws:ecs:region:aws_account_id:task-definition/first-
run-task-definition:7"
    LoadBalancerInfo:
      ContainerName: "sample-app"
      ContainerPort: 80
    PlatformVersion: "LATEST"

```

- b. Use the [s3 mb](#) command to create an Amazon S3 bucket for the AppSpec file.

```
aws s3 mb s3://tutorial-bluegreen-bucket
```

- c. Use the [s3 cp](#) command to upload the AppSpec file to the Amazon S3 bucket.

```
aws s3 cp ./appspec.yaml s3://tutorial-bluegreen-bucket/appspec.yaml
```

2. Create the CodeDeploy deployment using the following steps.

- Create a file named `create-deployment.json` with the contents of the CodeDeploy deployment. This example uses the resources that you created earlier in the tutorial.

```
{
    "applicationName": "tutorial-bluegreen-app",
    "deploymentGroupName": "tutorial-bluegreen-dg",
    "revision": {
        "revisionType": "S3",
        "s3Location": {
            "bucket": "tutorial-bluegreen-bucket",
            "key": "appspec.yaml",
            "bundleType": "YAML"
        }
    }
}
```

- Use the [create-deployment](#) command to create the deployment.

```
aws deploy create-deployment \
    --cli-input-json file://create-deployment.json \
    --region us-east-1
```

The output includes the deployment ID, with the following format:

```
{
    "deploymentId": "d-RPCR1U3TW"
}
```

- Use the [get-deployment-target](#) command to get the details of the deployment, specifying the deploymentId from the previous output.

```
aws deploy get-deployment-target \
    --deployment-id "d-INJU3A8TW" \
    --target-id tutorial-bluegreen-cluster:service-bluegreen \
    --region us-east-1
```

Continue to retrieve the deployment details until the status is Succeeded, as shown in the following output.

```
{
    "deploymentTarget": {
        "deploymentTargetType": "ECSTarget",
        "ecsTarget": {
            "deploymentId": "d-RPCR1U3TW",
            "targetId": "tutorial-bluegreen-cluster:service-bluegreen",
            "targetArn": "arn:aws:ecs:region:aws_account_id:service/service-
bluegreen",
            "lastUpdatedAt": 1543431490.226,
            "lifecycleEvents": [
                {
                    "lifecycleEventName": "BeforeInstall",
                    "startTime": 1543431361.022,
                    "endTime": 1543431361.433,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "Install",
                    "startTime": 1543431361.678,
```

```
        "endTime": 1543431485.275,
        "status": "Succeeded"
    },
    {
        "lifecycleEventName": "AfterInstall",
        "startTime": 1543431485.52,
        "endTime": 1543431486.033,
        "status": "Succeeded"
    },
    {
        "lifecycleEventName": "BeforeAllowTraffic",
        "startTime": 1543431486.838,
        "endTime": 1543431487.483,
        "status": "Succeeded"
    },
    {
        "lifecycleEventName": "AllowTraffic",
        "startTime": 1543431487.748,
        "endTime": 1543431488.488,
        "status": "Succeeded"
    },
    {
        "lifecycleEventName": "AfterAllowTraffic",
        "startTime": 1543431489.152,
        "endTime": 1543431489.885,
        "status": "Succeeded"
    }
],
"status": "Succeeded",
"taskSetsInfo": [
    {
        "identifier": "ecs-svc/9223370493425779968",
        "desiredCount": 1,
        "pendingCount": 0,
        "runningCount": 1,
        "status": "ACTIVE",
        "trafficWeight": 0.0,
        "targetGroup": {
            "name": "bluegreentarget1"
        }
    },
    {
        "identifier": "ecs-svc/9223370493423413672",
        "desiredCount": 1,
        "pendingCount": 0,
        "runningCount": 1,
        "status": "PRIMARY",
        "trafficWeight": 100.0,
        "targetGroup": {
            "name": "bluegreentarget2"
        }
    }
]
}
```

Step 7: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

Cleaning up the tutorial resources

1. Use the [delete-deployment-group](#) command to delete the CodeDeploy deployment group.

```
aws deploy delete-deployment-group \
--application-name tutorial-bluegreen-app \
--deployment-group-name tutorial-bluegreen-dg \
--region us-east-1
```

2. Use the [delete-application](#) command to delete the CodeDeploy application.

```
aws deploy delete-application \
--application-name tutorial-bluegreen-app \
--region us-east-1
```

3. Use the [delete-service](#) command to delete the Amazon ECS service. Using the --force flag allows you to delete a service even if it has not been scaled down to zero tasks.

```
aws ecs delete-service \
--service arn:aws:ecs:region:aws_account_id:service/service-bluegreen \
--force \
--region us-east-1
```

4. Use the [delete-cluster](#) command to delete the Amazon ECS cluster.

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

5. Use the [s3 rm](#) command to delete the AppSpec file from the Amazon S3 bucket.

```
aws s3 rm s3://tutorial-bluegreen-bucket/appspec.yaml
```

6. Use the [s3 rb](#) command to delete the Amazon S3 bucket.

```
aws s3 rb s3://tutorial-bluegreen-bucket
```

7. Use the [delete-load-balancer](#) command to delete the Application Load Balancer.

```
aws elbv2 delete-load-balancer \
--load-balancer-arn
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/e5ba62739c16e642 \
--region us-east-1
```

8. Use the [delete-target-group](#) command to delete the two Application Load Balancer target groups.

```
aws elbv2 delete-target-group \
--target-group-arn
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/bluegreentarget1/209a844cd01825a4 \
--region us-east-1
```

```
aws elbv2 delete-target-group \
--target-group-arn
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/bluegreentarget2/708d384187a3cfdc \
--region us-east-1
```

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 [the section called "Creating a cluster for the Fargate launch type using the console" \(p. 236\)](#) 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 function**.
3. On the **Author from scratch** screen, do the following:
 - a. For **Name**, enter a value.
 - b. For **Runtime**, choose your version of Python, for example, **Python 3.9**.
 - c. For **Role**, choose **Create a new role with basic Lambda permissions**.
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 3.9 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. Choose **Save**.

Step 2: Register an 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, Rules, 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: Create a task definition

Create a task definition.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task Definitions**.
3. Choose **Create new Task Definition, Create new revision with JSON**.
4. Copy and paste the following example task definition into the box and then choose **Save**.

```
{  
    "containerDefinitions": [  
        {  
            "entryPoint": [  
                "sh",  
                "-c"  
            ],  
            "portMappings": [  
                {  
                    "hostPort": 80,  
                    "protocol": "tcp",  
                    "containerPort": 80  
                }  
            ],  
            "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\""  
            ],  
            "cpu": 10,  
            "memory": 300,  
            "image": "httpd:2.4",  
            "name": "simple-app"  
        }  
    ],  
    "family": "console-sample-app-static"
```

}

5. Choose **Create**.

Step 4: 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 console at <https://console.aws.amazon.com/ecs/v2>.
2. Choose **Task definitions**.
3. Choose **console-sample-app-static**, and then choose **Deploy**, **Run new task**.
4. For **Cluster**, choose default, and then choose **Deploy**.
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 an Amazon EventBridge 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 [Creating a cluster using the classic console \(p. 879\)](#) 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 EventBridge rule correctly.

Prerequisite: Configure permissions for Amazon SNS

To allow EventBridge to publish to an Amazon SNS topic, use the `aws sns get-topic-attributes` and the `aws sns set-topic-attributes` commands.

For information about how to add the permission, see [Amazon SNS permissions](#) in the *Amazon Simple Notification Service Developer Guide*

Add the following permissions:

```
{  
  "Sid": "PublishEventsToMyTopic",  
  "Effect": "Allow",  
  "Principal": {  
    "Service": "events.amazonaws.com"  
  },  
  "Action": "sns: Publish",  
  "Resource": "arn:aws:sns:region:account-id:TaskStoppedAlert",  
}
```

}

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.

For information about how to create and subscribe to an Amazon SNS topic , see [Getting started with Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide* and use the following table to determine what options to select.

Option	Value
Type	Standard
Name	TaskStoppedAlert
Protocol	Email
Endpoint	An email address to which you currently have access

Step 2: Register an event rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

For information about how to create and subscribe to an Amazon SNS topic , see [Create a rule in Amazon EventBridge](#) in the *Amazon EventBridge User Guide* and use the following table to determine what options to select.

Option	Value
Rule type	Rule with an event pattern
Event source	AWS events or EventBridge partner events
Event pattern	Custom pattern (JSON editor)
Event pattern	<pre>{ "source": ["aws.ecs"], "detail-type": ["ECS Task State Change"], "detail": { "lastStatus": ["STOPPED"], "stoppedReason": ["Essential container in task exited"] } }</pre>

Option	Value
Target type	AWS service
Target	SNS topic
Topic	TaskStoppedAlert (The topic you created in Step 1)

Step 3: Test your rule

Verify that the rule is working by running 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. If you have an existing task definition that can satisfy the rule requirements, run a task using it. If you do not, the following steps will walk you through registering a Fargate task definition and running it that will.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, edit your JSON file,

Copy the following into the editor.

```
{
  "containerDefinitions": [
    {
      "command": [
        "sh",
        "-c",
        "sleep 5"
      ],
      "essential": true,
      "image": "amazonlinux:2",
      "name": "test-sleep"
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "FARGATE"
  ]
}
```

5. Choose **Create**.

To run a task from the console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster you created in the prerequisites.
3. From the **Tasks** tab, choose **Run new task**.
4. For **Application type**, choose **Task**.
5. For **Task definition**, choose **fargate-task-definition**.
6. For **Desired tasks**, enter the number of tasks to launch.

7. Choose **Create**.

Tutorial: Using Amazon EFS file systems with Amazon ECS using the console

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as you add and remove files. Your applications can have the storage they need, when they need it.

You can use Amazon EFS file systems with Amazon ECS to access file system data across your fleet of Amazon ECS tasks. That way, your tasks have access to the same persistent storage, no matter the infrastructure or container instance on which they land. When you reference your Amazon EFS file system and container mount point in your Amazon ECS task definition, Amazon ECS takes care of mounting the file system in your container. The following sections help you get started using Amazon EFS with Amazon ECS.

This feature is supported by tasks that use both the EC2 and Fargate launch types, however this tutorial will use an Amazon ECS task that uses the EC2 launch type. This tutorial is also meant to be followed step by step, however if you already have some of these resources created on your account then you may be able to skip some steps.

Note

Amazon EFS may not be available in all Regions. For more information about which Regions support Amazon EFS, see [Amazon Elastic File System Endpoints and Quotas](#) in the *AWS General Reference*.

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see [the section called "Cluster examples" \(p. 605\)](#).

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
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. Under **Cluster configuration**, for **Cluster name**, enter EFS-tutorial for the cluster name.
6. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
 - To remove a subnet, under **Subnets**, choose X for each subnet that you want to remove.
 - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
7. (Optional) To add Amazon EC2 instances to your cluster, expand **Infrastructure**, and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider:
 - To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:
 - For **Operating system/Architecture**, choose Amazon Linux 2.
 - For **EC2 instance type**, choose t2.micro.

For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.

- For **Capacity**, enter 1.

8. Choose **Create**.

Step 2: Create a security group for the Amazon EFS file system

In this step, you create a security group for your Amazon EFS file system that allows inbound access from your container instances.

Create a security group with the following options:

- **Security group name** - a unique name for your security group. For example, `EFS-access-for-sg-dc025fa2`.
- **VPC** - the VPC that you identified earlier for your cluster.
- **Inbound rule**
 - **Type** - NFS
 - **Source** - Custom with the security group ID.

For information about how to create a security group, see [Create a security group](#) in the *Amazon EC2 User Guide for Linux Instances*.

Step 3: Create an Amazon EFS file system

In this step, you create an Amazon EFS file system.

To create an Amazon EFS file system for Amazon ECS tasks.

1. Open the Amazon Elastic File System console at <https://console.aws.amazon.com/efs/>.
2. Choose **Create file system**.
3. On the **Configure network access** page, choose the VPC that your container instances are hosted in. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC.

Important

Your Amazon EFS file system, your Amazon ECS cluster, container instances and tasks must be in the same VPC.

4. Under **Create mount targets**, for **Security groups**, add the security group that you created in step 2. Choose **Next Step**.
5. On the **Configure file system settings** page, configure optional settings and then choose **Next Step** to proceed.
 - a. (Optional) Add tags for your file system. For example, you could specify a unique name for the file system by entering that name in the **Value** column next to the **Name** key.
 - b. (Optional) Enable lifecycle management to save money on infrequently accessed storage. For more information, see [EFS Lifecycle Management](#) in the *Amazon Elastic File System User Guide*.
 - c. Choose a throughput mode for your file system. The **Bursting** mode is the default, and it is recommended for most file systems.
 - d. Choose a performance mode for your file system. The **General Purpose** mode is the default, and it is recommended for most file systems.

- e. (Optional) Enable encryption. Select the check box to enable encryption of your Amazon EFS file system at rest.
6. On the **Configure client access** page, choose **Next Step**.
7. Review your file system options and choose **Create File System** to complete the process.
8. From the file systems details screen, record the **File system ID**. In the next step, you will reference this value in your Amazon ECS task definition.

Step 4: Add content to the Amazon EFS file system

In this step, you mount the Amazon EFS file system to an Amazon EC2 instance and add content to it. This is for testing purposes in this tutorial, to illustrate the persistent nature of the data. When using this feature you would normally have your application or another method of writing data to your Amazon EFS file system.

To create an Amazon EC2 instance and mount the Amazon EFS file system

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch Instance**.
3. On the **Choose an Amazon Machine Image** page, select the latest **Amazon Linux 2 AMI (HVM)** AMI.
4. On the **Choose an Instance Type** page, keep the default instance type, **t2.micro** and choose **Next: Configure Instance Details**.
5. On the **Configure Instance Details** page, do the following:
 - a. For **Network**, select the VPC that you specified for your Amazon EFS file system and Amazon ECS cluster.
 - b. For **Auto-assign Public IP**, choose **Enable**. Otherwise, your instances do not get public IP addresses or public DNS names.
 - c. For **File systems**, select your Amazon EFS file system. You can optionally change the mount location or leave the default value.
 - d. Under **Advanced Details**, ensure that the user data script is populated automatically with the Amazon EFS file system mounting steps.
6. Advance to step 5 of the instance wizard by choosing **Next: Add Storage**, **Next: Add Tags**, and **Next: Configure Security Group**.
7. On the **Configure Security Group** page, choose **Select an existing security group** and select the security group that you created in step 1, and then choose **Review and Launch**.
8. On the **Review Instance Launch** page, choose **Launch**.
9. On the **Select an existing key pair or create a new key pair** dialog box, select **Choose an existing key pair** and choose your key pair. Select the acknowledgment check box, and choose **Launch Instances**.
10. On the **Launch Status** page, choose **View Instances** to see the status of your instances. Initially, their status is pending. After the status changes to **running**, your instances are ready for use.

Now, you connect to the Amazon EC2 instance and add content to the Amazon EFS file system.

To connect to the Amazon EC2 instance and add content to the Amazon EFS file system

1. SSH to the Amazon EC2 instance you created. For more information, see [Connect to Your Linux Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
2. From the terminal window for each instance, run the **df -T** command to verify that the Amazon EFS file system is mounted. In the following output, we have highlighted the Amazon EFS file system mount.

```
$ df -T
Filesystem      Type       1K-blocks   Used   Available Use% Mounted on
devtmpfs        devtmpfs    485468     0      485468  0% /dev
tmpfs           tmpfs       503480     0      503480  0% /dev/shm
tmpfs           tmpfs       503480    424     503056  1% /run
tmpfs           tmpfs       503480     0      503480  0% /sys/fs/cgroup
/dev/xvda1      xfs        8376300  1310952  7065348 16% /
127.0.0.1:/    nfs4        9007199254739968  0  9007199254739968  0% /mnt/efs/fs1
tmpfs           tmpfs       100700     0      100700  0% /run/user/1000
```

3. Navigate to the directory that the Amazon EFS file system is mounted at. In the example above, that is /mnt/efs/fs1.
4. Create a file named index.html with the following content:

```
<html>
  <body>
    <h1>It Works!</h1>
    <p>You are using an Amazon EFS file system for persistent container storage.</p>
  </body>
</html>
```

Step 5: Create a task definition

The following task definition creates a data volume named efs-html. The nginx container mounts the host data volume at the NGINX root, /usr/share/nginx/html.

To create a new task definition using the classic console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, copy and paste the following JSON text, replacing the fileSystemId with the ID of your Amazon EFS file system.

```
{
  "containerDefinitions": [
    {
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "containerPath": "/usr/share/nginx/html",
          "sourceVolume": "efs-html"
        }
      ],
      "name": "nginx",
      "image": "nginx"
    }
  ],
  "fileSystemConfiguration": [
    {
      "name": "efs-html",
      "fileSystemId": "ebs-fs-01234567890abcdef0"
    }
  ]
}
```

```

    "volumes": [
      {
        "name": "efs-html",
        "efsVolumeConfiguration": {
          "fileSystemId": "fs-1324abcd",
          "transitEncryption": "ENABLED"
        }
      },
      "family": "efs-tutorial"
    ]
  }
}

```

5. Choose **Save, Create**.

Step 6: Run a task and view the results

Now that your Amazon EFS file system is created and there is web content for the NGINX container to serve, you can run a task using the task definition that you created. The NGINX web server serves your simple HTML page. If you update the content in your Amazon EFS file system, those changes are propagated to any containers that have also mounted that file system.

The task runs in the subnet that you defined for the cluster.

To run a task and view the results using the console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster to run the standalone task in.

Determine the resource from where you launch the service.

To start a service from	Steps
Clusters	<ol style="list-style-type: none"> a. On the Clusters page, select the cluster to create the service in. b. From the Tasks tab, choose Run new task.
Launch type	<ol style="list-style-type: none"> a. On the Task page, choose the task definition. b. If there is more than one revision, select the revision. c. Choose Create, Run task.

3. (Optional) Choose how your scheduled task is distributed across your cluster infrastructure. Expand **Compute configuration**, and then do the following:

Distribution method	Steps
Launch type	<ol style="list-style-type: none"> a. In the Compute options section, select Launch type. b. For Launch type, choose EC2.

4. For **Application type**, choose **Task**.
5. For **Task definition**, choose the efs-tutorial task definition that you created earlier .

Important

The console validates the selection to ensure that the selected task definition family and revision is compatible with the defined compute configuration.

6. For **Desired tasks**, enter 1.
7. Choose **Create**.
8. On the **Cluster** page, choose **Infrastructure**.
9. Under **Container Instances**, choose the container instance to connect to.
10. On the **Container Instance** page, under **Networking**, record the **Public IP** for your instance.
11. Open a browser and enter the public IP address browser. You should see the following message:

It works!
You are using an Amazon EFS file system for persistent container storage.

Note

If you do not see the message, make sure that the security group for your container instance allows inbound network traffic on port 80.

Tutorial: Using FSx for Windows File Server file systems with Amazon ECS

FSx for Windows File Server provides fully managed Microsoft Windows file servers, that are backed by a fully native Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#) in the *FSx for Windows File Server User Guide*.

You can use FSx for Windows File Server to deploy Windows workloads that require access to shared external storage, highly available regional storage, or high-throughput storage. You can mount one or more FSx for Windows File Server file system volumes to an ECS container running on an ECS Windows instance. You can share FSx for Windows File Server file system volumes among multiple ECS containers within a single ECS task.

Note

FSx for Windows File Server might not be available in all Regions. For more information about which Regions support FSx for Windows File Server, see [Amazon FSx Endpoints and Quotas](#) in the *AWS General Reference*.

In this tutorial, you launch an ECS Optimized Windows instance that hosts an FSx for Windows File Server file system and containers that can access the file system. To do this, you first create an AWS Directory Service AWS Managed Microsoft Active Directory. Then, you create an Amazon FSx for Windows File Server file system and an ECS cluster with an ECS instance and an ECS task definition. You configure the task definition for your containers to use the FSx for Windows File Server file system. Finally, you test the file system.

It takes 20 to 45 minutes each time you launch or delete either the Active Directory or the FSx for Windows File Server file system. Be prepared to reserve at least 90 minutes to complete the tutorial or complete the tutorial over a few sessions.

Prerequisites for the tutorial

- An administrative user. See [Set up to use Amazon ECS \(p. 10\)](#).
- (Optional) A PEM key pair for connecting to your EC2 Windows instance through RDP access. For information about how to create key pairs, see [Amazon EC2 key pairs and Windows instances](#) in the *User Guide for Windows Instances*.

- A VPC with at least one public and one private subnet, and one security group. You can use your default VPC. You don't need a NAT gateway or device. AWS Directory Service doesn't support Network Address Translation (NAT) with Active Directory. For this to work, the Active Directory, FSx for Windows File Server file system, ECS Cluster, and ECS instance must be located within your VPC. For more information regarding VPCs and Active Directories, see [Amazon VPC console wizard configurations](#) and [AWS Managed Microsoft AD Prerequisites](#).
- The IAM ecsInstanceRole and ecsTaskExecutionRole permissions are associated with your account. These service-linked roles allow services to make API calls and access containers, secrets, directories and file servers on your behalf.

Step 1: Create IAM access roles

Create a cluster with the AWS Management Console.

1. See [Amazon ECS container instance IAM role \(p. 638\)](#) to check whether you have an ecsInstanceRole and to see how you can create one if you don't have one.
2. We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policy is attached to your ecsInstanceRole. Attach the policy if it is not already attached.
 - AmazonEC2ContainerServiceforEC2Role
 - AmazonSSMManagedInstanceStateCore
 - AmazonSSMDirectoryServiceAccess

To attach AWS managed policies.

- a. Open the [IAM console](#).
- b. In the navigation pane, choose **Roles**.
- c. Choose an **AWS managed role**.
- d. Choose **Permissions, Attach policies..**.
- e. To narrow the available policies to attach, use **Filter**.
- f. Select the appropriate policy and choose **Attach policy**.
3. See [Amazon ECS task execution IAM role \(p. 626\)](#) to check whether you have an ecsTaskExecutionRole and to see how you can create one if you don't have one.

We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policies are attached to your ecsTaskExecutionRole. Attach the policies if they are not already attached. Use the procedure given in the preceding section to attach the AWS managed policies.

- SecretsManagerReadWrite
- AmazonFSxReadOnlyAccess
- AmazonSSMReadOnlyAccess
- AmazonECSTaskExecutionRolePolicy

Step 2: Create Windows Active Directory (AD)

1. Follow the steps described in [Create Your AWS Managed AD Directory](#) in the *AWS Directory Service Administration Guide*. Use the VPC you have designated for this tutorial. On Step 3 of *Create Your AWS Managed AD Directory*, save the user name and password for use in a following step. Also, note

the fully qualified domain name for future steps. You can go on to complete the following step while the Active Directory is being created.

2. Create an AWS Secrets Manager secret to use in the following steps. For more information, see [Getting Started with AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.
 - a. Open the [Secrets Manager console](#).
 - b. Click **Store a new secret**.
 - c. Select **Other type of secrets**.
 - d. For **Secret key/value**, in the first row, create a key **username** with value **admin**. Click on **+ Add row**.
 - e. In the new row, create a key **password**. For value, type in the password you entered in Step 3 of *Create Your AWS Managed AD Directory*.
 - f. Click on the **Next** button.
 - g. Provide a secret name and description. Click **Next**.
 - h. Click **Next**. Click **Store**.
 - i. From the list of **Secrets** page, click on the secret you have just created.
 - j. Save the ARN of the new secret for use in the following steps.
 - k. You can proceed to the next step while your Active Directory is being created.

Step 3: Verify and update your security group

In this step, you verify and update the rules for the security group that you're using. For this, you can use the default security group that was created for your VPC.

Verify and update security group.

You need to create or edit your security group to send data from and to the ports, which are described in [Amazon VPC Security Groups](#) in the *FSx for Windows File Server User Guide*. You can do this by creating the security group inbound rule shown in the first row of the following table of inbound rules. This rule allows inbound traffic from network interfaces (and their associated instances) that are assigned to the security group. All of the cloud resources you create are within the same VPC and attached to the same security group. Therefore, this rule allows traffic to be sent to and from the FSx for Windows File Server file system, Active Directory, and ECS instance as required. The other inbound rules allow traffic to serve the website and RDP access for connecting to your ECS instance.

The following table shows which security group inbound rules are required for this tutorial.

Type	Protocol	Port range	Source
All traffic	All	All	<i>sg-securitygroup</i>
Custom TCP	TCP	8080	0.0.0.0/0
RDP	TCP	3389	your laptop IP address

The following table shows which security group outbound rules are required for this tutorial.

Type	Protocol	Port range	Destination
All traffic	All	All	0.0.0.0/0

1. Open the [EC2 console](#) and select **Security Groups** from the left-hand menu.
2. From the list of security groups now displayed, select check the check-box to the left of the security group that you are using for this tutorial.

Your security group details are displayed.
3. Edit the inbound and outbound rules by selecting the **Inbound rules** or **Outbound rules** tabs and choosing the **Edit inbound rules** or **Edit outbound rules** buttons. Edit the rules to match those displayed in the preceding tables. After you create your EC2 instance later on in this tutorial, edit the inbound rule RDP source with the public IP address of your EC2 instance as described in [Connect to your Windows instance](#) from the *Amazon EC2 User Guide for Windows Instances*.

Step 4: Create an FSx for Windows File Server file system

After your security group is verified and updated and your Active Directory is created and is in the active status, create the FSx for Windows File Server file system in the same VPC as your Active Directory. Use the following steps to create an FSx for Windows File Server file system for your Windows tasks.

Create your first file system.

1. Open the [Amazon FSx console](#).
2. On the dashboard, choose **Create file system** to start the file system creation wizard.
3. On the **Select file system type** page, choose **FSx for Windows File Server**, and then choose **Next**. The **Create file system** page appears.
4. In the **File system details** section, provide a name for your file system. Naming your file systems makes it easier to find and manage them. You can use up to 256 Unicode characters. Allowed characters are letters, numbers, spaces, and the special characters plus sign (+), minus sign (-), equal sign (=), period (.), underscore (_), colon (:), and forward slash (/).
5. For **Deployment type** choose **Single-AZ** to deploy a file system that is deployed in a single Availability Zone. *Single-AZ 2* is the latest generation of single Availability Zone file systems, and it supports SSD and HDD storage.
6. For **Storage type**, choose **HDD**.
7. For **Storage capacity**, enter the minimum storage capacity.
8. Keep **Throughput capacity** at its default setting.
9. In the **Network & security** section, choose the same Amazon VPC that you chose for your AWS Directory Service directory.
10. For **VPC Security Groups**, choose the security group that you verified in *Step 3: Verify and update your security group*.
11. For **Windows authentication**, choose **AWS Managed Microsoft Active Directory**, and then choose your AWS Directory Service directory from the list.
12. For **Encryption**, keep the default **Encryption key** setting of **aws/fsx (default)**.
13. Keep the default settings for **Maintenance preferences**.
14. Click on the **Next** button.
15. Review the file system configuration shown on the **Create file system** page. For your reference, note which file system settings you can modify after file system is created. Choose **Create file system**.
16. Note the file system ID. You will need to use it in a later step.

You can go on to the next steps to create a cluster and EC2 instance while the FSx for Windows File Server file system is being created.

Step 5: Create an Amazon ECS cluster

Create a cluster using the classic Amazon ECS console

1. Open the [Amazon ECS console](#).
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create Cluster**.
4. Choose **EC2 Windows + Networking** and choose **Next step**.
5. For **Cluster name** enter windows-fsx-cluster.
6. Click the check-box under the name of the cluster to **create an empty cluster**.
7. Click the **Create** button on the lower right corner.
8. Click the **View Cluster** button when the cluster is successfully created.

You are now on a page where you can view the details of your cluster.

Step 6: Create an Amazon ECS instance

Launch an ECS Optimized Windows EC2 instance into the ECS cluster you just created using the AWS Management Console and the classic Amazon ECS console

1. Go to [Amazon ECS-optimized AMI](#) in the *Amazon ECS Developer Guide* to find the latest version of the Windows Server 2019 Full AMI in the same Region as your VPC.
 2. You can get the latest version using one of the following steps.

Scroll down to the Windows Server 2019 Full AMI table.

 - a. Find the latest version in the table for your Region. Click **View AMI ID** link to a page where you'll find the AMI ID of the latest version. Save a copy of the AMI ID for the next steps.
 - b. Run the given Systems Manager command using the AWS CLI and save a copy of the AMI ID that is returned.
 3. Open the [Amazon EC2 console](#).
 4. Click on the **Launch Instance** button and select **Launch Instance**.
- You are now on a page that lists available EC2 instances.
5. **Select an AMI for your EC2 instance.**
 - a. Under **Quick Start**, click on **Community AMIs**.
 - b. In the **search** field, enter the AMI ID that you saved from the previous step and press return.
 - c. Select the Windows Server 2019 Full AMI that matches the AMI ID that you saved in the previous step.
- You are now on a page listing instance types.
6. For **Instance type** page, choose t2.medium or t2.micro and click on **Next: Configure Instance Details**.
 7. **Configure instance details.**
 - a. On the **Configure Instance Details** page, enter 1 for **Number of Instances**.
 - b. For **Network** select your VPC.
 - c. For **Subnets** select a public subnet.
 - d. Select **Enable** for **Auto-assign Public IP**.

- e. For **Domain join directory**, select the ID of the Active Directory that you created. This option domain joins your AD when the EC2 instance is launched.
- f. For **IAM role**, select your **ecsInstanceRole** from the drop-down menu.
- g. Scroll to the bottom of the page and enter the following into the **User data** text field.

```
<powershell>
Initialize-ECSAgent -Cluster windows-fsx-cluster -EnableTaskIAMRole
</powershell>
```

- h. Click **Next: Add Storage** button.
- i. Click **Next: Add Tags** button.
- j. Click **Next: Configure Security Group** button.
8. On the **Configure Security Group** page, select the security group that you verified and updated in *Step 3: Verify and update your security group*. If it doesn't already exist, add an inbound RDP TCP rule to allow traffic from your EC2 instance IP address through port 3389 if you want to be able to RDP into your instance.
9. Click on **Review and Launch** button.
10. On the **Review Instance Launch** page, click the **Launch** button.
11. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for RDP access. If you don't specify a key pair, you can't connect to your container instances with RDP. For more information, see [Prerequisites for the tutorial \(p. 723\)](#).
12. Click on **View instance** to see the new instance status among your list of instances.
13. Open the classic [Amazon ECS console](#) and select **Clusters**.
14. Select your **fsx-windows-cluster** cluster.
15. Select the **ECS Instances** tab and verify that your ECS instance has been registered in the **fsx-windows-cluster** cluster.

Step 7: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple web page on port 8080 of a container instance. The task launches two containers that have access to the FSx file system. The first container writes an HTML file to the file system. The second container downloads the HTML file from the file system and serves the webpage.

Register the sample task definition with the classic Amazon ECS console

1. Open the [Amazon ECS console](#).
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 compatibilities** page, choose **EC2** and then **Next step**.

Note

The Fargate launch type isn't compatible with Windows containers.

5. On the **Create new Task Definition** page, scroll to the bottom of the page and choose **Configure via JSON**.
6. Use the following sample task definition JSON. Replace the values for your task execution role and the details about your FSx file system and then choose **Save**.

```
{  
  "containerDefinitions": [  
    {
```

```

    "entryPoint": [
        "powershell",
        "-Command"
    ],
    "portMappings": [],
    "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType file -Value '<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>It Works!</h2> <p>You are using Amazon FSx for Windows File Server file system for persistent container storage.</p>' -Force"],
        "cpu": 512,
        "memory": 256,
        "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
        "essential": false,
        "name": "container1",
        "mountPoints": [
            {
                "sourceVolume": "fsx-windows-dir",
                "containerPath": "C:\\fsx-windows-dir",
                "readOnly": false
            }
        ]
    },
    {
        "entryPoint": [
            "powershell",
            "-Command"
        ],
        "portMappings": [
            {
                "hostPort": 443,
                "protocol": "tcp",
                "containerPort": 80
            }
        ],
        "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force; Start-Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -Destination C:\\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe w3svc"],
        "mountPoints": [
            {
                "sourceVolume": "fsx-windows-dir",
                "containerPath": "C:\\fsx-windows-dir",
                "readOnly": false
            }
        ],
        "cpu": 512,
        "memory": 256,
        "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
        "essential": true,
        "name": "container2"
    }
],
"family": "fsx-windows",
"executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
"volumes": [
{
    "name": "fsx-windows-dir",
    "fsxWindowsFileServerVolumeConfiguration": {
        "fileSystemId": "fs-0eeb5730b2EXAMPLE",
        "authorizationConfig": {
            "domain": "example.com",
            "credentialsParameter": "arn:arn-1234"
        },
}
]

```

```
        "rootDirectory": "share"
    }
}
]
```

7. Click on **container1** under **Container Definitions**.
8. Scroll to **STORAGE AND LOGGING** and, for **Mount points Source volume**, select **fsx-windows-vol** from the drop-down menu.
9. For **Container path**, enter **C:\fsx-windows-dir**.
10. Click on **Update** button.
11. Repeat the last four steps for **container2** under **Container Definitions**.
12. For **Task execution role**, choose your **ecsTaskExecutionRole** from the drop-down menu.
13. Verify your information and click on **Create** button.

Step 8: Run a task and view the results

Before running the task, verify that the status of your FSx for Windows File Server file system is **Available**. After it is available, you can run a task using the task definition that you created. The task starts out by creating containers that shuffle an HTML file between them using the file system. After the shuffle, a web server serves the simple HTML page.

Note

You might not be able to connect to the website from within a VPN.

Run a task and view the results with the classic console.

1. Open the [Amazon ECS console](#).
2. Choose your **fsx-windows-cluster** cluster.
3. Choose **Tasks** tab, and then **Run new task**.
4. For **Launch Type**, select EC2.
5. For **Task Definition**, choose the **fsx-windows** task definition that you created, and then choose **Run Task**.
6. Under the **Tasks** tab, choose the task that you just ran. Your task appears in the list of tasks.
7. When your task status is **RUNNING**, click on the task ID.
8. Expand container2.
9. Scroll down and click on the external IP address that is associated with the container. Your browser will open and display the following message.

Note

If you don't see this message, check that you aren't running in a VPN and make sure that the security group for your container instance allows inbound network HTTP traffic on port 8080.

Step 9: Clean up

Note

It takes 20 to 45 minutes to delete the FSx for Windows File Server file system or the AD. You must wait until the FSx for Windows File Server file system delete operations are complete before starting the AD delete operations.

Remove FSx for Windows File Server file system.

1. Open the [Amazon FSx console](#)

2. Click the radio button to the left of the FSx for Windows File Server file system that you just created.
3. Click on **Actions**.
4. Select **Delete file system**.

Remove AD.

1. Open the [AWS Directory Service console](#).
2. Click the radio button to the left of the AD you just created.
3. Click on **Actions**.
4. Select **Delete directory**.

Remove ECS cluster.

1. Open the [Amazon ECS console](#).
2. Select **Clusters**.
3. Select the cluster you just created.
4. Click the **Delete Cluster** button.

Remove ECS instance.

1. Open the [Amazon EC2 console](#).
2. From the left-hand menu, select **Instances**.
3. Check the check-box to the left of the EC2 instance you created during this exercise.
4. Click the **Instance state** and then **Terminate instance**.

Remove secret.

1. Open the [Secrets Manager console](#).
2. Select the secret you created for this walk through.
3. Click **Actions**.
4. Select **Delete secret**.

Tutorial: Deploying Fluent Bit on Amazon ECS for Windows containers

Fluent Bit is a fast and flexible log processor and router supported by various operating systems. It can be used to route logs to various AWS destinations such as Amazon CloudWatch Logs, Kinesis Data Firehose Amazon S3, and Amazon OpenSearch Service. Fluent Bit supports common partner solutions such as [Datadog](#), [Splunk](#), and custom HTTP servers. For more information about Fluent Bit, see the [Fluent Bit](#) website.

The **AWS for Fluent Bit** image is available on Amazon ECR on both the Amazon ECR Public Gallery and in an Amazon ECR repository in most Regions for high availability. For more information, see [aws-for-fluent-bit](#) on the GitHub website.

This tutorial walks you through how to deploy Fluent Bit containers on their Windows instances running in Amazon ECS to stream logs generated by the Windows tasks to Amazon CloudWatch for centralized logging.

This tutorial uses the following approach:

- Fluent Bit runs as a service with the Daemon scheduling strategy. This strategy ensures that a single instance of Fluent Bit always runs on the container instances in the cluster.
 - Listens on port 24224 using the forward input plug-in.
 - Expose port 24224 to the host so that the docker runtime can send logs to Fluent Bit using this exposed port.
 - Has a configuration which allows Fluent Bit to send the logs records to specified destinations.
- Launch all other Amazon ECS task containers using the fluentd logging driver. For more information, see [Fluentd logging driver](#) on the Docker documentation website.
 - Docker connects to the TCP socket 24224 on localhost inside the host namespace.
 - The Amazon ECS agent adds labels to the containers which includes the cluster name, task definition family name, task definition revision number, task ARN, and the container name. The same information is added to the log record using the labels option of the fluentd docker logging driver. For more information, see [labels, labels-regex, env, and env-regex](#) on the Docker documentation website.
 - Because the async option of the fluentd logging driver is set to true, when the Fluent Bit container is restarted, docker buffers the logs until the Fluent Bit container is restarted. You can increase the buffer limit by setting the fluentd-buffer-limit option. For more information, see [fluentd-buffer-limit](#) on the Docker documentation website.

The work flow is as follows:

- The Fluent Bit container starts and listens on port 24224 which is exposed to the host.
- Fluent Bit uses the task IAM role credentials specified in its task definition.
- Other tasks launched on the same instance use the fluentd docker logging driver to connect to the Fluent Bit container on port 24224.
- When the application containers generate logs, docker runtime tags those records, adds additional metadata specified in labels, and then forwards them on port 24224 in the host namespace.
- Fluent Bit receives the log record on port 24224 because it is exposed to the host namespace.
- Fluent Bit performs its internal processing and routes the logs as specified.

This tutorial uses the default CloudWatch Fluent Bit configuration which does the following:

- Creates a new log group for each cluster and task definition family.
- Creates a new log stream for each task container in above generated log group whenever a new task is launched. Each stream will be marked with the task id to which the container belongs.
- Adds additional metadata including the cluster name, task ARN, task container name, task definition family, and the task definition revision number in each log entry.

For example, if you have task_1 with container_1 and container_2 and task_2 with container_3, then the following are the CloudWatch log streams:

- /aws/ecs/windows.ecs_task_1
 - task-out.*TASK_ID*.container_1
 - task-out.*TASK_ID*.container_2
- /aws/ecs/windows.ecs_task_2
 - task-out.*TASK_ID*.container_3

Steps

- [Prerequisites \(p. 733\)](#)
- [Step 1: Create the IAM access roles \(p. 733\)](#)
- [Step 2: Create an Amazon ECS Windows container instance \(p. 734\)](#)
- [Step 3: Configure Fluent Bit \(p. 735\)](#)
- [Step 4: Register a Windows Fluent Bit task definition which routes the logs to CloudWatch \(p. 736\)](#)
- [Step 5: Run the ecs-windows-fluent-bit task definition as an Amazon ECS service using the daemon scheduling strategy \(p. 737\)](#)
- [Step 6: Register a Windows task definition which generates the logs \(p. 738\)](#)
- [Step 7: Run the windows-app-task task definition \(p. 739\)](#)
- [Step 8: Verify the logs on CloudWatch \(p. 739\)](#)
- [Step 9: Clean up \(p. 740\)](#)

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, see [Installing the AWS Command Line Interface](#).
- The aws-for-fluent-bit container image is available for the following Windows operating systems:
 - Windows Server 2019 Core
 - Windows Server 2019 Full
 - Windows Server 2022 Core
 - Windows Server 2022 Full
- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- You have a cluster. In this tutorial, the cluster name is **FluentBit-cluster**.
- You have a VPC with a public subnet where the EC2 instance will be launched. You can use your default VPC. You can also use a private subnet that allows Amazon CloudWatch endpoints to reach the subnet. For more information about Amazon CloudWatch endpoints, see [Amazon CloudWatch endpoints and quotas](#) in the *AWS General Reference*. For information about how to use the Amazon VPC wizard to create a VPC, see [the section called "Create a virtual private cloud" \(p. 12\)](#).

Step 1: Create the IAM access roles

Create the Amazon ECS IAM roles.

1. Create the Amazon ECS container instance role named "ecsInstanceRole". For more information, see [Amazon ECS container instance IAM role](#).
2. Create an IAM role for the Fluent Bit task named fluentTaskRole. For more information, see [the section called "Task IAM role" \(p. 631\)](#).

The IAM permissions granted in this IAM role are assumed by the task containers. In order to allow Fluent Bit to send logs to CloudWatch, you need to attach the following permissions to the task IAM role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "logs>CreateLogStream",  
        "logs>CreateLogGroup",  
        "logs>DescribeLogStreams",  
        "logs>PutLogEvents"  
    ],  
    "Resource": "*"  
}  
}  
}
```

3. Attach the policy to the role.
 - a. Save the above content in a file named fluent-bit-policy.json.
 - b. Run the following command to attach the inline policy to fluentTaskRole IAM role.

```
aws iam put-role-policy --role-name fluentTaskRole --policy-name fluentTaskPolicy  
--policy-document file://fluent-bit-policy.json
```

Step 2: Create an Amazon ECS Windows container instance

Create an Amazon ECS Windows container instance.

To create an Amazon ECS instance

1. Use the aws ssm get-parameters command to retrieve the AMI ID for the Region that hosts your VPC. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata](#).
2. Use the Amazon EC2 console to launch the instance.
 - a. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
 - b. From the navigation bar, select the Region to use.
 - c. From the **EC2 Dashboard**, choose **Launch instance**.
 - d. For **Name**, enter a unique name.
 - e. For **Application and OS Images (Amazon Machine Image)**, choose the AMI that you retrieved in the first step.
 - f. For **Instance type**, choose t3.xlarge.
 - g. For **Key pair (login)**, choose a key pair.
 - h. Under **Network settings**, for **Security group**, choose an existing security group, or create a new one.
 - i. Under **Network settings**, for **Auto-assign Public IP**, select **Enable**.
 - j. Under **Advanced details**, for **IAM instance profile**, choose **ecsInstanceRole**.
 - k. Configure your Amazon ECS container instance with the following user data. Under **Advanced Details**, paste the following script into the **User data** field, replacing **cluster_name** with the name of your cluster.

```
<powershell>  
Import-Module ECSTools  
Initialize-ECSAgent -Cluster cluster-name -EnableTaskENI -EnableTaskIAMRole -  
LoggingDrivers '[{"awslogs", "fluentd"}]'  
</powershell>
```

- l. When you are ready, select the acknowledgment field, and then choose **Launch Instances**.
- m. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

Step 3: Configure Fluent Bit

You can use the following default configuration provided by AWS to get quickly started:

- [Amazon CloudWatch](#) which is based on the Fluent Bit plug-in for [Amazon CloudWatch](#) on the *Fluent Bit Official Manual*.

Alternatively, you can use other default configurations provided by AWS. For more information, see [Overriding the entrypoint for the Windows image](#) on the `aws-for-fluent-bit` the Github website.

The default Amazon CloudWatch Fluent Bit configuration is shown below.

Replace the following variables:

- *region* with the Region where you want to send the Amazon CloudWatch logs.

```
[SERVICE]
  Flush      5
  Log_Level  info
  Daemon    off

[INPUT]
  Name      forward
  Listen    0.0.0.0
  Port      24224
  Buffer_Chunk_Size 1M
  Buffer_Max_Size 6M
  Tag_Prefix ecs.

# Amazon ECS agent adds the following log keys as labels to the docker container.
# We would use fluentd logging driver to add these to log record while sending it to Fluent
Bit.
[FILTER]
  Name      modify
  Match    ecs./*
  Rename   com.amazonaws.ecs.cluster ecs_cluster
  Rename   com.amazonaws.ecs.container-name ecs_container_name
  Rename   com.amazonaws.ecs.task-arn ecs_task_arn
  Rename   com.amazonaws.ecs.task-definition-family ecs_task_definition_family
  Rename   com.amazonaws.ecs.task-definition-version
  ecs_task_definition_version

[FILTER]
  Name      rewrite_tag
  Match    ecs./*
  Rule     $ecs_task_arn ^([a-z-:0-9]+)([a-zA-Z0-9-_]+)([a-z0-9]+)$  out.$3.
$ecs_container_name false
  Emitter_Name re_emitted

[OUTPUT]
  Name      cloudwatch_logs
  Match    out./*
  region   region
  log_group_name fallback-group
  log_group_template /aws/ecs/$ecs_cluster.$ecs_task_definition_family
```

```
log_stream_prefix  task-
auto_create_group  On
```

Every log which gets into Fluent Bit has a tag which you specify, or is automatically generated when you do not supply one. The tags can be used to route different logs to different destinations. For additional information, see [Tag](#) in the *Fluent Bit Official Manual*.

The Fluent Bit configuration described above has the following properties:

- The forward input plug-in listens for incoming traffic on TCP port 24224.
- Each log entry received on that port has a tag which the forward input plug-in modifies to prefix the record with `ecs.` string.
- The Fluent Bit internal pipeline routes the log entry to modify the filter using the Match regex. This filter replaces the keys in the log record JSON to the format which Fluent Bit can consume.
- The modified log entry is then consumed by the rewrite_tag filter. This filter changes the tag of the log record to the format `out.TASK_ID.CONTAINER_NAME`.
- The new tag will be routed to output cloudwatch_logs plug-in which creates the log groups and streams as described earlier by using the `log_group_template` and `log_stream_prefix` options of the CloudWatch output plug-in. For additional information, see [Configuration parameters](#) in the *Fluent Bit Official Manual*.

Step 4: Register a Windows Fluent Bit task definition which routes the logs to CloudWatch

Register a Windows Fluent Bit task definition which routes the logs to CloudWatch.

Note

This task definition exposes Fluent Bit container port 24224 to the host port 24224. Verify that this port is not open in your EC2 instance security group to prevent access from outside.

To register a task definition

1. Create a file named `fluent-bit.json` with the following contents.

Replace the following variables:

- `task-iam-role` with the Amazon Resource Name (ARN) of your task IAM role
- `region` with the Region where your task runs

```
{
  "family": "ecs-windows-fluent-bit",
  "taskRoleArn": "task-iam-role",
  "containerDefinitions": [
    {
      "name": "fluent-bit",
      "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:windowsservercore-latest",
      "cpu": 512,
      "portMappings": [
        {
          "hostPort": 24224,
          "containerPort": 24224,
          "protocol": "tcp"
        }
      ],
      "entryPoint": [
        "fluent-bit"
      ]
    }
  ]
}
```

Step 5: Run the `ecs-windows-fluent-bit` task definition as an Amazon ECS service using the daemon scheduling strategy

```
"Powershell",
"-Command"
],
"command": [
"C:\\\\entrypoint.ps1 -ConfigFile C:\\\\ecs_windows_forward_daemon\\\\cloudwatch.conf"
],
"environment": [
{
"name": "AWS_REGION",
"value": "region"
}
],
"memory": 512,
"essential": true,
"logConfiguration": {
"logDriver": "awslogs",
"options": {
"awslogs-group": "/ecs/fluent-bit-logs",
"awslogs-region": "region",
"awslogs-stream-prefix": "flb",
"awslogs-create-group": "true"
}
}
},
"memory": "512",
"cpu": "512"
}
```

- Run the following command to register the task definition.

```
aws ecs register-task-definition --cli-input-json file://fluent-bit.json --region region
```

You can list the task definitions for your account by running the `list-task-definitions` command. The output displays the family and revision values that you can use together with `run-task` or `start-task`.

Step 5: Run the ecs-windows-fluent-bit task definition as an Amazon ECS service using the daemon scheduling strategy

After you register a task definition for your account, you can run a task in the cluster. For this tutorial, you run one instance of the `ecs-windows-fluent-bit:1` task definition in your FluentBit-`cluster` cluster. Run the task in a service which uses the daemon scheduling strategy, which ensures that a single instance of Fluent Bit always runs on each of your container instances.

To run a task

- Run the following command to start the `ecs-windows-fluent-bit:1` task definition (registered in the previous step) as a service.

Note

This task definition uses the awslogs logging driver, your container instance need to have the necessary permissions.

Replace the following variables:

- *region* with the Region where your service runs

```
aws ecs create-service \
--cluster FluentBit-cluster \
--service-name FluentBitForwardDaemonService \
--task-definition ecs-windows-fluent-bit:1 \
--launch-type EC2 \
--scheduling-strategy DAEMON \
--region region
```

2. Run the following command to list your tasks.

Replace the following variables:

- *region* with the Region where your service tasks run

```
aws ecs list-tasks --cluster FluentBit-cluster --region region
```

Step 6: Register a Windows task definition which generates the logs

Register a task definition which generates the logs. This task definition deploys Windows container image which will write a incremental number to stdout every second.

The task definition uses the fluentd logging driver which connects to port 24224 which the Fluent Bit plug-in listens to. The Amazon ECS agent labels each Amazon ECS container with tags including the cluster name, task ARN, task definition family name, task definition revision number and the task container name. These key-value labels are passed to Fluent Bit.

Note

This task uses the default network mode. However, you can also use the awsvpc network mode with the task.

To register a task definition

1. Create a file named windows-app-task.json with the following contents.

```
{
  "family": "windows-app-task",
  "containerDefinitions": [
    {
      "name": "sample-container",
      "image": "mcr.microsoft.com/windows/servercore:ltsc2019",
      "cpu": 512,
      "memory": 512,
      "essential": true,
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
        "$count=1;while(1) { Write-Host $count; sleep 1; $count=$count+1;}"
      ],
      "logConfiguration": {
        "logDriver": "fluentd",
        "options": {
          "logFile": "/var/log/fluentd.log"
        }
      }
    }
  ]
}
```

```
        "options": {
            "fluentd-address": "localhost:24224",
            "tag": "{{ index .ContainerLabels \"com.amazonaws.ecs.task-definition-family\" }}",
            "fluentd-async": "true",
            "labels": "com.amazonaws.ecs.cluster,com.amazonaws.ecs.container-name,com.amazonaws.ecs.task-arn,com.amazonaws.ecs.task-definition-family,com.amazonaws.ecs.task-definition-version"
        }
    },
    "memory": "512",
    "cpu": "512"
}
```

- Run the following command to register the task definition.

Replace the following variables:

- *region* with the Region where your task runs

```
aws ecs register-task-definition --cli-input-json file://windows-app-task.json --region region
```

You can list the task definitions for your account by running the `list-task-definitions` command. The output of displays the family and revision values that you can use together with `run-task` or `start-task`.

Step 7: Run the windows-app-task task definition

After you register the windows-app-task task definition, run it in your FluentBit-cluster cluster.

To run a task

- Run the windows-app-task:1 task definition you registered in the previous step.

Replace the following variables:

- *region* with the Region where your task runs

```
aws ecs run-task --cluster FluentBit-cluster --task-definition windows-app-task:1 --count 2 --region region
```

- Run the following command to list your tasks.

```
aws ecs list-tasks --cluster FluentBit-cluster
```

Step 8: Verify the logs on CloudWatch

In order to verify your Fluent Bit setup, check for the following log groups in the CloudWatch console:

- /ecs/fluent-bit-logs - This is the log group which corresponds to the Fluent Bit daemon container which is running on the container instance.

- `/aws/ecs/FluentBit-cluster.windows-app-task` - This is the log group which corresponds to all the tasks launched for windows-app-task task definition family inside FluentBit-cluster cluster.

`task-out.FIRST_TASK_ID.sample-container` - This log stream contains all the logs generated by the first instance of the task in the sample-container task container.

`task-out.SECOND_TASK_ID.sample-container` - This log stream contains all the logs generated by the second instance of the task in the sample-container task container.

The `task-out.TASK_ID.sample-container` log stream has fields similar to the following:

```
{  
    "source": "stdout",  
    "ecs_task_arn": "arn:aws:ecs:region:0123456789012:task/FluentBit-cluster/13EXAMPLE",  
    "container_name": "/ecs-windows-app-task-1-sample-container-cEXAMPLE",  
    "ecs_cluster": "FluentBit-cluster",  
    "ecs_container_name": "sample-container",  
    "ecs_task_definition_version": "1",  
    "container_id": "61f5e6EXAMPLE",  
    "log": "10",  
    "ecs_task_definition_family": "windows-app-task"  
}
```

To verify the Fluent Bit setup

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**. Make sure that you're in the Region where you deployed Fluent Bit to your containers.

In the list of log groups in the AWS Region, you should see the following:

- `/ecs/fluent-bit-logs`
- `/aws/ecs/FluentBit-cluster.windows-app-task`

If you see these log groups, the Fluent Bit setup is verified.

Step 9: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

To clean up the tutorial resources

1. Stop the `windows-simple-task` task and the `ecs-fluent-bit` task. For more information, see [the section called "Stopping tasks using the console" \(p. 434\)](#).
2. Run the following command to delete the `/ecs/fluent-bit-logs` log group. For more information, about deleting log groups see [delete-log-group](#) in the *AWS Command Line Interface Reference*.

```
aws logs delete-log-group --log-group-name /ecs/fluent-bit-logs  
aws logs delete-log-group --log-group-name /aws/ecs/FluentBit-cluster.windows-app-task
```

3. Run the following command to terminate the instance.

```
aws ec2 terminate-instances --instance-ids instance-id
```

- Run the following commands to delete the IAM roles.

```
aws iam delete-role --role-name ecsInstanceRole  
aws iam delete-role --role-name fluentTaskRole
```

- Run the following command to delete the Amazon ECS cluster.

```
aws ecs delete-cluster --cluster FluentBit-cluster
```

Fargate AWS CLI capacity provider examples

FSx for Windows File Server provides fully managed Microsoft Windows file servers, that are backed by a fully native Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#) in the *FSx for Windows File Server User Guide*.

Creating a new cluster that uses Fargate capacity providers

When a new Amazon ECS cluster is created, you can specify one or more capacity providers to associate with the cluster. The capacity providers are used to define a capacity provider strategy that determines the infrastructure that your tasks run on.

If you set up using the AWS Management Console and use the **Networking only** cluster template, the **FARGATE** and **FARGATE_SPOT** capacity providers are associated with the cluster automatically. For more information, see [Creating a cluster using the classic console \(p. 879\)](#).

To create an Amazon ECS cluster using Fargate capacity providers (AWS CLI)

Use the following command to create a new cluster and associate both the Fargate and Fargate Spot capacity providers with it.

- [create-cluster](#) (AWS CLI)

```
aws ecs create-cluster \  
  --cluster-name FargateCluster \  
  --capacity-providers FARGATE FARGATE_SPOT \  
  --region us-west-2
```

Adding Fargate capacity providers to an existing cluster

You can update the pool of available capacity providers for an existing Amazon ECS cluster by using the `PutClusterCapacityProviders` API operation.

Adding either the Fargate or Fargate Spot capacity providers to an existing cluster isn't supported in the AWS Management Console. You must either create a new Fargate cluster in the console or add the Fargate or Fargate Spot capacity providers to the existing cluster using the Amazon ECS API or AWS CLI.

To add the Fargate capacity providers to an existing cluster (AWS CLI)

Use the following command to add the Fargate and Fargate Spot capacity providers to an existing cluster. If the specified cluster has existing capacity providers associated with it, you must specify all existing capacity providers in addition to any new ones you want to add. Any existing capacity providers that are associated with a cluster that are omitted from a PutClusterCapacityProviders API call will be disassociated from the cluster. You can only disassociate an existing capacity provider from a cluster if it's not being used by any existing tasks. These same rules apply to the cluster's default capacity provider strategy. If the cluster has an existing default capacity provider strategy defined, it must be included in the PutClusterCapacityProviders API call. Otherwise, it is overwritten.

- [put-cluster-capacity-providers](#) (AWS CLI)

```
aws ecs put-cluster-capacity-providers \
    --cluster FargateCluster \
    --capacity-providers FARGATE
FARGATE_SPOT existing_capacity_provider1 existing_capacity_provider2 \
    --default-capacity-provider-strategy existing_default_capacity_provider_strategy \
    --region us-west-2
```

Running tasks using a Fargate capacity provider

You can run a task or create a service using either the Fargate or Fargate Spot capacity providers by specifying a capacity provider strategy. If no capacity provider strategy is provided, the cluster's default capacity provider strategy is used.

Running a task that uses the Fargate or Fargate Spot capacity providers is supported in the AWS Management Console. You must add the Fargate or Fargate Spot capacity providers to cluster's default capacity provider strategy if using the AWS Management Console. When using the Amazon ECS API or AWS CLI you can specify either a capacity provider strategy or use the cluster's default capacity provider strategy.

To run a task using a Fargate capacity provider (AWS CLI)

Use the following command to run a task using the Fargate and Fargate Spot capacity providers.

- [run-task](#) (AWS CLI)

```
aws ecs run-task \
    --capacity-provider-strategy capacityProvider=FARGATE,weight=1
    capacityProvider=FARGATE_SPOT,weight=1 \
    --cluster FargateCluster \
    --task-definition task-def-family:revision \
    --network-configuration
    "awsvpcConfiguration={subnets=[string, string],securityGroups=[string, string],assignPublicIp=string}"
    \
    --count integer \
    --region us-west-2
```

Note

When running standalone tasks using Fargate Spot it's important to note that the task might be interrupted before it can complete and exit. Therefore, it's important that you code your application to gracefully exit within two minutes when it receives a SIGTERM signal and can be resumed. For more information, see [Handling Fargate Spot termination notices \(p. 223\)](#).

Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI

The following tutorial shows how to create an Amazon ECS task that runs a Windows container that has credentials to access Active Directory with the AWS CLI. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

Topics

- [Prerequisites \(p. 743\)](#)
- [Step 1: Create and configure the gMSA account on Active Directory Domain Services \(AD DS\) \(p. 744\)](#)
- [Step 2: Upload Credentials to Secrets Manager \(p. 745\)](#)
- [Step 3: Modify your CredSpec JSON to include domainless gMSA information \(p. 746\)](#)
- [Step 4: Upload CredSpec to Amazon S3 \(p. 747\)](#)
- [Step 5: \(Optional\) Create an Amazon ECS cluster \(p. 747\)](#)
- [Step 6: Create an IAM role for container instances \(p. 748\)](#)
- [Step 7: Create a custom task execution role \(p. 748\)](#)
- [Step 8: Create a task role for Amazon ECS Exec \(p. 749\)](#)
- [Step 9: Register a task definition that uses domainless gMSA \(p. 750\)](#)
- [Step 10: Register a Windows container instance to the cluster \(p. 751\)](#)
- [Step 11: Verify the container instance \(p. 751\)](#)
- [Step 12: Run a Windows task \(p. 752\)](#)
- [Step 13: Verify the container has gMSA credentials \(p. 753\)](#)
- [Step 14: Clean up \(p. 753\)](#)
- [Debugging Amazon ECS domainless gMSA for Windows containers \(p. 754\)](#)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in [Set up to use Amazon ECS \(p. 10\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.
- 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](#).
- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
 - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see [Getting Started with AWS Managed Microsoft AD](#) in the [AWS Directory Service Administration Guide](#).

- An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see [AWS Direct Connect](#).
- You have a VPC and subnets that can resolve the Active Directory domain name.
- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

gMSA data storage options by Operating System

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

Step 1: Create and configure the gMSA account on Active Directory Domain Services (AD DS)

Create and configure a gMSA account on the Active Directory domain.

1. Generate a Key Distribution Service root key

Note

If you are using AWS Directory Service, then you can skip this step.

The KDS root key and gMSA permissions are configured with your AWS managed Microsoft AD.

If you have not already created a gMSA Service Account in your domain, you'll need to first generate a Key Distribution Service (KDS) root key. The KDS is responsible for creating, rotating, and releasing the gMSA password to authorized hosts. When the ccg.exe needs to retrieve gMSA credentials, it contact KDS to retrieve the current password.

To check if the KDS root key has already been created, run the following PowerShell cmdlet with domain admin privileges on a domain controller using the ActiveDirectory PowerShell module. For more information about the module, see [ActiveDirectory Module](#) on the Microsoft Learn website.

```
PS C:\> Get-KdsRootKey
```

If the command returns a key ID, you can skip the rest of this step. Otherwise, create the KDS root key by running the following command:

```
PS C:\> Add-KdsRootKey -EffectiveImmediately
```

Although the argument `EffectiveImmediately` to the command implies the key is effective immediately, you need to wait 10 hours before the KDS root key is replicated and available for use on all domain controllers.

2. Create the gMSA account

To create the gMSA account and allow the `ccg.exe` to retrieve the gMSA password, run the following PowerShell commands from a Windows Server or client with access to the domain. Replace `ExampleAccount` with the name that you want for your gMSA account.

- a.

```
PS C:\> Install-WindowsFeature RSAT-AD-PowerShell
```
- b.

```
PS C:\> New-ADGroup -Name "ExampleAccount Authorized Hosts" -SamAccountName "ExampleAccountHosts" -GroupScope DomainLocal
```
- c.

```
PS C:\> New-ADServiceAccount -Name "ExampleAccount" -DnsHostName "contoso" -ServicePrincipalNames "host/ExampleAccount", "host/contoso" -PrincipalsAllowedToRetrieveManagedPassword "ExampleAccountHosts"
```
- d. Create a user with a permanent password that doesn't expire. These credentials are stored in AWS Secrets Manager and used by each task to join the domain.

```
PS C:\> New-ADUser -Name "ExampleAccount" -AccountPassword (ConvertTo-SecureString -AsPlainText "Test123" -Force) -Enabled 1 -PasswordNeverExpires 1
```
- e.

```
PS C:\> Add-ADGroupMember -Identity "ExampleAccountHosts" -Members "ExampleAccount"
```
- f. Install the PowerShell module for creating CredSpec objects in Active Directory and output the CredSpec JSON.

```
PS C:\> Install-PackageProvider -Name NuGet -Force
```



```
PS C:\> Install-Module CredentialSpec
```
- g.

```
PS C:\> New-CredentialSpec -AccountName ExampleAccount
```

3. Copy the JSON output from the previous command into a file called `gmsa-cred-spec.json`. This is the CredSpec file. It is used in Step 3, [Step 3: Modify your CredSpec JSON to include domainless gMSA information \(p. 746\)](#).

Step 2: Upload Credentials to Secrets Manager

Copy the Active Directory credentials into a secure credential storage system, so that each task retrieves it. This is the domainless gMSA method. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

- Run the following AWS CLI command and replace the username, password, and domain name to match your environment. Keep the ARN of the secret to use in the next step, [Step 3: Modify your CredSpec JSON to include domainless gMSA information \(p. 746\)](#)

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws secretsmanager create-secret \
--name gmsa-plugin-input \
--description "Amazon ECS - gMSA Portable Identity." \
--secret-string "{\"username\":\"ExampleAccount\",\"password\":\"Test123\",\"domainName\":\"contoso.com\"}"
```

Step 3: Modify your CredSpec JSON to include domainless gMSA information

Before uploading the CredSpec to one of the storage options, add information to the CredSpec with the ARN of the secret in Secrets Manager from the previous step. For more information, see [Additional credential spec configuration for non-domain-joined container host use case](#) on the Microsoft Learn website.

- Add the following information to the CredSpec file inside the ActiveDirectoryConfig. Replace the ARN with the secret in Secrets Manager from the previous step.

Note that the PluginGUID value must match the GUID in the following example snippet and is required.

```
"HostAccountConfig": {
    "PortableCcgVersion": "1",
    "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
    "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input\"}"
}
```

You can also use a secret in SSM Parameter Store by using the ARN in this format:
\"arn:aws:ssm:**aws-region:111122223333:parameter/gmsa-plugin-input**\".

- After you modify the CredSpec file, it should look like the following example:

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-4066351383-705263209-1606769140",
        "MachineAccountName": "ExampleAccount",
        "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
        "DnsTreeName": "contoso",
        "DnsName": "contoso",
        "NetBiosName": "contoso"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [

```

```
{  
    "Name": "ExampleAccount",  
    "Scope": "contoso"  
},  
{  
    "Name": "ExampleAccount",  
    "Scope": "contoso"  
}  
],  
"HostAccountConfig": {  
    "PortableCcgVersion": "1",  
    "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",  
    "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input\"}"  
}  
}
```

Step 4: Upload CredSpec to Amazon S3

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. Copy the CredSpec file to the computer or environment that you are running AWS CLI commands in.
2. Run the following AWS CLI command to upload the CredSpec to Amazon S3. Replace MyBucket with the name of your Amazon S3 bucket. You can store the file as an object in any bucket and location, but you must allow access to that bucket and location in the policy that you attach to the task execution role.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws s3 cp gmsa-cred-spec.json \  
s3://MyBucket/ecs-domainless-gmsa-credspec
```

Step 5: (Optional) Create an Amazon ECS cluster

By default, your account has an Amazon ECS cluster named default. This cluster is used by default in the AWS CLI, SDKs, and AWS CloudFormation. You can use additional clusters to group and organize tasks and infrastructure, and assign defaults for some configuration.

You can create a cluster from the AWS Management Console, AWS CLI, SDKs, or AWS CloudFormation. The settings and configuration in the cluster don't affect gMSA.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

```
$ aws ecs create-cluster --cluster-name windows-domainless-gmsa-cluster
```

Important

If you choose to create your own cluster, you must specify `--cluster` `clusterName` for each command that you intend to use with that cluster.

Step 6: Create an IAM role for container instances

A *container instance* is a host computer to run containers in ECS tasks, for example Amazon EC2 instances. Each container instance registers to an Amazon ECS cluster. Before you launch Amazon EC2 instances and register them to a cluster, you must create an IAM role for your container instances to use.

To create the container instance role, see [Amazon ECS container instance IAM role \(p. 638\)](#). The default `ecsInstanceRole` has sufficient permissions to complete this tutorial.

Step 7: Create a custom task execution role

Amazon ECS can use a different IAM role for the permissions needed to start each task, instead of the container instance role. This role is the *task execution role*. We recommend creating a task execution role with only the permissions required for ECS to run the task, also known as *least-privilege permissions*. For more information about the principle of least privilege, see [SEC03-BP02 Grant least privilege access](#) in the [AWS Well-Architected Framework](#).

1. To create a task execution role, see [Creating the task execution \(ecsTaskExecutionRole\) role \(p. 627\)](#). The default permissions allow the container instance to pull container images from Amazon Elastic Container Registry and `stdout` and `stderr` from your applications to be logged to Amazon CloudWatch Logs.

Because the role needs custom permissions for this tutorial, you can give the role a different name than `ecsTaskExecutionRole`. This tutorial uses `ecsTaskExecutionRole` in further steps.

2. Add the following permissions by creating a custom policy, either an inline policy that only exists in for this role, or a policy that you can reuse. Replace the ARN for the Resource in the first statement with the Amazon S3 bucket and location, and the second Resource with the ARN of the secret in Secrets Manager.

If you encrypt the secret in Secrets Manager with a custom key, you must also allow `kms:Decrypt` for the key.

If you use SSM Parameter Store instead of Secrets Manager, you must allow `ssm:GetParameter` for the parameter, instead of `secretsmanager:GetSecretValue`.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetObject"  
      ],  
      "Resource": "arn:aws:s3:::MyBucket/ecs-domainless-gmsa-credspec/gmsa-cred-spec.json"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "secretsmanager:GetSecretValue"  
      ],  
      "Resource": "arn:aws:secretsmanager:aws-region:111122223333:secret:gmsa-plugin-input"  
    }  
  ]  
}
```

Step 8: Create a task role for Amazon ECS Exec

This tutorial uses Amazon ECS Exec to verify functionality by running a command inside a running task. To use ECS Exec, the service or task must turn on ECS Exec and the task role (but not the task execution role) must have `ssmmessages` permissions. For the required IAM policy, see [IAM permissions required for ECS Exec \(p. 757\)](#).

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

To create a task role using the AWS CLI, follow these steps.

1. Create a file called `ecs-tasks-trust-policy.json` with the following contents:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "ecs-tasks.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

2. Create an IAM role. You can replace the name `ecs-exec-demo-task-role` but keep the name for following steps.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws iam create-role --role-name ecs-exec-demo-task-role \  
--assume-role-policy-document file://ecs-tasks-trust-policy.json
```

You can delete the file `ecs-tasks-trust-policy.json`.

3. Create a file called `ecs-exec-demo-task-role-policy.json` with the following contents:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ssmmessages:CreateControlChannel",  
                "ssmmessages:CreateDataChannel",  
                "ssmmessages:OpenControlChannel",  
                "ssmmessages:OpenDataChannel"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

4. Create an IAM policy and attach it to the role from the previous step.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws iam put-role-policy \
--role-name ecs-exec-demo-task-role \
--policy-name ecs-exec-demo-task-role-policy \
--policy-document file://ecs-exec-demo-task-role-policy.json
```

You can delete the file `ecs-exec-demo-task-role-policy.json`.

Step 9: Register a task definition that uses domainless gMSA

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. Create a file called `windows-gmsa-domainless-task-def.json` with the following contents:

```
{
  "family": "windows-gmsa-domainless-task",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "memory": 1024,
      "essential": true,
      "credentialSpecs": [
        "credentialspecdomainless:arn:aws:s3:::ecs-domainless-gmsa-credspec/
gmsa-cred-spec.json"
      ],
      "entryPoint": [
        "powershell",
        "-Command"
      ],
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value '<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>' -Force ; C:\\ServiceMonitor.exe w3svc"
      ],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80,
          "hostPort": 8080
        }
      ]
    },
    {
      "taskRoleArn": "arn:aws:iam::111122223333:role/ecs-exec-demo-task-role",
      "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
    }
  ]
}
```

2. Register the task definition by running the following command:

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws ecs register-task-definition \
--cli-input-json file://windows-gmsa-domainless-task-def.json
```

Step 10: Register a Windows container instance to the cluster

Launch an Amazon EC2 Windows instance and run the ECS container agent to register it as a container instance in the cluster. ECS runs tasks on the container instances that are registered to the cluster that the tasks are started in.

1. To launch an Amazon EC2 Windows instance that is configured for Amazon ECS in the AWS Management Console, see [Launching an Amazon ECS Windows container instance \(p. 326\)](#). Stop at the step for *user data*.
2. For gMSA, the user data must set the environment variable ECS_GMSA_SUPPORTED before starting the ECS container agent.

For ECS Exec, the agent must start with the argument -EnableTaskIAMRole.

To secure the instance IAM role by preventing tasks from reaching the EC2 IMDS web service to retrieve the role credentials, add the argument -AwsVpcBlockIMDS. This only applies to tasks that use the awsvpc network mode.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_GMSA_SUPPORTED", $TRUE, "Machine")
Import-Module ECSTools
Initialize-ECSAgent -Cluster windows-domainless-gmsa-cluster -EnableTaskIAMRole -
AwsVpcBlockIMDS
</powershell>
```

3. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

Step 11: Verify the container instance

You can verify that there is a container instance in the cluster using the AWS Management Console. However, gMSA needs additional features that are indicated as *attributes*. These attributes aren't visible in the AWS Management Console, so this tutorial uses the AWS CLI.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. List the container instances in the cluster. Container instances have an ID that is different from the ID of the EC2 instance.

```
$ aws ecs list-container-instances
```

Output:

```
{  
    "containerInstanceArns": [  
        "arn:aws:ecs:aws-region:111122223333:container-  
        instance/default/MyContainerInstanceID"  
    ]  
}
```

For example, 526bd5d0ced448a788768334e79010fd is a valid container instance ID.

2. Use the container instance ID from the previous step to get the details for the container instance. Replace MyContainerInstanceID with the ID.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws ecs describe-container-instances \  
    --container-instances MyContainerInstanceID
```

Note that the output is very long.

3. Verify that the attributes list has an object with the key called name and a value ecs.capability.gmsa-domainless. The following is an example of the object.

Output:

```
{  
    "name": "ecs.capability.gmsa-domainless"  
}
```

Step 12: Run a Windows task

Run an Amazon ECS task. If there is only 1 container instance in the cluster, you can use run-task. If there are many different container instances, it might be easier to use start-task and specify the container instance ID to run the task on, than to add placement constraints to the task definition to control what type of container instance to run this task on.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
aws ecs run-task --task-definition windows-gmsa-domainless-task \  
    --enable-execute-command
```

Note the task ID that is returned by the command.

2. Run the following command to verify that the task has started. This command waits and doesn't return the shell prompt until the task starts. Replace MyTaskID with the task ID from the previous step.

```
$ aws ecs wait tasks-running --task MyTaskID
```

Step 13: Verify the container has gMSA credentials

Verify that the container in the task has a Kerberos token. gMSA

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws ecs execute-command \
--task MyTaskID \
--container windows_sample_app \
--interactive \
--command powershell.exe
```

The output will be a PowerShell prompt.

2. Run the following command in the PowerShell terminal inside the container.

```
PS C:\> klist get ExampleAccount$
```

In the output, note the Principal is the one that you created previously.

Step 14: Clean up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. Stop the task. Replace MyTaskID with the task ID from step 12, [Step 12: Run a Windows task \(p. 752\)](#).

```
$ aws ecs stop-task --task MyTaskID
```

2. Terminate the Amazon EC2 instance. Afterwards, the container instance in the cluster will be deleted automatically after one hour. For more information, see [Container instance lifecycle \(p. 252\)](#).

You can find and terminate the instance by using the Amazon EC2 console. Or, you can run the following command. To run the command, find the EC2 instance ID in the output of the aws ecs describe-container-instances command from step 1, [Step 11: Verify the container instance \(p. 751\)](#). i-10a64379 is an example of an EC2 instance ID.

```
$ aws ec2 terminate-instances --instance-ids MyInstanceId
```

3. Delete the CredSpec file in Amazon S3. Replace MyBucket with the name of your Amazon S3 bucket.

```
$ aws s3api delete-object --bucket MyBucket --key ecs-domainless-gmsa-credspec/gmsa-credspec.json
```

4. Delete the secret from Secrets Manager. If you used SSM Parameter Store instead, delete the parameter.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws secretsmanager delete-secret --secret-id gmsa-plugin-input \  
--force-delete-without-recovery
```

5. Deregister and delete the task definition. By deregistering the task definition, you mark it as inactive so it can't be used to start new tasks. Then, you can delete the task definition.

- a. Deregister the task definition by specifying the version. ECS automatically makes versions of task definitions, that are numbered starting from 1. You refer to the versions in the same format as the labels on container images, such as :1.

```
$ aws ecs deregister-task-definition --task-definition windows-gmsa-domainless-  
task:1
```

- b. Delete the task definition.

```
$ aws ecs delete-task-definitions --task-definition windows-gmsa-domainless-task:1
```

6. (Optional) Delete the ECS cluster, if you created a cluster.

```
$ aws ecs delete-cluster --cluster windows-domainless-gmsa-cluster
```

Debugging Amazon ECS domainless gMSA for Windows containers

Amazon ECS task status

ECS tries to start a task exactly once. Any task that has an issue is stopped, and set to the status STOPPED. There are two common types of issues with tasks. First, tasks that couldn't be started. Second, tasks where the application has stopped inside one of the containers. In the AWS Management Console, look at the **Stopped reason** field of the task for the reason why the task was stopped. In the AWS CLI, describe the task and look at the stoppedReason. For steps in the AWS Management Console and AWS CLI, see [Checking stopped tasks for errors \(p. 766\)](#).

Windows Events

Windows Events for gMSA in containers are logged in the Microsoft-Windows-Containers-CCG log file and can be found in the Event Viewer in the section Applications and Services in Logs \Microsoft\Windows\Containers-CCG\Admin. For more debugging tips, see [Troubleshoot gMSAs for Windows containers](#) on the Microsoft Learn website.

ECS agent gMSA plugin

Logging for gMSA plugin for the ECS agent on the Windows container instance is in the following directory, C:/ProgramData/Amazon/gmsa-plugin/. Look in this log to see if the domainless user credentials were downloaded from the storage location, such as Secrets Manager, and that the credential format was read correctly.

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

- [Using Amazon ECS Exec for debugging \(p. 755\)](#)
- [Troubleshooting ECS Anywhere issues \(p. 764\)](#)
- [Checking stopped tasks for errors \(p. 766\)](#)
- [CannotPullContainer task errors \(p. 767\)](#)
- [Service event messages \(p. 770\)](#)
- [Invalid CPU or memory value specified \(p. 776\)](#)
- [CannotCreateContainerError: API error \(500\): devmapper \(p. 777\)](#)
- [Troubleshooting service load balancers \(p. 778\)](#)
- [Troubleshooting service auto scaling \(p. 779\)](#)
- [Using Docker debug output \(p. 780\)](#)
- [Amazon ECS Log File Locations \(p. 781\)](#)
- [Amazon ECS logs collector \(p. 784\)](#)
- [Agent introspection diagnostics \(p. 786\)](#)
- [Docker diagnostics \(p. 787\)](#)
- [AWS Fargate throttling quotas \(p. 789\)](#)
- [API failure reasons \(p. 790\)](#)
- [Troubleshooting IAM Roles for Tasks \(p. 796\)](#)

Using Amazon ECS Exec for debugging

With Amazon ECS Exec, you can directly interact with containers without needing to first interact with the host container operating system, open inbound ports, or manage SSH keys. You can use ECS Exec to run commands in or get a shell to a container running on an Amazon EC2 instance or on AWS Fargate. This makes it easier to collect diagnostic information and quickly troubleshoot errors. For example, in a development context, you can use ECS Exec to easily interact with various process in your containers and troubleshoot your applications. And, in production scenarios, you can use it to gain break-glass access to your containers to debug issues.

You can run commands in a running Linux or Windows container using ECS Exec from the Amazon ECS API, AWS Command Line Interface (AWS CLI), AWS SDKs, or the AWS Copilot CLI. For details on using ECS Exec, as well as a video walkthrough, using the AWS Copilot CLI, see the [Copilot Github documentation](#).

You can also use ECS Exec to maintain stricter access control policies and audit container access. By selectively turning on this feature, you can control who can run commands and on which tasks they can run those commands. With a log of each command and their output, you can use ECS Exec to audit which tasks were run and you can use CloudTrail to audit who accessed a container.

Architecture

ECS Exec makes use of AWS Systems Manager (SSM) Session Manager to establish a connection with the running container and uses AWS Identity and Access Management (IAM) policies to control access to

running commands in a running container. This is made possible by bind-mounting the necessary SSM agent binaries into the container. The Amazon ECS or AWS Fargate agent is responsible for starting the SSM core agent inside the container alongside your application code. For more information, see [Systems Manager Session Manager](#).

You can audit which user accessed the container using AWS CloudTrail and log each command (and their output) to Amazon S3 or Amazon CloudWatch Logs. To encrypt data between the local client and container with your own encryption key, you must provide the AWS Key Management Service (AWS KMS) key.

Considerations for using ECS Exec

For this topic, you should be familiar with the following aspects involved with using ECS Exec:

- ECS Exec is supported for tasks that run on the following infrastructure:
 - Linux containers on Amazon EC2 on any Amazon ECS-optimized AMI, including Bottlerocket
 - Linux and Windows containers on external instances (ECS Anywhere)
 - Linux and Windows containers on AWS Fargate
 - Windows containers on Amazon EC2 on the following Windows Amazon ECS-optimized AMIs (with the container agent version 1.56 or later):
 - Amazon ECS-optimized Windows Server 2022 Full AMI
 - Amazon ECS-optimized Windows Server 2022 Core AMI
 - Amazon ECS-optimized Windows Server 2019 Full AMI
 - Amazon ECS-optimized Windows Server 2019 Core AMI
 - Amazon ECS-optimized Windows Server 20H2 Core AMI
- ECS Exec is not currently supported using the AWS Management Console.
- If you are using interface Amazon VPC endpoints with Amazon ECS, you must create the interface Amazon VPC endpoints for the Systems Manager Session Manager (`ssmmessages`). For more information about Systems Manager VPC endpoints, see [Use AWS PrivateLink to set up a VPC endpoint for Session Manager](#) in the *AWS Systems Manager User Guide*.
- If you are using interface Amazon VPC endpoints with Amazon ECS, and you are using AWS KMS key for encryption, then you must create the interface Amazon VPC endpoint for AWS KMS key. For more information, see [Connecting to AWS KMS key through a VPC endpoint](#) in the *AWS Key Management Service Developer Guide*.
- You can't turn on ECS Exec for existing tasks. It can only be turned on for new tasks.
- When a user runs commands on a container using ECS Exec, these commands are run as the `root` user. The SSM agent and its child processes run as root even when you specify a user ID for the container.
- The ECS Exec session has an idle timeout time of 20 minutes. This value cannot be changed.
- The SSM agent requires that the container file system is able to be written to in order to create the required directories and files. Therefore, making the root file system read-only using the `readonlyRootFilesystem` task definition parameter, or any other method, isn't supported.
- Users can run all of the commands that are available within the container context. The following actions might result in orphaned and zombie processes: terminating the main process of the container, terminating the command agent, and deleting dependencies. To cleanup zombie processes, we recommend adding the `initProcessEnabled` flag to your task definition.
- While starting SSM sessions outside of the `execute-command` action is possible, this results in the sessions not being logged and being counted against the session limit. We recommend limiting this access by denying the `ssm:start-session` action using an IAM policy. For more information, see [Limiting access to the Start Session action \(p. 763\)](#).
- ECS Exec will use some CPU and memory. You'll want to accommodate for that when specifying the CPU and memory resource allocations in your task definition.

- You must be using AWS CLI version 1.22.3 or later or AWS CLI version 2.3.6 or later. For information about how to update the AWS CLI, see [Installing or updating the latest version of the AWS CLI](#) in the [AWS Command Line Interface User Guide Version 2](#).
- You cannot use ECS Exec when you use `run-task` to launch a task on a cluster that uses managed scaling with asynchronous placement (launch a task with no instance).
- You cannot run ECS Exec against Microsoft Nano Server containers. For more information about Nano Server containers, see [Nano Server](#) on the Docker web site.
- You can have only one ECS Exec session per process ID (PID) namespace. If you are [sharing a PID namespace in a task](#), you cannot start multiple sessions into multiple containers.

Prerequisites for using ECS Exec

Before you start using ECS Exec, make sure you that you have completed these actions:

- Install and configure the AWS CLI. For more information, see [AWS CLI](#).
- Install Session Manager plugin for the AWS CLI. For more information, see [Install the Session Manager plugin for the AWS CLI](#).
- ECS Exec has version requirements depending on whether your tasks are hosted on Amazon EC2 or AWS Fargate:
 - If you're using Amazon EC2, you must use an Amazon ECS optimized AMI that was released after January 20th, 2021, with an agent version of 1.50.2 or greater. For more information, see [Amazon ECS optimized AMIs](#).
 - If you're using AWS Fargate, you must use platform version 1.4.0 or higher (Linux) or 1.0.0 (Windows). For more information, see [AWS Fargate platform versions](#).

Using ECS Exec

IAM permissions required for ECS Exec

The ECS Exec feature requires a task IAM role to grant containers the permissions needed for communication between the managed SSM agent (execute-command agent) and the SSM service. For more information, see [Amazon ECS task IAM role](#). You should add the following permissions to a task IAM role and include the task IAM role in your task definition. For more information, see [Adding and Removing IAM Policies](#).

Use the following policy for your task IAM role to add the required SSM permissions. For information about the task IAM role, see [the section called "Task IAM role" \(p. 631\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ssmmessages:CreateControlChannel",  
                "ssmmessages:CreateDataChannel",  
                "ssmmessages:OpenControlChannel",  
                "ssmmessages:OpenDataChannel"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Optional task definition changes

If you set the task definition parameter `initProcessEnabled` to `true`, this starts the init process inside the container, which removes any zombie SSM agent child processes found. The following provides an example.

```
{
  "taskRoleArn": "ecsTaskRole",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "EC2",
    "FARGATE"
  ],
  "executionRoleArn": "ecsTaskExecutionRole",
  "memory": ".5 gb",
  "cpu": ".25 vcpu",
  "containerDefinitions": [
    {
      "name": "amazon-linux",
      "image": "amazonlinux:latest",
      "essential": true,
      "command": ["sleep", "3600"],
      "linuxParameters": {
        "initProcessEnabled": true
      }
    }
  ],
  "family": "ecs-exec-task"
}
```

Turning on ECS Exec for your tasks and services

You can turn on the ECS Exec feature for your services and standalone tasks by specifying the `--enable-execute-command` flag when using one of the following AWS CLI commands: [create-service](#), [update-service](#), [start-task](#), or [run-task](#).

For example, if you run the following command, the ECS Exec feature is turned on for a newly created service. For more information about creating services, see [create-service](#).

```
aws ecs create-service \
--cluster cluster-name \
--task-definition task-definition-name \
--enable-execute-command \
--service-name service-name \
--desired-count 1
```

After you turn on ECS Exec for a task, you can run the following command to confirm the task is ready to be used. If the `lastStatus` property of the `ExecuteCommandAgent` is listed as `RUNNING` and the `enableExecuteCommand` property is set to `true`, then your task is ready.

```
aws ecs describe-tasks \
--cluster cluster-name \
--tasks task-id
```

The following output snippet is an example of what you might see.

```
{
  "tasks": [
```

```
{  
    ...  
    "containers": [  
        {  
            ...  
            "managedAgents": [  
                {  
                    "lastStartedAt": "2021-03-01T14:49:44.574000-06:00",  
                    "name": "ExecuteCommandAgent",  
                    "lastStatus": "RUNNING"  
                }  
            ]  
        },  
        ...  
        {"enableExecuteCommand": true,  
        ...  
    ]  
}
```

Running commands using ECS Exec

After you have confirmed the ExecuteCommandAgent is running, you can open an interactive shell on your container using the following command. If your task contains multiple containers, you must specify the container name using the `--container` flag. Amazon ECS only supports initiating interactive sessions, so you must use the `--interactive` flag.

The following command will run an interactive `/bin/sh` command against a container named `container-name` for a task with an id of `task-id`.

```
aws ecs execute-command --cluster cluster-name \  
  --task task-id \  
  --container container-name \  
  --interactive \  
  --command "/bin/sh"
```

Logging and Auditing using ECS Exec

Turning on logging and auditing in your tasks and services

Important

For more information about CloudWatch pricing, see [CloudWatch Pricing](#). Amazon ECS also provides monitoring metrics that are provided at no additional cost. For more information, see [Amazon ECS CloudWatch metrics \(p. 547\)](#).

Amazon ECS provides a default configuration for logging commands run using ECS Exec by sending logs to CloudWatch Logs using the `awslogs` log driver that's configured in your task definition. If you want to provide a custom configuration, the AWS CLI supports a `--configuration` flag for both the `create-cluster` and `update-cluster` commands. It's also important to know that the container image requires `script` and `cat` to be installed in order to have command logs uploaded correctly to Amazon S3 or CloudWatch Logs. For more information about creating clusters, see [create-cluster](#).

Note

This configuration only handles the logging of the `execute-command` session. It doesn't affect logging of your application.

The following example creates a service and then logs the output to your CloudWatch Logs LogGroup named `cloudwatch-log-group-name` and your Amazon S3 bucket named `s3-bucket-name`.

You must use an AWS KMS customer managed key to encrypt the log group when you set the CloudWatchEncryptionEnabled option to true. For information about how to encrypt the log group, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](#), in the *Amazon CloudWatch Logs User Guide*.

```
aws ecs create-cluster \
    --cluster-name cluster-name \
    --configuration executeCommandConfiguration="{ \
        kmsKeyId=string, \
        logging=OVERRIDE, \
        logConfiguration={ \
            cloudWatchLogGroupName=cloudwatch-log-group-name, \
            cloudWatchEncryptionEnabled=true, \
            s3BucketName=s3-bucket-name, \
            s3EncryptionEnabled=true, \
            s3KeyPrefix=demo \
        } \
    }"
```

The logging property determines the behavior of the logging capability of ECS Exec:

- NONE: logging is turned off
- DEFAULT: logs are sent to the configured awslogs driver (If the driver isn't configured, then no log is saved.)
- OVERRIDE: logs are sent to the provided Amazon CloudWatch Logs LogGroup, Amazon S3 bucket, or both

IAM permissions required for Amazon CloudWatch Logs or Amazon S3 Logging

To enable logging, the Amazon ECS task role that's referenced in your task definition needs to have additional permissions. These additional permissions can be added as a policy to the task role. They are different depending on if you direct your logs to Amazon CloudWatch Logs or Amazon S3.

Amazon CloudWatch Logs

The following example policy adds the required Amazon CloudWatch Logs permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs>CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/ecs/Cloudwatch-log-group-name:*"
        }
    ]
}
```

}

Amazon S3

The following example policy adds the required Amazon S3 permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetEncryptionConfiguration"
            ],
            "Resource": "arn:aws:s3:::s3-bucket-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::s3-bucket-name/*"
        }
    ]
}
```

IAM permissions required for encryption using your own AWS KMS key (KMS key)

By default, the data transferred between your local client and the container uses TLS 1.2 encryption that AWS provides. To further encrypt data using your own KMS key, you must create a KMS key and add the `kms:Decrypt` permission to your task IAM role. This permission is used by your container to decrypt the data. For more information about creating a KMS key, see [Creating keys](#).

You would add the following inline policy to your task IAM role which requires the AWS KMS permissions. For more information, see [IAM permissions required for ECS Exec \(p. 757\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "kms-key-arn"
        }
    ]
}
```

For the data to be encrypted using your own KMS key, the user or group using the `execute-command` action must be granted the `kms:GenerateDataKey` permission.

The following example policy for your user or group contains the required permission to use your own KMS key. You must specify the Amazon Resource Name (ARN) of your KMS key.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:GenerateDataKey"  
            ],  
            "Resource": "kms-key-arn"  
        }  
    ]  
}
```

Using IAM policies to limit access to ECS Exec

You can limit user access to the execute-command API action by using one or more of the following IAM policy condition keys:

- aws:ResourceTag/*clusterTagKey*
- ecs:ResourceTag/*clusterTagKey*
- aws:ResourceTag/*taskTagKey*
- ecs:ResourceTag/*taskTagKey*
- ecs:container-name
- ecs:cluster
- ecs:task
- ecs:enable-execute-command

With the following example IAM policy, users can run commands in containers that are running within tasks with a tag that has an environment key and development value and in a cluster that's named *cluster-name*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecs:ExecuteCommand",  
                "ecs:DescribeTasks"  
            ],  
            "Resource": "arn:aws:ecs:region:aws-account-id:task/cluster-name/*",  
            "Condition": {  
                "StringEquals": {  
                    "ecs:ResourceTag/environment": "development"  
                }  
            }  
        }  
    ]  
}
```

With the following IAM policy example, users can't use the execute-command API when the container name is production-app.

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ecs:ExecuteCommand"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:container-name": "production-app"
                }
            }
        }
    ]
}

```

With the following IAM policy, users can only launch tasks when ECS Exec is turned off.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs>CreateService",
                "ecs:UpdateService"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:enable-execute-command": "false"
                }
            }
        }
    ]
}

```

Note

Because the execute-command API action contains only task and cluster resources in a request, only cluster and task tags are evaluated.

For more information about IAM policy condition keys, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

Limits access to the Start Session action

While starting SSM sessions on your container outside of ECS Exec is possible, this could potentially result in the sessions not being logged. Sessions started outside of ECS Exec also count against the session quota. We recommend limiting this access by denying the `ssm:start-session` action directly for your Amazon ECS tasks using an IAM policy. You can deny access to all Amazon ECS tasks or to specific tasks based on the tags used.

The following is an example IAM policy that denies access to the `ssm:start-session` action for tasks in all Regions with a specified cluster name. You can optionally include a wildcard with the `cluster-name`.

```

{
    "Version": "2012-10-17",
    "Statement": [

```

```
{  
    "Effect": "Deny",  
    "Action": "ssm:StartSession",  
    "Resource": "arn:aws:ecs:*:111122223333:task/cluster-name/*"  
}  
]  
}
```

The following is an example IAM policy that denies access to the `ssm:start-session` action on resources in all Regions tagged with tag key `Task-Tag-Key` and tag value `Exec-Task`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "ssm:StartSession",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/Task-Tag-KeyExec-Task"  
                }  
            }  
        }  
    ]  
}
```

Troubleshooting issues with ECS Exec

The following are troubleshooting notes to help diagnose why you may be getting an error when using ECS Exec.

Verify using the Amazon ECS Exec Checker

The Amazon ECS Exec Checker script provides a way to verify and validate that your Amazon ECS cluster and task have met the prerequisites for using the ECS Exec feature. The Exec Checker script verifies both your AWS CLI environment and cluster and tasks are ready for ECS Exec, by calling various APIs on your behalf. The tool requires the latest version of the AWS CLI and that the `jq` is available. For more information, see [Amazon ECS Exec Checker](#) on GitHub.

Error when calling execute-command

If a `The execute command failed` error occurs, the following are possible causes.

- The task does not have the required permissions. Verify that the task definition used to launch your task has a task IAM role defined and that the role has the required permissions. For more information, see [IAM permissions required for ECS Exec \(p. 757\)](#).
- The SSM agent is not installed or is not running
- There is an interface Amazon VPC endpoint for Amazon ECS, but there is not one for Systems Manager Session Manager

Troubleshooting ECS Anywhere issues

ECS Anywhere provides support for registering an *external instance* such as a on-premises server or virtual machine (VM) to your Amazon ECS cluster. The following are common issues that you might encounter and general troubleshooting recommendations for them.

Topics

- [External instance registration issues \(p. 765\)](#)
- [External instance network issues \(p. 765\)](#)
- [Issues running tasks on your external instance \(p. 765\)](#)

External instance registration issues

When registering an external instance with your Amazon ECS cluster, the following requirements must be met:

- An Systems Manager activation, which consists of an *activation ID* and *activation code*, must be retrieved. You use it to register the external instance as a Systems Manager managed instance. When a Systems Manager activation is requested, you can specify a registration limit and expiration date. The registration limit specifies the maximum number of instances that can be registered using the activation. For it, the default value is 1 instance. The expiration date specifies when the activation expires. The default value is 24 hours. If the Systems Manager activation that you're using to register your external instance isn't valid, request a new one. For more information, see [Registering an external instance to a cluster \(p. 342\)](#).
- An IAM policy is used to provide your external instance the permissions that it needs to communicate with AWS APIs. If this managed policy isn't created properly and doesn't contain the required permissions, external instance registration fails. For more information, see [Required IAM permissions for external instances \(p. 340\)](#).
- Amazon ECS provides an installation script that installs Docker, the Amazon ECS container agent, and the Systems Manager Agent on your external instance. If the installation script fails, it's likely that the script can't be run again on the same instance without an error occurring. If this happens, follow the cleanup process to clear AWS resources from the instance so you can run the installation script again. For more information, see [Deregistering an external instance \(p. 346\)](#).

Note

Be aware that, if the installation script successfully requested and used the Systems Manager activation, running the installation script a second time uses the Systems Manager activation again. This might in turn cause you to reach the registration limit for that activation. If this limit is reached, you must create a new activation.

- When running the installation script on an external instance for GPU workloads, if the NVIDIA driver is not detected or configured properly an error will occur. The installation script uses the `nvidia-smi` command to confirm the existence of the NVIDIA driver.

External instance network issues

To communicate any changes, your external instance requires a network connection to AWS. If your external instance loses its network connection to AWS, tasks that are running on your instances continue to run anyway unless stopped manually. After the connection to AWS is restored, the AWS credentials that are used by the Amazon ECS container agent and Systems Manager Agent on the external instance renew automatically. For more information about the AWS domains that are used for communication between your external instance and AWS, see [Networking with ECS Anywhere \(p. 339\)](#).

Issues running tasks on your external instance

If your tasks or containers fail to run on your external instance, the most common causes are either network or permission related. If your containers are pulling their images from Amazon ECR or are configured to send container logs to CloudWatch Logs, your task definition must specify a valid task execution IAM role. Without a valid task execution IAM role, your containers will fail to start. For more

information, see [Conditional IAM permissions \(p. 342\)](#). For more information about network related issues, see [External instance network issues \(p. 765\)](#).

Important

Amazon ECS provides the Amazon ECS logs collection tool. You can use it to collect logs from your external instances for troubleshooting purposes. For more information, see [Amazon ECS logs collector \(p. 784\)](#).

Checking stopped tasks for errors

If you have trouble starting a task, your task might be stopping because of application or configuration errors. For example, you run the task and the task displays a PENDING status and then disappears. You can view stopped task errors like this in the Amazon ECS console by viewing the stopped task and inspecting it for error messages.

If your task definition uses the awslogs log driver, the application logs that are written to Amazon CloudWatch Logs are displayed on the **Logs** tab in the Amazon ECS console as long as the stopped task appears.

If your task was created by an Amazon ECS service, the actions that Amazon ECS takes to maintain the service are published in the service events. You can view the events in the AWS Management Console, AWS CLI, AWS SDKs, the Amazon ECS API, or tools that use the SDKs and API. These events include Amazon ECS stopping and replaces a task because the containers in the task have stopped running, or have failed too many health checks from Elastic Load Balancing. For more information, see [Service event messages \(p. 770\)](#).

If your task ran on a container instance on Amazon EC2 or external computers, you can also look at the logs of the container runtime and the ECS Agent. These logs are on the host EC2 instance or external computer. For more information, see [Amazon ECS Log File Locations \(p. 781\)](#).

Important

Stopped tasks only appear in the Amazon ECS console, AWS CLI, and AWS SDKs for at least 1 hour after the task stops. After that, the details of the stopped task expire and aren't available in Amazon ECS.

Amazon ECS also sends task state change events to Amazon EventBridge. You can't view events in EventBridge. Instead, you create rules to send the events to other persistent storage such as Amazon CloudWatch Logs. You can use the storage to view your stopped task details after it has expired from view in the Amazon ECS console. For more information, see [Task state change events \(p. 563\)](#).

For a sample EventBridge configuration to archive Amazon ECS events to Amazon CloudWatch Logs, see [ECS Stopped Tasks in CloudWatch Logs](#) on the GitHub website.

Follow these steps to check stopped tasks for errors.

Console

AWS Management Console

The following steps can be used to check stopped tasks for errors using the new AWS Management Console.

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : name** page, choose the **Tasks** tab.

5. Choose the stopped task to inspect.
6. In the **Status** section, inspect the **Stopped reason** field to see the reason that the task was stopped.

AWS CLI

1. List the stopped tasks in a cluster. The output contains the Amazon Resource Name (ARN) of the task, which you need to describe the task.

```
aws ecs list-tasks \
  --cluster cluster_name \
  --desired-status STOPPED \
  --region us-west-2
```

2. Describe the stopped task to retrieve the stoppedReason in the response.

```
aws ecs describe-tasks \
  --cluster cluster_name \
  --tasks arn:aws:ecs:us-west-2:account_id:task/cluster_name/task_ID \
  --region us-west-2
```

Additional resources

The following pages provide additional information about error codes:

- [Why is my Amazon ECS task stopped](#)
- [Stopped tasks error codes](#)

CannotPullContainer task errors

The following errors indicate that, when creating a task, the container image specified can't 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's because a route to the internet doesn't 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:

- For tasks in public subnets, specify **ENABLED** for **Auto-assign public IP** when launching the task. For more information, see [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#).
- For tasks in private subnets, specify **DISABLED** for **Auto-assign public IP** when launching the task, and configure a NAT gateway in your VPC to route requests to the internet. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*.

Context canceled

The common cause for this error is because the VPC your task is using doesn't have a route to pull the container image from Amazon ECR.

Image not found

When you specify an Amazon ECR image in your container definition, you must use the full URI of your ECR repository along with the image name in that repository. If the repository or image can't 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 URI and the image name. Also make sure 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 \(p. 626\)](#).

Amazon ECR endpoint connection issue

If you are trying to pull an Amazon ECR image and you do not have the correct permissions for the Amazon ECR endpoint, you see an error similar to the following:

```
CannotPullContainerError: API error
```

To resolve this issue, Amazon ECS must communicate with the Amazon ECR endpoint. For information about how to resolve this issues, see [How can I resolve the "CannotPullContainerError: API error" in Amazon ECS](#) on the AWS Support website.

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/GetImageBlob11111111: no space left on device
```

To resolve this issue, free up disk space.

If you are using the Amazon ECS-optimized AMI, you can use the following command to retrieve the 20 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, similar to the preceding example, the root volume might 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 writes them in files using the JSON format. You can 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-opt": {  
        "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 \(p. 161\)](#).

Docker Hub rate limiting

If you receive one of the following errors, you're likely hitting the Docker Hub rate limits:

ERROR: toomanyrequests: Too Many Requests.

You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: <https://www.docker.com/increase-rate-limits>.

For more information about the Docker Hub rate limits, see [Understanding Docker Hub rate limiting](#).

If you have increased the Docker Hub rate limit and you need to authenticate your Docker pulls for your container instances, see [Private registry authentication for container instances](#) in the *Amazon Elastic Container Service Developer Guide*.

Fail to copy image

If you receive an error similar to the following when launching a task, it's because there is no access to the image:

CannotPullContainerError: ref pull has been retried 1 time(s): failed to copy:
httpReaderSeeker: failed open: unexpected status code

To resolve this issue, you can:

- For Fargate tasks, see [How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks on Fargate](#).
- For other tasks, see [How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks](#).

Pull access denied

If you receive an error similar to the following when launching a task, it's because there is no access to the image:

CannotPullContainerError: pull access denied

To resolve this issue, you might need to authenticate your Docker client with Amazon ECR. For more information, see [Private registry authentication](#) in the *Amazon ECR User Guide*.

For additional information about STOPPED errors, see [Stopped tasks error codes](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Service event messages

When troubleshooting a problem with a service, the first place you should check for diagnostic information is the service event log. You can view service events using the `DescribeServices` API, the AWS CLI, or by using the AWS Management Console.

When viewing service event messages using the Amazon ECS API, only the events from the service scheduler are returned. These include the most recent task placement and instance health events. However, the Amazon ECS console displays service events from the following sources.

- Task placement and instance health events from the Amazon ECS service scheduler. These events will have a prefix of **service (*service-name*)**. To ensure that this event view is helpful, we only show the 100 most recent events and duplicate event messages are omitted until either the cause is resolved or six hours passes. If the cause is not resolved within six hours, you will receive another service event message for that cause.
- Service Auto Scaling events. These events will have a prefix of **Message**. The 10 most recent scaling events are shown. These events only occur when a service is configured with an Application Auto Scaling scaling policy.

Use the following steps to view your current service event messages.

Console

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. Choose the service to inspect.
5. Choose **Deployments and events**, under **Events**, view the messages.

AWS CLI

Use the [`describe-services`](#) command to view the service event messages for a specified service.

The following AWS CLI example describes the ***service-name*** service in the ***default*** cluster, which will provide the latest service event messages.

```
aws ecs describe-services \
--cluster default \
--services service-name \
--region us-west-2
```

Service event messages

The following are examples of service event messages you may see in the Amazon ECS console.

Service (*service-name*) has reached a steady state.

The service scheduler will send a service (*service-name*) has reached a steady state. service event when the service is healthy and at the desired number of tasks, thus reaching a steady state.

The service scheduler reports the status periodically, so you might receive this message multiple times.

Service (*service-name*) was unable to place a task because no container instance met all of its requirements.

The service scheduler will send this event message when it could not find the available resources to add another task. The possible causes for this are:

No container instances were found in your cluster

If no container instances are registered in the cluster you attempt to run a task in, you will receive this error. You should add container instances to your cluster. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

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.

Too many ports registered

The closest matching container instance for task placement can not exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using dynamic host port mapping may remediate the issue.

Port already in-use

The task definition of this task uses the same port in its port mapping as an task already running on the container instance that was chosen to run on. The service event message would have the chosen container instance ID as part of the message below.

The closest matching container-instance is already using a port required by your task.

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.

Note

If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 304\)](#).

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 (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the awsvpcTrunking account setting, the ENI limit for each container instance depends on the instance type. For more information, see [IP Addresses Per Network Interface Per Instance Type](#) in the *Amazon EC2 User Guide for Linux Instances*.
- If you **have** opted in to the awsvpcTrunking account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each container instance will still be at the default value. For more information, see [IP Addresses Per Network Interface Per Instance Type](#) in the *Amazon EC2 User Guide for Linux Instances*.
- If you **have** opted in to the awsvpcTrunking account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see [Supported Amazon EC2 instance types \(p. 287\)](#).

For more information about opting in to the awsvpcTrunking account setting, see [Elastic network interface trunking \(p. 283\)](#).

You can add container instances to your cluster to provide more available network adapters.

Container instance missing required attribute

Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the ECS_AVAILABLE_LOGGING_DRIVERS agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you do not have any available container instances that can satisfy this requirement, the task cannot be placed.

A common cause of this error is if your service is using tasks that use the awsvpc network mode and the EC2 launch type and the cluster you specified does not have a container instance registered to it in the same subnet that was specified in the awsvpcConfiguration when the service was created.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see [Task definition parameters \(p. 799\)](#) and [Amazon ECS container agent configuration \(p. 370\)](#).

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* has insufficient CPU units available.

The closest matching container instance for task placement does not contain enough CPU units to meet the requirements in the task definition. Review the CPU requirements in both the task size and container definition parameters of the task definition.

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* encountered error "AGENT".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs; for more information, see [Amazon ECS Container Agent Log \(p. 781\)](#). You should also verify that the

agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following command.

- For the Amazon ECS-optimized Amazon Linux 2 AMI

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI

```
sudo stop ecs && sudo start ecs
```

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 \(p. 778\)](#).

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 \(p. 528\)](#).

After the service is updated, for example with an updated task definition, the service scheduler resumes normal behavior.

service (*service-name*) operations are being throttled. Will try again later.

This service is unable to launch more tasks due to API throttling limits. Once the service scheduler is able to launch more tasks, it will resume.

To request an API rate limit quota increase, open the [AWS Support Center](#) page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

service (*service-name*) was unable to stop or start tasks during a deployment because of the service deployment configuration. Update the minimumHealthyPercent or maximumPercent value and try again.

This service is unable to stop or start tasks during a service deployment due to the deployment configuration. The deployment configuration consists of the `minimumHealthyPercent` and `maximumPercent` values which are defined when the service is created, but can also be updated on an existing service.

The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is 50 and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is 200 and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks.

Likewise, if the maximum percent is 125 and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is triggered.

service (*service-name*) was unable to place a task. Reason: You've reached the limit on the number of tasks you can run concurrently

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas" \(p. 536\)](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: Internal error.

The service is unable to start a task due to a subnet being in an unsupported Availability Zone.

For information about the supported Fargate Regions and Availability Zones, see [the section called "AWS Fargate Regions" \(p. 540\)](#).

For information about how to view the subnet Availability Zone, see [View your subnet](#) in the *Amazon VPC User Guide*.

service (*service-name*) was unable to place a task. Reason: The requested CPU configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas" \(p. 536\)](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Service (*service-name*) was unable to place a task. Reason: The requested MEMORY configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas" \(p. 536\)](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: You've reached the limit on the number of vCPUs you can run concurrently

AWS Fargate is transitioning from task count-based quotas to vCPU-based quotas.

You can request a quota increase for the Fargate vCPU-based quota. For more information, see [the section called "Service quotas" \(p. 536\)](#). To request a Fargate quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

service (*service-name*) was unable to reach steady state because task set (*taskSet-ID*) was unable to scale in. Reason: The number of protected tasks are more than the desired count of tasks.

The service has more protected tasks than the desired count of tasks. You can do one the following:

- Wait until the protection on the current tasks expire, enabling them to be terminated.
- Determine which tasks can be stopped. Then use the `UpdateTaskProtection` API with the `protectionEnabled` option set to `false` to unset protection for these tasks.

- Increase the desired task count of the service to more than the number of protected tasks.

service (*service-name*) was unable to reach steady state. Reason: No Container Instances were found in your capacity provider.

The service scheduler will send this event message when it could not find the available resources to add another task. The possible causes for this are:

No container instances were found in your cluster

If no container instances are registered in the cluster you attempt to run a task in, you will receive this error. You should add container instances to your cluster. For more information, see [Launching an Amazon ECS Linux container instance \(p. 272\)](#).

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.

Too many ports registered

The closest matching container instance for task placement can not exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using dynamic host port mapping may remediate the issue.

Port already in-use

The task definition of this task uses the same port in its port mapping as an task already running on the container instance that was chosen to run on. The service event message would have the chosen container instance ID as part of the message below.

The closest matching container-instance is already using a port required by your task.

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.

Note

If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 304\)](#).

Not enough available ENI attachment points

Tasks that use the awsvpc network mode each receive their own elastic network interface (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the awsvpcTrunking account setting, the ENI limit for each container instance depends on the instance type. For more information, see [IP Addresses Per Network Interface Per Instance Type](#) in the *Amazon EC2 User Guide for Linux Instances*.
- If you **have** opted in to the awsvpcTrunking account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each

container instance will still be at the default value. For more information, see [IP Addresses Per Network Interface Per Instance Type](#) in the *Amazon EC2 User Guide for Linux Instances*.

- If you **have** opted in to the awsvpcTrunking account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see [Supported Amazon EC2 instance types \(p. 287\)](#).

For more information about opting in to the awsvpcTrunking account setting, see [Elastic network interface trunking \(p. 283\)](#).

You can add container instances to your cluster to provide more available network adapters.

Container instance missing required attribute

Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the ECS_AVAILABLE_LOGGING_DRIVERS agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you do not have any available container instances that can satisfy this requirement, the task cannot be placed.

A common cause of this error is if your service is using tasks that use the awsvpc network mode and the EC2 launch type and the cluster you specified does not have a container instance registered to it in the same subnet that was specified in the awsvpcConfiguration when the service was created.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see [Task definition parameters \(p. 799\)](#) and [Amazon ECS container agent configuration \(p. 370\)](#).

Invalid CPU or memory value specified

When registering a task definition using the Amazon ECS API or AWS CLI, if you specify an invalid cpu or memory value, the following error is returned.

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.

Note

When using Terraform, the following error may be returned.

Error: ClientException: No Fargate configuration exists for given values.

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. 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.

For task definitions that only specify EC2 for the `requiresCompatibilities` parameter, the supported CPU values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs). The memory value must be an integer and the limit is dependent upon the amount of available memory on the underlying Amazon EC2 instance you use.

For task definitions that specify FARGATE for the `requiresCompatibilities` parameter (even if EC2 is also specified), you must use one of the values in the following table, which determines your range of supported values for the CPU and memory parameter.

For tasks hosted on Fargate, the following table shows the valid CPU and memory combinations.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

For tasks hosted on Amazon EC2, supported task CPU values are between 0.125 vCPUs and 192 vCPUs.

Note

Task-level CPU and memory parameters are ignored for Windows containers.

CannotCreateContainerError: API error (500): devmapper

The following Docker error indicates that the thin pool storage on your container instance is full, and that the Docker daemon cannot create new containers:

```
CannotCreateContainerError: API error (500): devmapper: Thin Pool has 4350 free data blocks which is less than minimum required 4454 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior
```

By default, Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.d and later launch with an 8-GiB volume for the operating system that's attached at /dev/xvda and mounted as the root of the file system. There's an additional 22-GiB volume that's attached at /dev/xvdcz that Docker uses for image and metadata storage. If this storage space is filled up, the Docker daemon cannot create new containers.

The easiest way to add storage to your container instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you can't do this, you can add storage

to the volume group that Docker uses and extend its logical volume by following the procedures in [AMI storage configuration \(p. 262\)](#).

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

- To view the thin poll information, run the following command on your container instance:

```
docker info
```

- (Amazon ECS container agent 1.8.0 and later) Reduce the amount of time that stopped or exited containers remain on your container instances. The ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION agent configuration variable sets the time duration to wait from when a task is stopped until the Docker container is removed (by default, this value is 3 hours). This removes the Docker container data. If this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).
- Remove non-running containers and unused images from your container instances. You can use the following example commands to manually remove stopped containers and unused images. Deleted containers can't be inspected later, and deleted images must be pulled again before starting new containers from them.

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

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

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

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

- Remove unused data blocks within containers. You can use the following command to run **fstrim** on any running container and discard any data blocks that are unused by the container file system.

```
sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -I% fstrim /proc/%/root/"
```

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.

Amazon ECS service-linked role doesn't exist

The Amazon ECS service-linked role allows Amazon ECS services to register container instances with Elastic Load Balancing load balancers. The service-linked role must be created in your account. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

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 where 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 isn't configured to use, the task never passes the health check and it's killed.

Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that don't 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're using with the health check.

Ping Path

This value is often set to `index.html`, but if your service doesn't respond to that request, then the health check fails. If your container doesn't 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, you can use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*. If you update the task definition for the service, the container name and container port that are specified in the load balancer configuration must remain in the task definition.

You've reached the limit on the number of tasks you can run concurrently.

For a new account, your quotas might be lower than the service quotas. The service quota for your account can be viewed in the Service Quotas console. To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Troubleshooting service auto scaling

Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress and they resume once the deployment has completed. However, scale-out processes continue to occur,

unless suspended, during a deployment. For more information, see [Suspending and resuming scaling for Application Auto Scaling](#).

Using Docker debug output

If you are having trouble with Docker containers or images, you can turn on debug mode on your Docker daemon. Enabling debugging provides more verbose output from the daemon and you can use this information to find out more about why your containers or images are having issues.

Enabling Docker debug mode can be especially useful in retrieving error messages that are sent from container registries, such as Amazon ECR, and, in many circumstances, enabling debug mode is the only way to see these error messages.

Important

This procedure is written for the Amazon ECS-optimized Amazon Linux AMI. For other operating systems, see [Enable debugging](#) and [Control and configure Docker with systemd](#) in the Docker documentation.

To enable Docker daemon debug mode on the Amazon ECS-optimized Amazon Linux AMI

1. Connect to your container instance.
2. Open the Docker options file with a text editor, such as `vi`. For the Amazon ECS-optimized Amazon Linux AMI, the Docker options file is at `/etc/sysconfig/docker`.
3. Find the Docker options statement and add the `-D` option to the string, inside the quotes.

Note

If the Docker options statement begins with a `#`, remove that character to uncomment the statement and enable the options.

For the Amazon ECS-optimized Amazon Linux AMI, the Docker options statement is called `OPTIONS`. For example:

```
# Additional startup options for the Docker daemon, for example:  
# OPTIONS="--ip-forward=true --iptables=true"  
# By default we limit the number of open files per container  
OPTIONS="-D --default-ulimit nofile=1024:4096"
```

4. Save the file and exit your text editor.
5. Restart the Docker daemon.

```
sudo service docker restart
```

The output is as follows:

```
Stopping docker: [ OK ]  
Starting docker: . [ OK ]
```

6. Restart the Amazon ECS agent.

```
sudo service ecs restart
```

Your Docker logs should now show more verbose output.

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from  
server: \"{\\"errors\\\": [{\\\"code\\\": \\"DENIED\\\", \\"message\\\": \\"User:
```

```
arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform:  
ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test  
\\\""}]\\" http.Header{\"Connection\":[:]string{\"keep-alive\"}, \"Content-Type\":  
[:]string{\"application/json; charset=utf-8\"}, \"Date\":[:]string{\"Wed, 30 Dec 2015  
21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[:]string{\"registry/2.0\"},  
\"Content-Length\":[:]string{\"235\"}}"
```

Amazon ECS Log File Locations

Amazon ECS stores logs in the `/var/log/ecs` folder of your container instances. There are logs available from the Amazon ECS container agent and from the `ecs-init` service that controls the state of the agent (start/stop) on the container instance. You can view these log files by connecting to a container instance using SSH.

Note

If you are not sure how to collect all of the logs on your container instances, you can use the Amazon ECS logs collector. For more information, see [Amazon ECS logs collector \(p. 784\)](#).

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs on your container instances.

For container agent version 1.36.0 and later, by default the logs are located at `/var/log/ecs/ecs-agent.log` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log` on Windows instances.

For container agent version 1.35.0 and earlier, by default the logs are located at `/var/log/ecs/ecs-agent.log.timestamp` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.timestamp` on Windows instances.

By default, the agent logs are rotated hourly with a maximum of 24 logs being stored.

The following are the container agent configuration variables that can be used to change the default agent logging behavior. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

ECS_LOGFILE

Example values: `/ecs-agent.log`

Default value on Linux: Null

Default value on Windows: Null

The location where agent logs should be written. If you are running the agent via `ecs-init`, which is the default method when using the Amazon ECS-optimized AMI, the in-container path will be `/log` and `ecs-init` mounts that out to `/var/log/ecs/` on the host.

ECS_LOGLEVEL

Example values: `crit, error, warn, info, debug`

Default value on Linux: `info`

Default value on Windows: `info`

The level of detail to log.

ECS_LOGLEVEL_ON_INSTANCE

Example values: none, crit, error, warn, info, debug

Default value on Linux: none, if ECS_LOG_DRIVER is explicitly set to a non-empty value; otherwise the same value as ECS_LOGLEVEL

Default value on Windows: none, if ECS_LOG_DRIVER is explicitly set to a non-empty value; otherwise the same value as ECS_LOGLEVEL

Can be used to override ECS_LOGLEVEL and set a level of detail that should be logged in the on-instance log file, separate from the level that is logged in the logging driver. If a logging driver is explicitly set, on-instance logs are turned off by default, but can be turned back on with this variable.

ECS_LOG_DRIVER

Example values: awslogs, fluentd, gelf, json-file, journald, logentries syslog, splunk

Default value on Linux: json-file

Default value on Windows: Not applicable

Determines the logging driver to be used by the agent container.

ECS_LOG_ROLLOVER_TYPE

Example values: size, hourly

Default value on Linux: hourly

Default value on Windows: hourly

Determines whether the container agent log file will be rotated hourly or based on size. By default, the agent log file is rotated each hour.

ECS_LOG_OUTPUT_FORMAT

Example values: logfmt, json

Default value on Linux: logfmt

Default value on Windows: logfmt

Determines the log output format. When the json format is used, each line in the log will be a structured JSON map.

ECS_LOG_MAX_FILE_SIZE_MB

Example values: 10

Default value on Linux: 10

Default value on Windows: 10

When the ECS_LOG_ROLLOVER_TYPE variable is set to size, this variable determines the maximum size (in MB) of the log file before it is rotated. If the rollover type is set to hourly, then this variable is ignored.

ECS_LOG_MAX_ROLL_COUNT

Example values: 24

Default value on Linux: 24

Default value on Windows: 24

Determines the number of rotated log files to keep. Older log files are deleted after this limit is reached.

For container agent version 1.36.0 and later, the following is an example log file when the logfmt format is used.

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit: ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs" module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs" module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start listening..." module=eventstream.go
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

The following is an example log file when the JSON format is used.

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic Container Service Agent", "module": "engine.go"}
```

For container agent versions 1.35.0 and earlier, the following is the format of the log file.

```
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0 (895f3c1)
2016-08-15T15:54:41Z [INFO] Loading configuration
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be overridden to 3h0m0s, parsed value 0, minimum threshold 1m0s
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load state
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20 1.21 1.22]
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

Amazon ECS ecs-init Log

The `ecs-init` process stores logs at `/var/log/ecs/ecs-init.log`.

You can use the following command to view the log files.

```
cat /var/log/ecs/ecs-init.log
```

Output:

```
2018-02-16T18:13:54Z [INFO] pre-start
```

```
2018-02-16T18:13:56Z [INFO] start
2018-02-16T18:13:56Z [INFO] No existing agent container to remove.
2018-02-16T18:13:56Z [INFO] Starting Amazon Elastic Container Service Agent
```

IAM Roles for Tasks Credential Audit Log

When the credential provider for the IAM role is used to provide credentials to tasks, these requests are saved in an audit log. The audit log inherits the same log rotation settings as the container agent log. The ECS_LOG_ROLLOVER_TYPE, ECS_LOG_MAX_FILE_SIZE_MB, and ECS_LOG_MAX_ROLL_COUNT container agent configuration variables can be set to affect the behavior of the audit log. For more information, see [Amazon ECS Container Agent Log \(p. 781\)](#).

For container agent version 1.36.0 and later, the audit log is located at /var/log/ecs/audit.log. When the log is rotated, a timestamp in `YYYY-MM-DD-HH` format is added to the end of the log file name.

For container agent version 1.35.0 and earlier, the audit log is located at /var/log/ecs/audit.log.`YYYY-MM-DD-HH`.

The log entry format is as follows:

- Timestamp
- HTTP response code
- IP address and port number of request origin
- Relative URI of the credential provider
- The user agent that made the request
- The ARN of the task to which the requesting container belongs
- The GetCredentials API name and version number
- The name of the Amazon ECS cluster to which the container instance is registered
- The container instance ARN

An example log entry is shown below.

You can use the following command to view the log files.

```
cat /var/log/ecs/audit.log.2016-07-13-16
```

Output:

```
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-requests/2.7.0
CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN GetCredentials
1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Amazon ECS logs collector

If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector. It is available on GitHub for both [Linux](#) and [Windows](#). The script collects general operating system logs as well as Docker and Amazon ECS container agent logs, which can be helpful for troubleshooting AWS Support cases. It then compresses and archives the collected

information into a single file that can easily be shared for diagnostic purposes. It also supports enabling debug mode for the Docker daemon and the Amazon ECS container agent on Amazon Linux variants, such as the Amazon ECS-optimized AMI. Currently, the Amazon ECS logs collector supports the following operating systems:

- Amazon Linux
- Red Hat Enterprise Linux 7
- Debian 8
- Ubuntu 14.04
- Ubuntu 16.04
- Ubuntu 18.04
- Windows Server 2016

Note

The source code for the Amazon ECS logs collector is available on GitHub for both [Linux](#) and [Windows](#). We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of this software.

To download and run the Amazon ECS logs collector for Linux

1. Connect to your container instance.
2. Download the Amazon ECS logs collector script.

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. Run the script to collect the logs and create the archive.

Note

To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `--mode=enable-debug` option to the command below. This may restart the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see [Container instance draining \(p. 354\)](#).

```
[ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
```

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

To download and run the Amazon ECS logs collector for Windows

1. Connect to your container instance. For more information, see [Connecting to Your Windows Instance](#) in the [Amazon EC2 User Guide for Windows Instances](#).
2. Download the Amazon ECS logs collector script using PowerShell.

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-logs-collector.ps1
```

3. Run the script to collect the logs and create the archive.

Note

To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `-RunMode debug` option to the command below. This restarts the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see [Container instance draining \(p. 354\)](#).

```
.\ecs-logs-collector.ps1
```

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

Agent introspection diagnostics

The Amazon ECS agent introspection API can provide helpful diagnostic information. For example, you can use the agent introspection API to get the Docker ID for a container in your task. You can use the agent introspection API by connecting to a container instance using SSH.

Important

Your container instance must have an IAM role that allows access to Amazon ECS in order to reach the introspection API. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

The below example shows two tasks, one that is currently running and one that was stopped.

Note

The command below is piped through the `python -mjson.tool` for greater readability.

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

Output:

```
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
          Dload  Upload Total   Spent    Left Speed
100  1095  100  1095    0      0  117k      0  --::--  --::--  --::--  133k
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-a8d14064fd24",
      "Containers": [
        {
          "DockerId": "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
          "DockerName": "ecs-console-sample-app-static-6-simple-app-86caf9bcabe3e9c61600",
          "Name": "simple-app"
        },
        {
          "DockerId": "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
          "DockerName": "ecs-console-sample-app-static-6-busybox-ce83ce978a87a890ab01",
          "Name": "busybox"
        }
      ],
      "Status": "STOPPED"
    }
  ]
}
```

```

        "Family": "console-sample-app-static",
        "KnownStatus": "STOPPED",
        "Version": "6"
    },
    {
        "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-
a638-097bea534740",
        "Containers": [
            {
                "DockerId":
                "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                "DockerName": "ecs-console-sample-app-static-6-simple-app-
f0e5859699a7aecfb101",
                "Name": "simple-app"
            },
            {
                "DockerId":
                "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                "Name": "busybox"
            }
        ],
        "DesiredStatus": "RUNNING",
        "Family": "console-sample-app-static",
        "KnownStatus": "RUNNING",
        "Version": "6"
    }
]
}

```

In the above example, the stopped task (**090eff9b-1ce3-4db6-848a-a8d14064fd24**) has two containers. You can use **docker inspect container-ID** to view detailed information on each container. For more information, see [Amazon ECS container agent introspection \(p. 410\)](#).

Docker diagnostics

Docker provides several diagnostic tools that help you troubleshoot problems with your containers and tasks. For more information about all of the available Docker command line utilities, see the [Docker Command Line](#) topic in the Docker documentation. You can access the Docker command line utilities by connecting to a container instance using SSH.

The exit codes that Docker containers report can also provide some diagnostic information (for example, exit code 137 means that the container received a SIGKILL signal). For more information, see [Exit Status](#) in the Docker documentation.

List Docker containers

You can use the **docker ps** command on your container instance to list the running containers. In the below example, only the Amazon ECS container agent is running. For more information, see [docker ps](#) in the Docker documentation.

```
docker ps
```

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

cee0d6986de0 Up 22 hours	amazon/amazon-ecs-agent:latest "/agent" 127.0.0.1:51678->51678/tcp ecs-agent	22 hours ago
-----------------------------	---	--------------

You can use the **docker ps -a** command to see all containers (even stopped or killed containers). This is helpful for listing containers that are unexpectedly stopping. In the following example, container f7f1f8a7a245 exited 9 seconds ago, so it doesn't show up in a **docker ps** output without the **-a** flag.

```
docker ps -a
```

Output:

CONTAINER ID	IMAGE	COMMAND	NAMES
CREATED	STATUS	PORTS	
db4d48e411b1 seconds ago	amazon/ecs-emptyvolume-base:autogenerated	"not-applicable"	19 ecs-console-
f7f1f8a7a245 hours ago	busybox:buildroot-2014.02	"\sh -c '/bin/sh -c	22 ecs-console-
	Exited (137) 9 seconds ago		
sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700 189a8ff4b5f0 hours ago	sample-app-static-6-busybox-ce83ce978a87a890ab01	"httpd-foreground"	22 ecs-console-
	httpd:2		
	Exited (137) 40 seconds ago		
sample-app-static-6-simple-app-86caf9bcabe3e9c61600 0c7dca9321e3 hours ago	amazon/ecs-emptyvolume-base:autogenerated	"not-applicable"	22 ecs-console-
sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700 cee0d6986de0 hours ago	amazon/amazon-ecs-agent:latest	"httpd"	22 ecs-agent
	Up 22 hours	127.0.0.1:51678->51678/tcp	

View Docker Logs

You can view the STDOUT and STDERR streams for a container with the **docker logs** command. In this example, the logs are displayed for the **dc7240fe892a** container and piped through the **head** command for brevity. For more information, go to [docker logs](#) in the Docker documentation.

Note

Docker logs are only available on the container instance if you are using the default json log driver. If you have configured your tasks to use the awslogs log driver, then your container logs are available in CloudWatch Logs. For more information, see [Using the awslogs log driver \(p. 161\)](#).

```
docker logs dc7240fe892a | head
```

Output:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472] AH00489:
Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
```

```
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Inspect Docker Containers

If you have the Docker ID of a container, you can inspect it with the **docker inspect** command. Inspecting containers provides the most detailed view of the environment in which a container was launched. For more information, see [docker inspect](#) in the Docker documentation.

```
docker inspect dc7240fe892a
```

Output:

```
[{
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
        },
        "Hostname": "dc7240fe892a",
        ...
    }
}
```

AWS Fargate throttling quotas

AWS Fargate limits Amazon ECS tasks and Amazon EKS pods launch rates to quotas (formerly referred to as limits) using a [token bucket algorithm](#) for each AWS account on a per-Region basis. With this algorithm, your account has a bucket that holds a specific number of tokens. The number of tokens in the bucket represents your rate quota at any given second. Each customer account has a tasks and pods token bucket that depletes based on the number of tasks and pods launched by the customer account. This token bucket has a bucket maximum that allows you to periodically make a higher number of requests, and a refill rate that allows you to sustain a steady rate of requests for as long as needed.

For example, the tasks and pods token bucket size for a Fargate customer account is 100 tokens, and the refill rate is 20 tokens per second. Therefore, you can immediately launch up to 100 Amazon ECS tasks and Amazon EKS pods per customer account, with a sustained launch rate of 20 Amazon ECS tasks and Amazon EKS pods per second.

Actions	Bucket maximum capacity (or Burst rate)	Bucket refill rate (or Sustained rate)
Fargate Resource rate quota for On Demand Amazon ECS tasks and Amazon EKS pods ¹ (p. 790)	100	20
Fargate Resource rate quota for Spot Amazon ECS tasks	100	20

¹ Accounts launching only Amazon EKS pods have a burst rate of 20 with a sustained pod launch rate of 20 pod launches per second when using the platform versions called out in the [Amazon EKS platform versions](#).

Throttling the RunTask API

In addition, Fargate limits the request rate when launching tasks using the Amazon ECS RunTask API using a separate quota. Fargate limits Amazon ECS RunTask API requests for each AWS account on a per-Region basis. Each request that you make removes one token from the bucket. We do this to help the performance of the service, and to ensure fair usage for all Fargate customers. API calls are subject to the request quotas whether they originate from the Amazon Elastic Container Service console, a command line tool, or a third-party application. The rate quota for calls to the Amazon ECS RunTask API is 20 calls per second (burst and sustained). Each call to this API can, however, launch up to 10 tasks. This means you can launch 100 tasks in one second by making 10 calls to this API, requesting 10 tasks to be launched in each call. Similarly, you could also make 20 calls to this API, requesting 5 tasks to be launched in each call. For more information on API throttling for Amazon ECS RunTask API , see [API request throttling](#) in the Amazon ECS API Reference.

In practice, task and pod launch rates are also dependent on other considerations such as container images to be downloaded and unpacked, health checks and other integrations enabled, such as registering tasks or pods into a load balancer. Customers will see variations in task and pod launch rates compared with the quotas represented above based on the features that customers enable.

Adjusting rate quotas

You can request an increase for Fargate rate throttling quotas for your AWS account. To request a quota adjustment, contact the [AWS Support Center](#).

API failure reasons

When an API action that you have triggered through the Amazon ECS API, console, or the AWS CLI exits with a `failures` error message, the following may assist in troubleshooting the cause. The failure will return a reason and the Amazon Resource Name (ARN) of the resource associated with the failure.

Many resources are Region-specific, so when using the console ensure that you set the correct Region for your resources. When using the AWS CLI, make sure that your AWS CLI commands are being sent to the correct Region with the `--region region` parameter.

For more information about the structure of the `Failure` data type, see [Failure](#) in the *Amazon Elastic Container Service API Reference*.

The following are examples of failure messages you might receive when running API commands.

API action	Failure reason or Stopped reason	Cause
DescribeClusters	MISSING	The specified cluster wasn't found. Verify the spelling of the cluster name.
DescribeInstances	MISSING	The specified container instance wasn't found. Verify that you specified the cluster the container instance is registered to and that both the container instance ARN or ID is correct.
DescribeServices	MISSING	The specified service wasn't found. Verify that the correct cluster or Region is specified and that the service ARN or name is valid.
DescribeTasks	MISSING	The specified task wasn't found. Verify the correct cluster or Region is specified and that both the task ARN or ID is valid.
DescribeTasks	TaskFailedToStart: RESOURCE:*	<p>For RESOURCE:CPU errors, the number of CPUs requested by the task are unavailable on your container instances. This generally happens when the CPU unit requirement in your task definition is larger than the CPU size of the EC2 instances defined in the Auto Scaling group mapped to the capacity provider. You need to check your capacity provider configuration. For information about how to add, view, and modify your capacity providers, see the section called "Capacity providers" (p. 220).</p> <p>For RESOURCE:MEMORY errors, the amount of memory requested by the task are unavailable on your container instances. This generally happens when the memory amount requirement in your task definition is larger than the supported memory on the EC2 instances defined in the Auto Scaling group mapped to the capacity provider. You need to check your capacity provider configuration. For information</p>

API action	Failure reason or Stopped reason	Cause
		about how to add, view, and modify your capacity providers, see the section called "Capacity providers" (p. 220) .
	TaskFailedToStart: AGENT	<p>The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected.</p> <p>For information about how to troubleshoot an agent that's disconnected, see How do I troubleshoot a disconnected Amazon ECS agent.</p>
	TaskFailedToStart: MemberOf placement constraint unsatisfied	There is no container instance that meets the placement constraints defined in your task definition.
	TaskFailedToStart: ATTRIBUTE	Your task definition contains a parameter that requires a specific container instance attribute that isn't available on your container instances. For example, if your task uses the <code>awsvpc</code> network mode, but there are no instances in your specified subnets with the <code>ecs.capability.task-eni</code> attribute. For more information about which attributes are required for specific task definition parameters and agent configuration variables, see Task definition parameters (p. 799) and Amazon ECS container agent configuration (p. 370) .
	TaskFailedToStart: NO ACTIVE INSTANCES	<p>There are no active instances in your capacity provider. For information about how to add, view, and modify your capacity providers, see the section called "Capacity providers" (p. 220).</p> <p>For information about how to manage your Auto Scaling groups, see Auto Scaling groups in the <i>Amazon EC2 Auto Scaling User Guide</i>.</p>

API action	Failure reason or Stopped reason	Cause
	TaskFailedToStart: EMPTY CAPACITY PROVIDER	<p>There are no instances in your cluster. This is most likely because of an empty capacity provider, or because the instances in the capacity provider are not registered to the cluster. For information about how to manage your capacity providers, see Amazon ECS capacity providers (p. 220). For information about how to manage your Auto Scaling groups, see Auto Scaling groups in the Amazon EC2 Auto Scaling User Guide.</p>
GetTaskProtection	MISSING	<p>The specified task wasn't found. Verify that the cluster name or ARN and the task ARN or ID are valid.</p>
	TASK_NOT_VALID	<p>The specified task is not part of an Amazon ECS service. Only Amazon ECS service-managed tasks can be protected. Verify the task ARN or ID and try again.</p>

API action	Failure reason or Stopped reason	Cause
RunTask or StartTask	RESOURCE : * For RESOURCE : ENI errors, your cluster doesn't have any available elastic network interface attachment points, which are required for tasks that use the awsvpc network mode. 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 more information about how many network interfaces are supported for each instance type, see IP Addresses Per Network Interface Per Instance Type in the <i>Amazon EC2 User Guide for Linux Instances</i> . For RESOURCE : GPU errors, the number of GPUs requested by the task are unavailable and you might need to add GPU-enabled container instances to your cluster. For more information, see Working with GPUs on Amazon ECS (p. 142) .	The resource or resources that are requested by the task are unavailable on the container instances in the cluster. If the resource is CPU, memory, ports, or elastic network interfaces, you might need to add additional container instances to your cluster.
AGENT		The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected. For information about how to troubleshoot an agent that's disconnected, see How do I troubleshoot a disconnected Amazon ECS agent .

API action	Failure reason or Stopped reason	Cause
	LOCATION	The container instance that you attempted to launch a task onto is in a different Availability Zone than the subnets that you specified in your awsVpcConfiguration.
	ATTRIBUTE	Your task definition contains a parameter that requires a specific container instance attribute that isn't available on your container instances. For example, if your task uses the awsvpc network mode, but there are no instances in your specified subnets with the ecs.capability.task-eni attribute. For more information about which attributes are required for specific task definition parameters and agent configuration variables, see Task definition parameters (p. 799) and Amazon ECS container agent configuration (p. 370) .
StartTask	MISSING	The container instance you attempted to launch the task onto can't be found. Perhaps the wrong cluster or Region is specified, or the container instance ARN or ID is misspelled.
	INACTIVE	The container instance that you attempted to launch a task onto was previously deregistered with Amazon ECS and can't be used.
UpdateTaskProtection	DEPLOYMENT_BLOCKED	Cannot set task protection as one or more protected tasks are preventing the service deployment from reaching a steady state. Unset task protection on existing tasks or wait until task protection expires.
	MISSING	The specified task wasn't found. Verify that the cluster name or ARN and the task ARN or ID are valid.

API action	Failure reason or Stopped reason	Cause
	TASK_NOT_VALID	The specified task is not part of an Amazon ECS service. Only Amazon ECS service-managed tasks can be protected. Verify the task ARN or ID and try again.

Note

Besides the failure scenarios described here, APIs can also fail due to exceptions, resulting in error responses. For a list of such exceptions, see [Common Errors](#).

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 **AmazonECS_FullAccess**, **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 console at <https://console.aws.amazon.com/ecs/v2>.
 - b. In the navigation pane, choose **Task definitions**.
 - c. Choose **Create new task definition, Create new task definition with JSON**.
 - d. In the JSON editor box, edit your JSON file,

Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

Note

Replace the `cluster-name` value with the name of your cluster. Replace the `region` value with the Region where you created your cluster.

Replace the `awslogs-region` value with the Region where 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 --cluster cluster-name
--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
}
```

- e. 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 **Deploy, Run task**.
 - b. On the **Run Task** page, choose the cluster, and then choose **Deploy** 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"
  ]
}
```

If you receive an "Unable to locate credentials" error, then the following are possible causes.

- The IAM roles for tasks feature is not enabled on your container instances. For more information, see [Using task IAM roles on your Amazon EC2 or external instances \(p. 633\)](#).
- The credential URL is being throttled. You can use the ECS_TASK_METADATA_RPS_LIMIT container agent parameter to configure the throttle limits. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

Parameter references and resource templates

The following sections describe the task definition and service definition parameters.

Topics

- [Task definition parameters \(p. 799\)](#)
- [Task definition template \(p. 843\)](#)
- [Service definition parameters \(p. 847\)](#)

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

You can use these parameters in a JSON file to configure your task definition. For more information, see [the section called "Example logging option task definitions" \(p. 191\)](#).

The following are more detailed descriptions for each task definition parameter.

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's registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

Launch types

When you register a task definition, you can specify a launch type that Amazon ECS should validate the task definition against. If the task definition doesn't validate against the compatibilities specified, a client exception is returned. For more information, see [Amazon ECS launch types \(p. 85\)](#).

The following parameter is allowed in a task definition.

`requiresCompatibilities`

Type: String array

Required: No

Valid Values: EC2 | FARGATE | EXTERNAL

The launch type to validate the task definition against. This initiates a check to ensure that all of the parameters that are used in the task definition meet the requirements of the launch type.

Task role

`taskRoleArn`

Type: String

Required: No

When you register a task definition, you can provide a task role for an IAM role that allows the containers in the task permission to call the AWS APIs that are specified in its associated policies on your behalf. For more information, see [Task IAM role \(p. 631\)](#).

When you launch the Amazon ECS-optimized Windows Server AMI, IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set. Your containers must also run some configuration code to use the feature. For more information, see [Additional configuration for Windows IAM roles for tasks \(p. 637\)](#).

Task execution role

`executionRoleArn`

Type: String

Required: Conditional

The Amazon Resource Name (ARN) of the task execution role that grants the Amazon ECS container agent permission to make AWS API calls on your behalf.

Note

The task execution IAM role is required depending on the requirements of your task. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

Network mode

`networkMode`

Type: String

Required: No

The Docker networking mode to use for the containers in the task. For Amazon ECS tasks that are hosted on Amazon EC2 Linux instances, the valid values are `none`, `bridge`, `awsvpc`, and `host`. If no network mode is specified, the default network mode is `bridge`. For Amazon ECS tasks hosted on Amazon EC2 Windows instances, the valid values are `default`, and `awsvpc`. If no network mode is specified, the default network mode is used.

If the network mode is set to `none`, the task's containers don't have external connectivity and port mappings can't be specified in the container definition.

If the network mode is `bridge`, the task uses Docker's built-in virtual network on Linux, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Linux uses the `bridge` Docker network driver..

If the network mode is `host`, the task uses the host's network which bypasses Docker's built-in virtual network by mapping container ports directly to the ENI of the Amazon EC2 instance that hosts the task. Dynamic port mappings can't be used in this network mode. A container in a task definition that uses this mode must specify a specific `hostPort` number. A port number on a host can't be used by multiple tasks. As a result, you can't run multiple tasks of the same task definition on a single Amazon EC2 instance.

Important

When running tasks that use the `host` network mode, do not run containers using the root user (UID 0) for better security. As a security best practice, always use a non-root user.

If 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 for tasks that are hosted on Amazon EC2 instances \(p. 90\)](#). Currently, only the Amazon ECS-optimized AMI, other Amazon Linux variants with the `ecs-init` package, or AWS Fargate infrastructure support the `awsvpc` network mode.

If the network mode is `default`, the task uses Docker's built-in virtual network on Windows, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Windows uses the `nat` Docker network driver..

The `host` and `awsvpc` network modes offer the highest networking performance for containers because they use the Amazon EC2 network stack. With the `host` and `awsvpc` network modes, exposed container ports are mapped directly to the corresponding host port (for the `host` network mode) or the attached elastic network interface port (for the `awsvpc` network mode). Because of this, you can't use dynamic host port mappings.

If using the Fargate launch type, the `awsvpc` network mode is required. If using the EC2 launch type, the allowable network mode depends on the underlying EC2 instance's operating system. If Linux, any network mode can be used. If Windows, the `default`, and `awsvpc` modes can be used.

Runtime platform

`operatingSystemFamily`

Type: String

Required: Conditional

Default: LINUX

This parameter is required for Amazon ECS tasks that are hosted on Fargate.

When you register a task definition, you specify the operating system family.

The valid values for Amazon ECS tasks that are hosted on Fargate are `LINUX`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2022_FULL`, and `WINDOWS_SERVER_2022_CORE`.

The valid values for Amazon ECS tasks hosted on EC2 are `LINUX`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2016_FULL`, `WINDOWS_SERVER_2004_CORE`, and `WINDOWS_SERVER_20H2_CORE`.

All task definitions that are used in a service must have the same value for this parameter.

When a task definition is part of a service, this value must match the service `platformFamily` value.

cpuArchitecture

Type: String

Required: Conditional

Default: X86_64

This parameter is required for Amazon ECS tasks hosted on Fargate.

When you register a task definition, you specify the CPU architecture. The valid values are X86_64 and ARM64.

All task definitions that are used in a service must have the same value for this parameter.

When you have Linux tasks for either the Fargate launch type, or the EC2 launch type, you can set the value to ARM64. For more information, see [the section called “Working with 64-bit ARM workloads on Amazon ECS” \(p. 159\)](#).

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. For tasks that are hosted on Amazon EC2 instances, these fields are optional. For tasks that are hosted on Fargate (both Linux and Windows), 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: Conditional

Note

This parameter is not supported for Windows containers.

The hard limit of CPU units to present for 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.

For tasks that run on EC2 or external instances, this field is optional. If your cluster doesn't have any registered container instances with the requested CPU units available, the task fails. Supported values for tasks that run on EC2 or external instances are between 0.125 vCPUs and 10 vCPUs.

For tasks that run on Fargate (both Linux and Windows containers), 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	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

memory

Type: String

Required: Conditional

Note

This parameter is not supported for Windows containers.

The hard limit of memory (in MiB) to present to 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.

For tasks that are hosted on Amazon EC2 instances, this field is optional and any value can be used. If a task-level memory value is specified, then the container-level memory value is optional. If your cluster doesn't have any registered container instances with the requested memory available, the task fails. You can maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type. For more information, see [Container Instance Memory Management \(p. 304\)](#).

For tasks hosted on Fargate (both Linux and Windows containers), 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	Operating systems supported for Fargate
512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)	256 (.25 vCPU)	Linux

Memory value (MiB)	CPU value	Operating systems supported for Fargate
1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)	512 (.5 vCPU)	Linux
2048 (2 GB), 3072 (3 GB), 4096 (4GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)	1024 (1 vCPU)	Linux, Windows
Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)	2048 (2 vCPU)	Linux, Windows
Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)	4096 (4 vCPU)	Linux, Windows
Between 16 GB and 60 GB in 4 GB increments Note This option requires Linux platform 1.4.0 or later.	8192 (8 vCPU)	Linux
Between 32 GB and 120 GB in 8 GB increments Note This option requires Linux platform 1.4.0 or later.	16384 (16vCPU)	Linux

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 \(p. 804\)](#)
- [Advanced container definition parameters \(p. 811\)](#)
- [Other container definition parameters \(p. 827\)](#)

Standard container definition parameters

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

Topics

- [Name \(p. 805\)](#)
- [Image \(p. 805\)](#)
- [Memory \(p. 805\)](#)
- [Port mappings \(p. 807\)](#)
- [Private Repository Credentials \(p. 810\)](#)

Name

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're linking multiple containers in a task definition, the name of one container can be entered in the links of another container. This is to connect the containers.

Image

image

Type: String

Required: Yes

The image used to start a container. This string is passed directly to the Docker daemon. By default, images in the Docker Hub registry are available. 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 aren't propagated to already running tasks.
- Images in private registries are supported. For more information, see [Private registry authentication for tasks \(p. 195\)](#).
- 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@sha256:94af1f2e64d908bc90dbca0035a5b567EXAMPLE`).
- 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

memory

Type: Integer

Required: Conditional

The amount (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. The total amount of memory reserved for all containers within a task must be lower than the task `memory` value, if one is specified. This parameter maps to Memory in the [Create a container](#) section of the [Docker Remote API](#) and the `--memory` option to [docker run](#).

If you're using the Fargate launch type, this parameter is optional.

If you're using the EC2 launch type, you must specify either a task-level memory value or a container-level memory value. If you specify both a container-level memory and `memoryReservation` value, the memory value must be greater than the `memoryReservation` value. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance that the container is placed on. Otherwise, the value of `memory` is used.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

Note

If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 304\)](#).

`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 use more memory when needed. The container can use up to the hard limit that's 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](#).

If a task-level memory value isn't specified, you must specify a non-zero integer for one or both of `memory` or `memoryReservation` in a container definition. 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 that the container is placed on. Otherwise, the value of `memory` is used.

For example, suppose that 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 allows the container to only reserve 128 MiB of memory from the remaining resources on the container instance. At the same time, this configuration also allows the container to use more memory resources when needed.

Note

This parameter isn't supported for Windows containers.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

Note

If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 304\)](#).

Port mappings

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 `aws vpc` network mode, only specify the `containerPort`. The `hostPort` can be kept blank or the same value as the `containerPort`.

Port mappings on Windows use the NetNAT gateway address rather than `localhost`. There's no loopback for port mappings on Windows, so you can't access a container's mapped port from the host itself.

Most fields of this parameter (including `containerPort`, `hostPort`, `protocol`) map 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, host ports must either be undefined or 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.
- Metadata: The task metadata endpoint.

appProtocol

Type: String

Required: No

The application protocol that's used for the port mapping. This parameter only applies to Service Connect. We recommend that you set this parameter to be consistent with the protocol that your application uses. If you set this parameter, Amazon ECS adds protocol-specific connection handling to the service connect proxy. If you set this parameter, Amazon ECS adds protocol-specific telemetry in the Amazon ECS console and CloudWatch.

If you don't set a value for this parameter, then TCP is used. However, Amazon ECS doesn't add protocol-specific telemetry for TCP.

For more information, see [the section called "Service Connect" \(p. 503\)](#).

Valid protocol values: "HTTP" | "HTTP2" | "GRPC"

containerPort

Type: Integer

Required: Yes, when `portMappings` are used

The port number on the container that's bound to the user-specified or automatically assigned host port.

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

For Windows containers on Fargate, you can't use port 3150 for the containerPort. This is because it's reserved.

Suppose that you're using containers in a task with the EC2 launch type and you specify a container port and not a host port. Then, 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 don't count toward the 100 reserved ports quota of a container instance.

`containerPortRange`

Type: String

Required: No

The port number range on the container that's bound to the dynamically mapped host port range.

You can only set this parameter by using the `register-task-definition` API. The option is available in the `portMappings` parameter. For more information, see [register-task-definition](#) in the *AWS Command Line Interface Reference*.

The following rules apply when you specify a `containerPortRange`:

- You must use either the bridge network mode or the awsvpc network mode.
- This parameter is available for both the EC2 and AWS Fargate launch types.
- This parameter is available for both the Linux and Windows operating systems.
- The container instance must have at least version 1.67.0 of the container agent and at least version 1.67.0-1 of the `ecs-init` package.
- You can specify a maximum of 100 port ranges for each container.
- You don't specify a `hostPortRange`. The value of the `hostPortRange` is set as follows:
 - For containers in a task with the awsvpc network mode, the `hostPort` is set to the same value as the `containerPort`. This is a static mapping strategy.
 - For containers in a task with the bridge network mode, the Amazon ECS agent finds open host ports from the default ephemeral range and passes it to docker to bind them to the container ports.
- The `containerPortRange` valid values are between 1 and 65535.
- A port can only be included in one port mapping for each container.
- You can't specify overlapping port ranges.
- The first port in the range must be less than last port in the range.
- Docker recommends that you turn off the `docker-proxy` in the Docker daemon config file when you have a large number of ports.

For more information, see [Issue #11185](#) on Github.

For information about how to turn off the `docker-proxy` in the Docker daemon config file, see [Docker daemon](#) in the *Amazon ECS Developer Guide*.

You can call [DescribeTasks](#) to view the `hostPortRange`, which are the host ports that are bound to the container ports.

The port ranges aren't included in the Amazon ECS task events, which are sent to EventBridge. For more information, see [the section called "Events and EventBridge" \(p. 558\)](#).

`hostPortRange`

Type: String

Required: No

The port number range on the host that's used with the network binding. This is assigned by Docker and delivered by the Amazon ECS agent.

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 kept blank or be the same value as `containerPort`.

Suppose that you're 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`. Your container automatically receives a port in the ephemeral port range for your container instance operating system and Docker version. This is referred to as *dynamic* host port mapping.

The default ephemeral port range Docker version 1.6.0 and later is listed on the instance under `/proc/sys/net/ipv4/ip_local_port_range`. If this kernel parameter is unavailable, the default ephemeral port range from 49153–65535 is used. Don't attempt to specify a host port in the ephemeral port range. This is because these are reserved for automatic assignment. In general, ports under 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 ports 51678–51680. 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. A container instance might have up to 100 reserved ports at a time, including the default reserved ports. Automatically assigned ports don't count toward the 100 reserved ports quota.

name

Type: String

Required: No, required for Service Connect to be configured in a service

The name that's used for the port mapping. This parameter only applies to Service Connect. This parameter is the name that you use in the Service Connect configuration of a service.

For more information, see [Service Connect \(p. 503\)](#).

In the following example, both of the required fields for Service Connect are used.

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

protocol

Type: String

Required: No

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

Important

Only `tcp` is supported for Service Connect. Remember that `tcp` is implied if this field isn't set.

Important

UDP support is only available on container instances that were launched with version 1.2.0 of the Amazon ECS container agent (such as the `amzn-ami-2015.03.c-amazon-ecs-optimized` AMI) or later, or with container agents that have been updated to version 1.3.0 or later. To update your container agent to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#).

If you're 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  
    }  
    ...  
]
```

Private Repository Credentials

`repositoryCredentials`

Type: [RepositoryCredentials object](#)

Required: No

The repository credentials for private registry authentication.

For more information, see [Private registry authentication for tasks \(p. 195\)](#).

`credentialsParameter`

Type: String

Required: Yes, when `repositoryCredentials` are used

The Amazon Resource Name (ARN) of the secret containing the private repository credentials.

For more information, see [Private registry authentication for tasks \(p. 195\)](#).

Note

When you use the Amazon ECS API, AWS CLI, or AWS SDKs, if the secret exists in the same Region as the task that you're launching then you can use either the full ARN or the name of the secret. When you use the AWS Management Console, you must specify the full ARN of the secret.

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

```
"containerDefinitions": [  
    {  
        "image": "private-repo/private-image",  
        "repositoryCredentials": {  
            "credentialsParameter":  
                "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"  
        }  
    }  
]
```

Advanced container definition parameters

The following advanced container definition parameters provide extended capabilities to the [docker run](#) command that's used to launch containers on your Amazon ECS container instances.

Topics

- [Health check \(p. 811\)](#)
- [Environment \(p. 813\)](#)
- [Network settings \(p. 817\)](#)
- [Storage and logging \(p. 819\)](#)
- [Security \(p. 823\)](#)
- [Resource limits \(p. 825\)](#)
- [Docker labels \(p. 826\)](#)

Health check

healthCheck

The container health check command and the 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 that are specified in the task definition. Amazon ECS doesn't monitor Docker health checks that are embedded in a container image but aren't 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.

You can view the health status of both individual containers and a task by either one of two methods. You can call the [DescribeTasks](#) API operation or view the task details in the console.

The health check is designed to make sure that your containers survive agent restarts, upgrades, or temporary unavailability.

The following describes the possible healthStatus values for a container:

- **HEALTHY**—The container health check passed.
- **UNHEALTHY**—The container health check failed.
- **UNKNOWN**—The container health check is being evaluated or there's no container health check defined.

The following describes the possible `healthStatus` values for a task. The container health check status of non-essential containers don't affect the health status of a task.

- **HEALTHY**—All essential containers within the task passed their health checks.
- **UNHEALTHY**—One or more essential containers failed their health check.
- **UNKNOWN**—The essential containers within the task are still having their health checks evaluated. There are only nonessential containers with health checks defined, or there are no container health checks defined.

If a task is run manually and not as part of a service, it continues its lifecycle regardless of its health status. For tasks that are part of a service, if the task reports as unhealthy, then the task is stopped and the service scheduler replaces it.

The following are notes about container health check support:

- When the Amazon ECS agent cannot connect to the Amazon ECS service, the service reports the container as UNHEALTHY.
- The health check statuses are the "last heard from" response from the Amazon ECS agent. There are no assumptions made about the status of the container health checks.
- Container health checks require version 1.17.0 or later of the Amazon ECS container agent. For more information, see [Updating the Amazon ECS container agent \(p. 364\)](#).
- If you're using Linux platform version 1.1.0 or **laterContainer**, container health checks are supported for Fargate tasks. For more information, see [AWS Fargate platform versions \(p. 77\)](#).

command

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

When registering a task definition in the AWS Management Console, use a comma separated list of commands. These commands are converted to a string after the task definition is created. An example input for a health check is the following.

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

When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, enclose the list of commands in brackets. An example input for a health check is the following.

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

An exit code of 0, with no `stderr` output, 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 period of time (in seconds) between each health check. You can specify between 5 and 300 seconds. The default value is 30 seconds.

timeout

The period of time (in seconds) to wait for a health check to succeed before it's considered a failure. You can 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 can specify between 1 and 10 retries. The default value is three retries.

startPeriod

The optional grace period to provide containers time to bootstrap in before failed health checks count towards the maximum number of retries. You can specify between 0 and 300 seconds. By default, startPeriod is disabled.

Environment

cpu

Type: Integer

Required: Conditional

The number of cpu units the Amazon ECS container agent reserves for the container. On Linux, 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 that use the Fargate launch type. The total amount of CPU reserved for all the containers that are within a task must be lower than the task-level cpu value.

Note

You can determine the number of CPU units that are available to each Amazon EC2 instance type. To do this, multiply 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, assume that you run a single-container task on a single-core instance type with 512 CPU units specified for that container. Moreover, that task is the only task running on the container instance. In this example, the container can use the full 1,024 CPU unit share at any given time. However, assume then that you launched another copy of the same task on that container instance. Each task is guaranteed a minimum of 512 CPU units when needed. Similarly, if the other container isn't using the remaining CPU, each container can float to higher CPU usage. However, if both tasks were 100% active all of the time, they are 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 isn't required, and you can use CPU values below two in your container definitions. For CPU values below two (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. Docker then converts this value to 1,024 CPU shares. CPU values of one are passed to Docker as one, which the Linux kernel converts to two CPU shares.
- **Agent versions >= 1.2.0:** Null, zero, and CPU values of one are passed to Docker as two CPU shares.

On Windows container instances, the CPU quota is enforced as an absolute quota. Windows containers only have access to the specified amount of CPU that's defined in the task definition. A null or zero CPU value is passed to Docker as 0. Windows then interprets this value as 1% of one CPU.

For more examples, see [How Amazon ECS manages CPU and memory resources](#).

gpu

Type: [ResourceRequirement object](#)

Required: No

The number of physical GPUs that the Amazon ECS container agent reserves for the container. The number of GPUs reserved for all containers in a task must not exceed the number of available GPUs on the container instance the task is launched on. For more information, see [Working with GPUs on Amazon ECS \(p. 142\)](#).

Note

This parameter isn't supported for Windows containers or containers that are hosted on Fargate.

Elastic Inference accelerator

Type: [ResourceRequirement object](#)

Required: No

For the `InferenceAccelerator` type, the value matches the `deviceName` for an `InferenceAccelerator` specified in a task definition. For more information, see [the section called "Elastic Inference accelerator name" \(p. 834\)](#).

Note

Starting April 15, 2023, AWS will not onboard new customers to Amazon Elastic Inference (EI), and will help current customers migrate their workloads to options that offer better price and performance. After April 15, 2023, new customers will not be able to launch instances with Amazon EI accelerators in Amazon SageMaker, Amazon ECS, or Amazon EC2. However, customers who have used Amazon EI at least once during the past 30-day period are considered current customers and will be able to continue using the service.

Note

This parameter isn't supported for Windows containers or containers that are hosted on Fargate.

`essential`

Type: Boolean

Required: No

Suppose that the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason. Then, 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 doesn't 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. Suppose that you have an application that's composed of multiple containers. Then, group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see [Architecting your application for Amazon ECS \(p. 85\)](#).

`"essential": true|false`

`entryPoint`

Important

Early versions of the Amazon ECS container agent don't 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's 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, see <https://docs.docker.com/engine/reference/builder/#entrypoint>.

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

command

Type: String array

Required: No

The command that's 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, see <https://docs.docker.com/engine/reference/builder/#cmd>. If there are multiple arguments, make sure that each argument is a separated string in the array.

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

workingDirectory

Type: String

Required: No

The working directory to run commands inside the container in. 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"
```

environmentFiles

Type: Object array

Required: No

A list of files containing the environment variables to pass to a container. This parameter maps to the `--env-file` option to [docker run](#).

This isn't available for Windows containers.

You can specify up to 10 environment files. The file must have a `.env` file extension. Each line in an environment file contains an environment variable in `VARIABLE=VALUE` format. Lines that start with `#` are treated as comments and are ignored. For more information about the appropriate environment variable file syntax, see [Declare default environment variables in file](#).

If there are individual environment variables specified in the container definition, they take precedence over the variables contained within an environment file. If multiple environment files are specified that contain the same variable, they're processed from the top down. We recommend that you use unique variable names. For more information, see [Passing environment variables to a container \(p. 197\)](#).

value

Type: String

Required: Yes

The Amazon Resource Name (ARN) of the Amazon S3 object containing the environment variable file.

type

Type: String

Required: Yes

The file type to use. The only supported value is s3.

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

An object that represents the secret to expose to your container. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

name

Type: String

Required: Yes

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

valueFrom

Type: String

Required: Yes

The secret to expose to the container. The supported values are either the full Amazon Resource Name (ARN) of the AWS Secrets Manager secret or the full ARN of the parameter in the AWS Systems Manager Parameter Store.

Note

If the Systems Manager Parameter Store parameter exists in the same AWS Region as the task that you're launching, 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

disableNetworking

Type: Boolean

Required: No

When this parameter is true, networking is off within the container. This parameter maps to NetworkDisabled in the [Create a container](#) section of the [Docker Remote API](#).

Note

This parameter isn't supported for Windows containers or tasks using the awsvpc network mode.

The default is false.

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

links

Type: String array

Required: No

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

Note

This parameter isn't supported for Windows containers or tasks using the awsvpc network mode.

Important

Containers that are collocated on the same container instance might communicate with each other without requiring links or host port mappings. The network isolation on a container instance is controlled by security groups and VPC settings.

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

hostname

Type: String

Required: No

The hostname to use for your container. This parameter maps to Hostname in the [Create a container](#) section of the [Docker Remote API](#) and the --hostname option to [docker run](#).

Note

If you're using the awsvpc network mode, the hostname parameter isn't supported.

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

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 isn't supported for Windows containers or tasks using the awsvpc network mode.

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

dnsSearchDomains

Type: String array

Required: No

Pattern: ^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]\$

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

Note

This parameter isn't supported for Windows containers or tasks that use the awsvpc network mode.

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

extraHosts

Type: Object array

Required: No

A list of hostnames and IP address mappings to append to the /etc/hosts file on the container.

This parameter maps to ExtraHosts in the [Create a container](#) section of the [Docker Remote API](#) and the --add-host option to [docker run](#).

Note

This parameter isn't supported for Windows containers or tasks that use the awsvpc network mode.

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

hostname

Type: String

Required: Yes, when extraHosts are used

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

ipAddress

Type: String

Required: Yes, when extraHosts are used

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

Storage and logging

readonlyRootFilesystem

Type: Boolean

Required: No

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

Note

This parameter is not supported for Windows containers.

The default is false.

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

mountPoints

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to Volumes in the [Create a container](#) section of the [Docker Remote API](#) and the --volume option to [docker run](#).

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.

For example task definitions that use a log configuration, see [Example task definitions \(p. 213\)](#).

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 might 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 about the options for different supported log drivers, see [Configure logging drivers](#) in the Docker documentation.

Consider the following when specifying a log configuration for your containers:

- Amazon ECS supports a subset of the logging drivers that are available to the Docker daemon. Additional log drivers might be available in future releases of the Amazon ECS container agent.
- This parameter requires version 1.18 or later of the Docker Remote API on your container instance.
- For tasks that use the EC2 launch type, the Amazon ECS container agent that runs on a container instance must register the logging drivers that are available on that instance with the ECS_AVAILABLE_LOGGING_DRIVERS environment variable before containers that are placed on that instance can use these log configuration options. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).
- For tasks that use the Fargate launch type, because you don't have access to the underlying infrastructure your tasks are hosted on, any additional software needed must be installed outside of the task. For example, the Fluentd output aggregators or a remote host running Logstash to send Gelf logs to.

```
"logConfiguration": {  
    "logDriver": "awslogs", "fluentd", "gelf", "json-  
file", "journald", "logentries", "splunk", "syslog", "awsfirelens",  
    "options": {"string": "string"  
        ...},  
    "secretOptions": [{"  
        "name": "string",  
        "valueFrom": "string"  
    }]  
}
```

logDriver

Type: String

Valid values: "awslogs", "fluentd", "gelf", "json-
file", "journald", "logentries", "splunk", "syslog", "awsfirelens"

Required: Yes, when logConfiguration is used

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

For tasks that use the Fargate launch type, the supported log drivers are awslogs, splunk, and awsfirelens.

For tasks that use the EC2 launch type, the supported log drivers are awslogs, fluentd, gelf, json-file, journald, logentries, syslog, splunk, and awsfirelens.

For more information about how to use the awslogs log driver in task definitions to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 161\)](#).

For more information about using the awsfirelens log driver, see [Custom Log Routing](#).

Note

If you have a custom driver that isn't listed, you can fork the Amazon ECS container agent project that's [available on GitHub](#) and customize it to work with that driver. We encourage you to submit pull requests for changes that you want to have included. However, we don't currently support running modified copies of this software.

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 key/value map of configuration options to send to the log driver.

When you use FireLens to route logs to an AWS service or AWS Partner Network destination for log storage and analytics, you can set the `log-driver-buffer-limit` option to limit the number of events that are buffered in memory, before being sent to the log router container. It can help to resolve potential log loss issue because high throughput might result in memory running out for the buffer inside of Docker. For more information, see [the section called "Fluentd buffer limit" \(p. 168\)](#).

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

`secretOptions`

Type: Object array

Required: No

An object that represents the secret to pass to the log configuration. Secrets that are used in log configuration can include an authentication token, certificate, or encryption key. For more information, see [Passing sensitive data to a container \(p. 200\)](#).

`name`

Type: String

Required: Yes

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

`valueFrom`

Type: String

Required: Yes

The secret to expose to the log configuration of the container.

```
"logConfiguration": {  
    "logDriver": "splunk",
```

```
"options": {  
    "splunk-url": "https://cloud.splunk.com:8080",  
    "splunk-token": "...",  
    "tag": "...",  
    ...  
},  
"secretOptions": [  
    {"name": "splunk-token",  
     "valueFrom": "/ecs/logconfig/splunkcred"  
}]  
}
```

firelensConfiguration

Type: [FirelensConfiguration Object](#)

Required: No

The FireLens configuration for the container. This is used to specify and configure a log router for container logs. For more information, see [Custom log routing \(p. 166\)](#).

```
{  
    "firelensConfiguration": {  
        "type": "fluentd",  
        "options": {  
            "KeyName": ""  
        }  
    }  
}
```

options

Type: String to string map

Required: No

The key/value map of options to use when configuring the log router. This field is optional and can be used to specify a custom configuration file or to add additional metadata, such as the task, task definition, cluster, and container instance details to the log event. If specified, the syntax to use is "options": {"enable-ecs-log-metadata": "true|false", "config-file-type": "s3|file", "config-file-value": "arn:aws:s3:::mybucket/fluent.conf|filepath"}. For more information, see [Creating a task definition that uses a FireLens configuration \(p. 172\)](#).

type

Type: String

Required: Yes

The log router to use. The valid values are fluentd or fluentbit.

Security

For more information about container security, see [Task and container security](#) in the *Amazon ECS Best Practices Guide*.

credentialSpecs

Type: String array

Required: No

A list of ARNs in SSM or Amazon S3 to a credential spec (CredSpec) file that configures the container for Active Directory authentication. We recommend that you use this parameter instead of the dockerSecurityOptions. The maximum number of ARNs is 1.

There are two formats for each ARN.

credentialspecdomainless:MyARN

You use credentialspecdomainless:MyARN to provide a CredSpec with an additional section for a secret in Secrets Manager. You provide the login credentials to the domain in the secret.

Each task that runs on any container instance can join different domains.

You can use this format without joining the container instance to a domain.

credentialspec:MyARN

You use credentialspec:MyARN to provide a CredSpec for a single domain.

You must join the container instance to the domain before you start any tasks that use this task definition.

In both formats, replace MyARN with the ARN in SSM or Amazon S3.

The credspec must provide a ARN in Secrets Manager for a secret containing the username, password, and the domain to connect to. For better security, the instance isn't joined to the domain for domainless authentication. Other applications on the instance can't use the domainless credentials. You can use this parameter to run tasks on the same instance, even if the tasks need to join different domains. For more information, see [Using gMSAs for Windows Containers](#) and [Using gMSAs for Linux Containers](#).

privileged

Type: Boolean

Required: No

When this parameter is true, the container is given elevated privileges on the host container instance (similar to the root user). We recommend against running containers with privileged. In most cases, you can specify the exact privileges that you need by using the specific parameters instead of using privileged.

This parameter maps to Privileged in the [Create a container](#) section of the [Docker Remote API](#) and the --privileged option to [docker run](#).

Note

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

The default is false.

```
"privileged": true|false
```

user

Type: String

Required: No

The user 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](#).

Important

When running tasks that use the host network mode, don't run containers using the root user (UID 0). As a security best practice, always use a non-root user.

You can specify the `user` using the following formats. If specifying a UID or GID, you must specify it as a positive integer.

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

Note

This parameter is not supported for Windows containers.

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

dockerSecurityOptions

Type: String array

Valid values: `"no-new-privileges"` | `"apparmor:PROFILE"` | `"label:value"` | `"credentialspec:CredentialSpecFilePath"`

Required: No

A list of strings to provide custom configuration for multiple security systems. For more information about valid values, see [Docker Run Security Configuration](#). This field isn't valid for containers in tasks using the Fargate launch type.

For Linux tasks on EC2, this parameter can be used to reference custom labels for SELinux and AppArmor multi-level security systems.

For any tasks on EC2, this parameter can be used to reference a credential spec file that configures a container for Active Directory authentication. For more information, see [Using gMSAs for Windows Containers \(p. 416\)](#) and [Using gMSAs for Linux Containers \(p. 421\)](#).

This parameter maps to `SecurityOpt` in the [Create a container](#) section of the [Docker Remote API](#) and the `--security-opt` option to [docker](#).

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

Note

The Amazon ECS container agent that runs on a container instance must register with the `ECS_SELINUX_CAPABLE=true` or `ECS_APPARMOR_CAPABLE=true` environment variables before containers that are placed on that instance can use these security options. For more information, see [Amazon ECS container agent configuration \(p. 370\)](#).

Resource limits

ulimits

Type: Object array

Required: No

A list of ulimit values to define for a container. This value overwrites the default resource quota setting for the operating system. This parameter maps to Ulimits in the [Create a container](#) section of the [Docker Remote API](#) and the --ulimit option to [docker run](#).

Amazon ECS tasks hosted on Fargate use the default resource limit values set by the operating system with the exception of the nofile resource limit parameter. The nofile resource limit sets a restriction on the number of open files that a container can use. On Fargate, the default nofile soft limit is 1024 and hard limit is 4096. You can set the values of both limits up to 1048576. For more information, see [Task resource limits \(p. 69\)](#).

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" | "rttime" | "sigpending" | "stack"
        "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 can 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 using the console \(p. 125\)](#).

Topics

- [Linux parameters \(p. 827\)](#)
- [Container dependency \(p. 830\)](#)
- [Container timeouts \(p. 831\)](#)
- [System controls \(p. 832\)](#)
- [Interactive \(p. 833\)](#)
- [Pseudo terminal \(p. 834\)](#)

Linux parameters

linuxParameters

Type: [LinuxParameters](#) object

Required: No

Linux-specific options that are applied to the container, such as [KernelCapabilities](#).

Note

This parameter isn't supported for Windows containers.

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

capabilities

Type: [KernelCapabilities](#) object

Required: No

The Linux capabilities for the container that are added to or dropped from the default configuration provided by Docker. For more information about the default capabilities and the other available capabilities, see [Runtime privilege and Linux capabilities](#) in the *Docker run*

reference. For more information about these Linux capabilities, see the [capabilities\(7\)](#) Linux manual page.

add

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 add to the default configuration provided by Docker. This parameter maps to CapAdd in the [Create a container](#) section of the [Docker Remote API](#) and the --cap-add option to [docker run](#).

Note

Tasks that are launched on Fargate only support adding the SYS_PTRACE kernel capability.

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 that's 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](#).

devices

Any host devices to expose to the container. This parameter maps to Devices in the [Create a container](#) section of the [Docker Remote API](#) and the --device option to [docker run](#).

Note

The devices parameter isn't supported when you use the Fargate launch type, or Windows containers.

Type: Array of [Device](#) objects

Required: No

hostPath

The path for the device on the host container instance.

Type: String

Required: Yes

containerPath

The path inside the container to expose the host device at.

Type: String

Required: No

permissions

The explicit permissions to provide to the container for the device. By default, the container has permissions for `read`, `write`, and `mknod` on the device.

Type: Array of strings

Valid Values: `read` | `write` | `mknod`

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.

maxSwap

The total amount of swap memory (in MiB) a container can use. This parameter is translated to the `--memory-swap` option to [docker run](#) where the value is the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container doesn't use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container uses the swap configuration for the container instance that it's running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

Note

If you're using tasks that use the Fargate launch type, the `maxSwap` parameter isn't supported.

sharedMemorySize

The value for the size (in MiB) of the `/dev/shm` volume. This parameter maps to the `--shm-size` option to [docker run](#).

Note

If you're using tasks that use the Fargate launch type, the `sharedMemorySize` parameter isn't supported.

Type: Integer

swappiness

You can use this parameter to tune a container's memory swappiness behavior. A `swappiness` value of `0` prevents swapping from happening unless required. A `swappiness` value of `100` causes pages to be swapped frequently. Accepted values are whole numbers between `0` and `100`. If you don't specify a value, the default value of `60` is used. Moreover, if you don't specify a value for `maxSwap`, then this parameter is ignored. This parameter maps to the `--memory-swappiness` option to [docker run](#).

Note

If you're using tasks that use the Fargate launch type, the `swappiness` parameter isn't supported.

If you're using tasks on Amazon Linux 2023 the `swappiness` parameter isn't supported.

`tmpfs`

The container path, mount options, and maximum size (in MiB) of the tmpfs mount. This parameter maps to the `--tmpfs` option to [docker run](#).

Note

If you're using tasks that use the Fargate launch type, the `tmpfs` parameter isn't supported.

Type: Array of [Tmpfs](#) objects

Required: No

`containerPath`

The absolute file path where the tmpfs volume is to be mounted.

Type: String

Required: Yes

`mountOptions`

The list of tmpfs volume mount options.

Type: Array of strings

Required: No

Valid Values: "defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"

`size`

The maximum size (in MiB) of the tmpfs volume.

Type: Integer

Required: Yes

Container dependency

`dependsOn`

Type: Array of [ContainerDependency](#) objects

Required: No

The dependencies defined for container startup and shutdown. A container can contain multiple dependencies. When a dependency is defined for container startup, for container shutdown it is reversed. For an example, see [Example: Container dependency \(p. 217\)](#).

Note

If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

For Amazon ECS tasks that are hosted on Amazon EC2 instances, the instances require at least version 1.26.0 of the container agent to enable container dependencies. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#). If you're using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version 1.26.0-1 of the ecs-init package. If your container instances are launched from version 20190301 or later, they contain the required versions of the container agent and ecs-init. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

For Amazon ECS tasks that are hosted on Fargate, this parameter requires that the task or service uses platform version 1.3.0 or later (Linux) or 1.0.0 (Windows).

```
"dependsOn": [  
    {  
        "containerName": "string",  
        "condition": "string"  
    }  
]
```

containerName

Type: String

Required: Yes

The container name that must meet the specified condition.

condition

Type: String

Required: Yes

The dependency condition of the container. The following are the available conditions and their behavior:

- START – This condition emulates the behavior of links and volumes today. The condition validates that a dependent container is started before permitting other containers to start.
- COMPLETE – This condition validates that a dependent container runs to completion (exits) before permitting other containers to start. This can be useful for non-essential containers that run a script and then exit. This condition can't be set on an essential container.
- SUCCESS – This condition is the same as COMPLETE, but it also requires that the container exits with a zero status. This condition can't be set on an essential container.
- HEALTHY – This condition validates that the dependent container passes its container health check before permitting other containers to start. This requires that the dependent container has health checks configured in the task definition. This condition is confirmed only at task startup.

Container timeouts

startTimeout

Type: Integer

Required: No

Example values: 120

Time duration (in seconds) to wait before giving up on resolving dependencies for a container.

For example, you specify two containers in a task definition with `containerA` having a dependency on `containerB` reaching a COMPLETE, SUCCESS, or HEALTHY status. If a `startTimeout` value is specified for `containerB` and it doesn't reach the desired status within that time, then `containerA` doesn't start.

Note

If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

For Amazon ECS tasks that are hosted on Fargate, this parameter requires that the task or service uses platform version 1.3.0 or later (Linux).

`stopTimeout`

Type: Integer

Required: No

Example values: 120

Time duration (in seconds) to wait before the container is forcefully killed if it doesn't exit normally on its own.

For tasks that use the Fargate launch type, the task or service requires platform version 1.3.0 or later (Linux) or 1.0.0 or later (for Windows). The max stop timeout value is 120 seconds. However, if the parameter isn't specified, the default value of 30 seconds is used.

For tasks that use the EC2 launch type, if the `stopTimeout` parameter isn't specified, the value set for the Amazon ECS container agent configuration variable `ECS_CONTAINER_STOP_TIMEOUT` is used. If neither the `stopTimeout` parameter or the `ECS_CONTAINER_STOP_TIMEOUT` agent configuration variable is set, the default values of 30 seconds for Linux containers and 30 seconds on Windows containers are used. Container instances require at least version 1.26.0 of the container agent to enable a container stop timeout value. However, we recommend using the latest container agent version. For information about how to check your agent version and update to the latest version, see [Updating the Amazon ECS container agent \(p. 364\)](#). If you're using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version 1.26.0-1 of the `ecs-init` package. If your container instances are launched from version 20190301 or later, they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

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](#). For example, you can configure `net.ipv4.tcp_keepalive_time` setting to maintain longer lived connections.

We don't recommend that you specify network-related `systemControls` parameters for multiple containers in a single task that also uses either the `awsvpc` or host network mode. Doing this has the following disadvantages:

- For tasks that use the awsvpc network mode including Fargate, if you set `systemControls` for any container, it applies to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that's started last determines which `systemControls` take effect.
- For tasks that use the host network mode, the network namespace `systemControls` aren't supported.

If you're setting an IPC resource namespace to use for the containers in the task, the following conditions apply to your system controls. For more information, see [IPC mode \(p. 842\)](#).

- For tasks that use the host IPC mode, IPC namespace `systemControls` aren't supported.
- For tasks that use the task IPC mode, IPC namespace `systemControls` values apply to all containers within a task.

Note

This parameter is not supported for Windows containers.

Note

This parameter is only supported for tasks that are hosted on AWS Fargate if the tasks are using platform version 1.4.0 or later (Linux). This isn't supported for Windows containers on Fargate.

```
"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.shmmmax" | "kernel.shmmnmi" | "kernel.shm_rmid_forced", and Sysctls that start with "fs.mqueue.*"

Valid network namespace values: Sysctls that start with "net.*"

All of these values are supported by Fargate.

value

Type: String

Required: No

The value for the namespaced kernel parameter that's specified in `namespace`.

Interactive

interactive

Type: Boolean

Required: No

When this parameter is `true`, you can 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](#).

The default is `false`.

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](#).

The default is `false`.

Elastic Inference accelerator name

Note

Starting April 15, 2023, AWS will not onboard new customers to Amazon Elastic Inference (EI), and will help current customers migrate their workloads to options that offer better price and performance. After April 15, 2023, new customers will not be able to launch instances with Amazon EI accelerators in Amazon SageMaker, Amazon ECS, or Amazon EC2. However, customers who have used Amazon EI at least once during the past 30-day period are considered current customers and will be able to continue using the service.

The Elastic Inference accelerator resource requirement for your task definition. For more information, see [What Is Elastic Inference?](#) in the Amazon Elastic Inference Developer Guide.

The following parameters are allowed in a task definition:

`deviceName`

Type: String

Required: Yes

The Elastic Inference accelerator device name. The `deviceName` must also be referenced in a container definition see [Elastic Inference accelerator \(p. 814\)](#).

`deviceType`

Type: String

Required: Yes

The Elastic Inference accelerator to use.

Task placement constraints

When you register a task definition, you can provide task placement constraints that customize how Amazon ECS places tasks.

If you're using the Fargate launch type, task placement constraints aren't supported. By default Fargate tasks are spread across Availability Zones.

For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see [Amazon ECS task placement constraints \(p. 437\)](#).

The following parameters are allowed in a container definition:

expression

Type: String

Required: No

A cluster query language expression to apply to the constraint. For more information, see [Cluster query language \(p. 442\)](#).

type

Type: String

Required: Yes

The type of constraint. Use memberOf to restrict the selection to a group of valid candidates.

Proxy configuration

proxyConfiguration

Type: [ProxyConfiguration](#) object

Required: No

The configuration details for the App Mesh proxy.

For tasks that use the EC2 launch type, the container instances require at least version 1.26.0 of the container agent and at least version 1.26.0-1 of the ecs-init package to enable a proxy configuration. If your container instances are launched from the Amazon ECS-optimized AMI version 20190301 or later, then they contain the required versions of the container agent and ecs-init. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).

For tasks that use the Fargate launch type, this feature requires that the task or service uses platform version 1.3.0 or later.

Note

This parameter is not supported for Windows containers.

```
"proxyConfiguration": {  
    "type": "APPMESH",  
    "containerName": "string",  
    "properties": [  
        {  
            "name": "string",  
            "value": "string"  
        }  
    ]  
}
```

type

Type: String

Value values: APPMESH

Required: No

The proxy type. The only supported value is APPMESH.

`containerName`

Type: String

Required: Yes

The name of the container that serves as the App Mesh proxy.

`properties`

Type: Array of [KeyValuePair](#) objects

Required: No

The set of network configuration parameters to provide the Container Network Interface (CNI) plugin, specified as key-value pairs.

- `IgnoredUID` – (Required) The user ID (UID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredGID` is specified, this field can be empty.
- `IgnoredGID` – (Required) The group ID (GID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredUID` is specified, this field can be empty.
- `AppPorts` – (Required) The list of ports that the application uses. Network traffic to these ports is forwarded to the `ProxyIngressPort` and `ProxyEgressPort`.
- `ProxyIngressPort` – (Required) Specifies the port that incoming traffic to the `AppPorts` is directed to.
- `ProxyEgressPort` – (Required) Specifies the port that outgoing traffic from the `AppPorts` is directed to.
- `EgressIgnoredPorts` – (Required) The outbound traffic going to these specified ports is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.
- `EgressIgnoredIPs` – (Required) The outbound traffic going to these specified IP addresses is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.

`name`

Type: String

Required: No

The name of the key-value pair.

`value`

Type: String

Required: No

The value of the key-value pair.

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.

The following are the types of data volumes that can be used:

- Docker volumes — A Docker-managed volume that is created under `/var/lib/docker/volumes` on the host Amazon EC2 instance. Docker volume drivers (also referred to as plugins) are used to integrate the volumes with external storage systems, such as Amazon EBS. The built-in local volume driver or a third-party volume driver can be used. Docker volumes are only supported when running tasks on Amazon EC2 instances. Windows containers only support the use of the local driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Using volumes](#).
- Bind mounts — A file or directory on the host machine is mounted into a container. Bind mount host volumes are supported when running tasks on either AWS Fargate or Amazon EC2 instances. To use bind mount host volumes, specify a `host` and optional `sourcePath` value in your task definition. For more information, see [Using bind mounts](#).

For more information, see [Using data volumes in tasks \(p. 100\)](#).

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` object.

`host`

Required: No

Note

The `host` parameter is only supported when using tasks hosted on Amazon EC2 instances.

The `host` parameter is used to tie the lifecycle of the bind mount to the host Amazon EC2 instance, rather than the task, 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.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.

`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host Amazon EC2 instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host Amazon EC2 instance until you delete it manually. If the `sourcePath` value does not exist on the host Amazon EC2 instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

`dockerVolumeConfiguration`

Type: [DockerVolumeConfiguration](#) Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when running tasks on EC2 instances. Windows containers only support the use of the local driver. To use bind mounts, specify a host instead.

scope

Type: String

Valid Values: task | shared

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a task are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as shared persist after the task stops.

autoprovision

Type: Boolean

Default value: false

Required: No

If this value is true, the Docker volume is created if it does not already exist. This field is only used if the scope is shared. If the scope is task then this parameter must either be omitted or set to false.

driver

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use docker plugin ls to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see [Docker plugin discovery](#). This parameter maps to Driver in the [Create a volume](#) section of the [Docker Remote API](#) and the --driver option to [docker volume create](#).

driverOpts

Type: String

Required: No

A map of Docker driver specific options to pass through. This parameter maps to DriverOpts in the [Create a volume](#) section of the [Docker Remote API](#) and the --opt option to [docker volume create](#).

labels

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to Labels in the [Create a volume](#) section of the [Docker Remote API](#) and the --label option to [docker volume create](#).

efsVolumeConfiguration

Type: [EFSVolumeConfiguration](#) Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying / will have the same effect as omitting this parameter.

Important

If an EFS access point is specified in the `authorizationConfig`, the `rootDirectory` parameter must either be omitted or set to / which will enforce the path set on the EFS access point.

`transitEncryption`

Type: String

Valid values: ENABLED | DISABLED

Required: No

Whether or not to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. Transit encryption must be enabled if Amazon EFS IAM authorization is used. If this parameter is omitted, the default value of DISABLED is used. For more information, see [Encrypting Data in Transit](#) in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you do not specify a transit encryption port, it will use the port selection strategy that the Amazon EFS mount helper uses. For more information, see [EFS Mount Helper](#) in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: [EFSAuthorizationConfiguration](#) Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the efsVolumeConfiguration must either be omitted or set to / which will enforce the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the EFSVolumeConfiguration. For more information, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

iam

Type: String

Valid values: ENABLED | DISABLED

Required: No

Whether or not to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the EFSVolumeConfiguration. If this parameter is omitted, the default value of DISABLED is used. For more information, see [IAM Roles for Tasks](#).

FSxWindowsFileServerVolumeConfiguration

Type: [FSxWindowsFileServerVolumeConfiguration](#) Object

Required: Yes

This parameter is specified when you are using the [FSx for Windows File Server](#) file system for task storage.

fileSystemId

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

rootDirectory

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

authorizationConfig

credentialsParameter

Type: String

Required: Yes

The authorization credential options.

options:

- Amazon Resource Name (ARN) of an [AWS Secrets Manager](#) secret.
- ARN of an [AWS Systems Manager](#) parameter.

domain

Type: String

Required: Yes

A fully qualified domain name hosted by an [AWS Directory Service](#) Managed Microsoft AD (Active Directory) or self-hosted EC2 AD.

Tags

When you register a task definition, you can optionally specify metadata tags that are applied to the task definition. Tags help you categorize and organize your task definition. Each tag consists of a key and an optional value. You define both of them. For more information, see [Tagging your Amazon ECS resources \(p. 529\)](#).

Important

Don't add personally identifiable information or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags aren't intended to be used for private or sensitive data.

The following parameters are allowed in a tag object.

key

Type: String

Required: No

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

Type: String

Required: No

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

Other task definition parameters

The following task definition parameters can 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 using the console \(p. 125\)](#).

Topics

- [Ephemeral storage \(p. 841\)](#)
- [IPC mode \(p. 842\)](#)
- [PID mode \(p. 842\)](#)

Ephemeral storage

ephemeralStorage

Type: [EphemeralStorage](#) object

Required: No

The amount of ephemeral storage (in GB) to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks that are hosted on AWS Fargate. For more information, see [the section called "Bind mounts" \(p. 114\)](#).

Note

This parameter is only supported for tasks that are hosted on AWS Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).

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 the containers that are 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 the containers that are 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 value for ipcMode is set to shareable. For more information, see [IPC settings in the Docker run reference](#).

If the host IPC mode is used, there's a heightened risk of undesired IPC namespace exposure. For more information, see [Docker security](#).

If you're setting namespaced kernel parameters that use systemControls for the containers in the task, the following applies to your IPC resource namespace. For more information, see [System controls \(p. 832\)](#).

- For tasks that use the host IPC mode, IPC namespace that's related systemControls aren't supported.
- For tasks that use the task IPC mode, systemControls that relate to the IPC namespace 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

Valid Values: host | task

Required: No

The process namespace to use for the containers in the task. The valid values are host or task. On Fargate for Linux containers, the only valid value is task. For example, monitoring sidecars might need pidMode to access information about other containers running in the same task.

If host is specified, all containers within the tasks that specified the host PID mode on the same container instance share the same process namespace 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 each container. For more information, see [PID settings](#) in the *Docker run* reference.

If the host PID mode is used, there's a heightened risk of undesired process namespace exposure. For more information, see [Docker security](#).

Note

This parameter is not supported for Windows containers.

Note

This parameter is only supported for tasks that are hosted on AWS Fargate if the tasks are using platform version 1.4.0 or later (Linux). This isn't supported for Windows containers on Fargate.

Task definition template

An empty task definition template is shown as follows. 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 \(p. 799\)](#).

```
{  
    "family": "",  
    "taskRoleArn": "",  
    "executionRoleArn": "",  
    "networkMode": "none",  
    "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": ""  
                }  
            ],  
            "environmentFiles": [  
                {  
                    ""  
                }  
            ]  
        }  
    ]  
}
```

```
        "value": "",
        "type": "s3"
    }
],
"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": [
            ""
        ]
    }
],
"maxSwap": 0,
"swappiness": 0
},
"secrets": [
    {
        "name": "",
        "valueFrom": ""
    }
],
"dependsOn": [
    {
        "containerName": "",
        "condition": "COMPLETE"
    }
],
"startTimeout": 0,
"stopTimeout": 0,
"hostname": "",
"user": "",
"workingDirectory": "".
```

```
"disableNetworking": true,
"privileged": true,
"readonlyRootFilesystem": true,
"dnsServers": [
    ""
],
"dnsSearchDomains": [
    ""
],
"extraHosts": [
    {
        "hostname": "",
        "ipAddress": ""
    }
],
"dockerSecurityOptions": [
    ""
],
"interactive": true,
"pseudoTerminal": true,
"dockerLabels": {
    "KeyName": ""
},
"ulimits": [
    {
        "name": "nofile",
        "softLimit": 0,
        "hardLimit": 0
    }
],
"logConfiguration": {
    "logDriver": "splunk",
    "options": {
        "KeyName": ""
    }
},
"secretOptions": [
    {
        "name": "",
        "valueFrom": ""
    }
],
},
"healthCheck": {
    "command": [
        ""
    ],
    "interval": 0,
    "timeout": 0,
    "retries": 0,
    "startPeriod": 0
},
"systemControls": [
    {
        "namespace": "",
        "value": ""
    }
],
"resourceRequirements": [
    {
        "value": "",
        "type": "InferenceAccelerator"
    }
],
"firelensConfiguration": {
    "type": "fluentd",
    "options": {
        "logFormat": "json"
    }
}
```

```

        "KeyName": ""
    }
}
],
"volumes": [
{
    "name": "",
    "host": {
        "sourcePath": ""
    },
    "dockerVolumeConfiguration": {
        "scope": "shared",
        "autoprovision": true,
        "driver": "",
        "driverOpts": {
            "KeyName": ""
        },
        "labels": {
            "KeyName": ""
        }
    },
    "efsVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "transitEncryption": "DISABLED",
        "transitEncryptionPort": 0,
        "authorizationConfig": {
            "accessPointId": "",
            "iam": "ENABLED"
        }
    },
    "fsxWindowsFileServerVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "authorizationConfig": {
            "credentialsParameter": "",
            "domain": ""
        }
    }
}
],
"placementConstraints": [
{
    "type": "memberOf",
    "expression": ""
}
],
"requiresCompatibilities": [
    "EC2"
],
"cpu": "",
"memory": "",
"tags": [
{
    "key": "",
    "value": ""
}
],
"pidMode": "task",
"ipcMode": "task",
"proxyConfiguration": {
    "type": "APPmesh",
    "containerName": "",
    "properties": [
    {

```

```
        "name": "",  
        "value": ""  
    }  
]  
,  
"inferenceAccelerators": [  
    {  
        "deviceName": "",  
        "deviceType": ""  
    }  
,  
"ephemeralStorage": {  
    "sizeInGiB": 0  
},  
"runtimePlatform": {  
    "cpuArchitecture": "X86_64",  
    "operatingSystemFamily": "WINDOWS_SERVER_20H2_CORE"  
}  
]
```

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

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

Service definition parameters

A service definition defines how to run your Amazon ECS service. The following parameters can be specified in a service definition.

Launch type

`launchType`

Type: String

Valid values: EC2 | FARGATE | EXTERNAL

Required: No

The launch type on which to run your service. If a launch type is not specified, the default `capacityProviderStrategy` is used by default. For more information, see [Amazon ECS launch types \(p. 85\)](#).

If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

Capacity provider strategy

`capacityProviderStrategy`

Type: Array of objects

Required: No

The capacity provider strategy to use for the service.

A capacity provider strategy consists of one or more capacity providers along with the base and weight to assign to them. A capacity provider must be associated with the cluster to be used in

a capacity provider strategy. The PutClusterCapacityProviders API is used to associate a capacity provider with a cluster. Only capacity providers with an ACTIVE or UPDATING status can be used.

If a capacityProviderStrategy is specified, the launchType parameter must be omitted. If no capacityProviderStrategy or launchType is specified, the defaultCapacityProviderStrategy for the cluster is used.

If you want to specify a capacity provider that uses an Auto Scaling group, the capacity provider must already be created. New capacity providers can be created with the CreateCapacityProvider API operation.

To use an AWS Fargate capacity provider, specify either the FARGATE or FARGATE_SPOT capacity providers. The AWS Fargate capacity providers are available to all accounts and only need to be associated with a cluster to be used.

The PutClusterCapacityProviders API operation is used to update the list of available capacity providers for a cluster after the cluster is created.

capacityProvider

Type: String

Required: Yes

The short name or full Amazon Resource Name (ARN) of the capacity provider.

weight

Type: Integer

Valid range: Integers between 0 and 1,000.

Required: No

The weight value designates the relative percentage of the total number of tasks launched that use the specified capacity provider.

For example, assume that you have a strategy that contains two capacity providers and both have a weight of one. When the base is satisfied, the tasks split evenly across the two capacity providers. Using that same logic, assume that you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*. Then, for every one task that is run using *capacityProviderA*, four tasks use *capacityProviderB*.

base

Type: Integer

Valid range: Integers between 0 and 100,000.

Required: No

The base value designates how many tasks, at a minimum, to run on the specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

Task definition

taskDefinition

Type: String

Required: No

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

A task definition must be specified when using the rolling update (ECS) deployment controller.

Platform operating system

platformFamily

Type: string

Required: Conditional

Default: Linux

This parameter is required for Amazon ECS services hosted on Fargate.

This parameter is ignored for Amazon ECS services hosted on Amazon EC2.

The operating system on the containers that runs the service. The valid values are `LINUX`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2022_FULL`, and `WINDOWS_SERVER_2022_CORE`.

The `platformFamily` value for every task that you specify for the service must match the service `platformFamily` value. For example, if you set the `platformFamily` to `WINDOWS_SERVER_2019_FULL`, the `platformFamily` value for all the tasks must be `WINDOWS_SERVER_2019_FULL`.

Platform version

platformVersion

Type: String

Required: No

The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. 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 \(p. 77\)](#).

Note

Platform versions are not specified for tasks using the EC2 launch type.

Cluster

cluster

Type: String

Required: No

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.

Service name

`serviceName`

Type: String

Required: Yes

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.

Scheduling strategy

`schedulingStrategy`

Type: String

Valid values: REPLICA | DAEMON

Required: No

The scheduling strategy to use. If no scheduling strategy is specified, the REPLICA strategy is used. For more information, see [Service scheduler concepts \(p. 453\)](#).

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 \(p. 455\)](#).
- 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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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 \(p. 453\)](#).

Note

Fargate tasks do not support the DAEMON scheduling strategy.

Desired count

`desiredCount`

Type: Integer

Required: No

The number of instantiations of the specified task definition to place and keep running in your service.

This parameter is required if the REPLICA scheduling strategy is used. If the service uses the DAEMON scheduling strategy, this parameter is optional.

Deployment configuration

deploymentConfiguration

Type: Object

Required: No

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

maximumPercent

Type: Integer

Required: No

If a service is using the rolling update (ECS) deployment type, the maximumPercent parameter represents an upper limit on the number of your service's tasks that are allowed in the RUNNING, STOPPING, or PENDING state during a deployment. It is expressed as a percentage of the desiredCount that is rounded down to the nearest integer. You can use this parameter to define the deployment batch size. For example, if your service is using the REPLICA service scheduler and has a desiredCount of four tasks and a maximumPercent value of 200%, the scheduler might start four new tasks before stopping the four older tasks. This is provided that the cluster resources required to do this are available. The default maximumPercent value for a service using the REPLICA service scheduler is 200%.

If your service is using the DAEMON service scheduler type, the maximumPercent should remain at 100%. This 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.

If a service is using either the blue/green (CODE_DEPLOY) or EXTERNAL deployment types and tasks that use the EC2 launch type, the **maximum percent** value is set to the default value and is used to define the upper limit on the number of the tasks in the service that remain in the RUNNING state while the container instances are in the DRAINING state. If the tasks in the service use the Fargate launch type, the maximum percent value isn't used, although it's returned when describing your service.

minimumHealthyPercent

Type: Integer

Required: No

If a service is using the rolling update (ECS) deployment type, the minimumHealthyPercent represents a lower limit on the number of your service's tasks that must remain in the RUNNING state during a deployment. This is expressed as a percentage of the desiredCount that is rounded up to the nearest integer. You can use this parameter to deploy without using additional cluster capacity. For example, if your service has a desiredCount of four tasks and a minimumHealthyPercent of 50%, the service scheduler might stop two existing tasks to free up cluster capacity before starting two new tasks.

For services that *do not* use a load balancer, consider the following:

- A service is considered healthy if all essential containers within the tasks in the service pass their health checks.

- If a task has no essential containers with a health check defined, the service scheduler waits for 40 seconds after a task reaches a RUNNING state before the task is counted towards the minimum healthy percent total.
- If a task has one or more essential containers with a health check defined, the service scheduler waits for the task to reach a healthy status before counting it towards the minimum healthy percent total. A task is considered healthy when all essential containers within the task have passed their health checks. The amount of time the service scheduler can wait for is determined by the container health check settings. For more information, see [Health check \(p. 811\)](#).

For services that *do* use a load balancer, consider the following:

- If a task has no essential containers with a health check defined, the service scheduler waits for the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.
- If a task has an essential container with a health check defined, the service scheduler waits for both the task to reach a healthy status and the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.

The default value for a replica service for `minimumHealthyPercent` is 100%. The default `minimumHealthyPercent` value for a service using the DAEMON service schedule 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.

If a service is using either the blue/green (CODE_DEPLOY) or EXTERNAL deployment types and is running tasks that use the EC2 launch type, the **minimum healthy percent** value is set to the default value and is used to define the lower limit on the number of the tasks in the service that remain in the RUNNING state while the container instances are in the DRAINING state. If a service is using either the blue/green (CODE_DEPLOY) or EXTERNAL deployment types and is running tasks that use the Fargate launch type, the minimum healthy percent value is not used, although it is returned when describing your service.

Deployment controller

`deploymentController`

Type: Object

Required: No

The deployment controller to use for the service. If no deployment controller is specified, the ECS controller is used. For more information, see [Amazon ECS Deployment types \(p. 472\)](#).

`type`

Type: String

Valid values: ECS | CODE_DEPLOY | EXTERNAL

Required: yes

The deployment controller type to use. There are three deployment controller types available: ECS

The rolling update (ECS) deployment type involves replacing the current running version of the container with the latest version. The number of containers Amazon ECS adds or

removes from the service during a rolling update is controlled by adjusting the minimum and maximum number of healthy tasks allowed during a service deployment, as specified in the [deploymentConfiguration](#).

CODE_DEPLOY

The blue/green (CODE_DEPLOY) deployment type uses the blue/green deployment model powered by CodeDeploy, which allows you to verify a new deployment of a service before sending production traffic to it.

EXTERNAL

Use the external deployment type when you want to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service.

Task placement

placementConstraints

Type: Array of objects

Required: No

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task. This limit includes constraints in the task definition and those specified at run time. If you use the Fargate launch type, task placement constraints aren't supported.

type

Type: String

Required: No

The type of constraint. Use `distinctInstance` to ensure that each task in a particular group is running on a different container instance. Use `memberOf` to restrict the selection to a group of valid candidates. The value `distinctInstance` is not supported in task definitions.

expression

Type: String

Required: No

A cluster query language expression to apply to the constraint. You can't specify an expression if the constraint type is `distinctInstance`. For more information, see [Cluster query language \(p. 442\)](#).

placementStrategy

Type: Array of objects

Required: No

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

type

Type: String

Valid values: `random` | `spread` | `binpack`

Required: No

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's 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

Type: String

Required: No

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's 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.

Tags

tags

Type: Array of objects

Required: No

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. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see [Tagging your Amazon ECS resources \(p. 529\)](#).

key

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

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

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

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

enableECSManagedTags

Type: Boolean

Valid values: `true` | `false`

Required: No

Specifies whether to use Amazon ECS managed tags for the tasks in the service. If no value is specified, the default value is `false`. For more information, see [Tagging your resources for billing \(p. 532\)](#).

`propagateTags`

Type: String

Valid values: TASK_DEFINITION | SERVICE

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the `TagResource` API action.

Network configuration

`networkConfiguration`

Type: Object

Required: No

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 isn't supported for other network modes. If using the Fargate launch type, the `awsvpc` network mode is required. For more information, see [Task networking for tasks that are hosted on Amazon EC2 instances \(p. 90\)](#).

`awsvpcConfiguration`

Type: Object

Required: No

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

`subnets`

Type: Array of strings

Required: Yes

The subnets that are associated with the task or service. There is a limit of 16 subnets that can be specified according to `awsvpcConfiguration`.

`securityGroups`

Type: Array of strings

Required: No

The security groups associated with the task or service. If you don't specify a security group, the default security group for the VPC is used. There's a limit of five security groups that can be specified based on `awsvpcConfiguration`.

`assignPublicIP`

Type: String

Valid values: ENABLED | DISABLED

Required: No

Whether the task's elastic network interface receives a public IP address. If no value is specified, the default value of DISABLED is used.

healthCheckGracePeriodSeconds

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health checks after a task enters a RUNNING state. This is only valid if your service is configured to use a load balancer. If your service has a load balancer defined and you do not specify a health check grace period value, the default value of 0 is used.

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 2,147,483,647 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.

If you do not use an Elastic Load Balancing, we recommend that you use the `startPeriod` in the task definition health check parameters. For more information, see [Health check](#).

loadBalancers

Type: Array of objects

Required: No

A load balancer object representing the load balancers to use with your service. For services that use an Application Load Balancer or Network Load Balancer, there's a limit of five target groups that you can attach to a service.

After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the ECS rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*.

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.

targetGroupArn

Type: String

Required: No

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

A target group ARN is only specified when using an Application Load Balancer or Network Load Balancer.

`loadBalancerName`

Type: String

Required: No

The name of the load balancer to associate with the service.

If you're using an Application Load Balancer or a Network Load Balancer, omit the `loadBalancerName` parameter.

`containerName`

Type: String

Required: No

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

`containerPort`

Type: Integer

Required: No

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the task definition used by tasks in the service. For tasks that use the EC2 launch type, the container instance must allow inbound traffic on the `hostPort` of the port mapping.

`role`

Type: String

Required: No

The short name or full ARN of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only permitted if you are using a load balancer with a single target group for your service, and your task definition does not use the `awsvpc` network mode. 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*.

Important

If your account has already created the Amazon ECS service-linked role, that role is used by default for your service unless you specify a role here. The service-linked role is required if your task definition uses the `awsvpc` network mode, in which case you should not specify a role here. For more information, see [Using service-linked roles for Amazon ECS \(p. 624\)](#).

`serviceConnectConfiguration`

Type: Object

Required: No

The configuration for this service to discover and connect to services, and be discovered by, and connected from, other services within a namespace.

For more information, see [Service Connect \(p. 503\)](#).

enabled

Type: Boolean

Required: Yes

Specifies whether to use Service Connect with this service.

namespace

Type: String

Required: No

The short name or full Amazon Resource Name (ARN) of the AWS Cloud Map namespace for use with Service Connect. The namespace must be in the same AWS Region as the Amazon ECS service and cluster. The type of namespace doesn't affect Service Connect. For more information about AWS Cloud Map, see [Working with Services](#) in the *AWS Cloud Map Developer Guide*.

services

Type: Array of objects

Required: No

An array of Service Connect service objects. These are names and aliases (also known as endpoints) that are used by other Amazon ECS services to connect to this service.

This field isn't required for a "client" Amazon ECS service that's a member of a namespace only to connect to other services within the namespace. An example is frontend application that accepts incoming requests from either a load balancer that's attached to the service or by other means.

An object selects a port from the task definition, assigns a name for the AWS Cloud Map service, and an array of aliases (also known as endpoints) and ports for client applications to refer to this service.

portName

Type: String

Required: Yes

The portName must match the name of one of the portMappings from all of the containers in the task definition of this Amazon ECS service.

discoveryName

Type: String

Required: No

The discoveryName is the name of the new AWS Cloud Map service that Amazon ECS creates for this Amazon ECS service. This must be unique within the AWS Cloud Map namespace.

If this field isn't specified, portName is used.

clientAliases

Type: Array of objects

Required: No

The list of client aliases for this service connect service. You use these to assign names that can be used by client applications. The maximum number of client aliases that you can have in this list is 1.

Each alias ("endpoint") is a DNS name and port number that other Amazon ECS services ("clients") can use to connect to this service.

Each name and port combination must be unique within the namespace.

These names are configured within each task of the client service, not in AWS Cloud Map. DNS requests to resolve these names don't leave the task, and don't count toward the quota of DNS requests per second per elastic network interface.

port

Type: Integer

Required: Yes

The listening port number for the service connect proxy. This port is available inside of all of the tasks within the same namespace.

To avoid changing your applications in client Amazon ECS services, set this to the same port that the client application uses by default.

dnsName

Type: String

Required: No

The dnsName is the name that you use in the applications of client tasks to connect to this service. The name must be a valid DNS label.

The default value is the discoveryName . namespace if this field is not specified. If the discoveryName isn't specified, the portName from the task definition is used.

To avoid changing your applications in client Amazon ECS services, set this to the same name that the client application uses by default. For example, a few common names are database, db, or the lowercase name of a database, such as mysql or redis.

ingressPortOverride

Type: Integer

Required: No

(Optional) The port number for the Service Connect proxy to listen on.

Use the value of this field to bypass the proxy for traffic on the port number that's specified in the named portMapping in the task definition of this application, and then use it in your Amazon VPC security groups to allow traffic into the proxy for this Amazon ECS service.

In awsvpc mode, the default value is the container port number that's specified in the named portMapping in the task definition of this application. In bridge mode, the default value is the dynamic ephemeral port of the Service Connect proxy.

logConfiguration

Type: [LogConfiguration](#) Object

Required: No

This defines where the Service Connect proxy logs are published. Use the logs for debugging during unexpected events. This configuration sets the `logConfiguration` parameter in the Service Connect proxy container in each task in this Amazon ECS service. The proxy container isn't specified in the task definition.

We recommend that you use the same log configuration as the application containers of the task definition for this Amazon ECS service. For FireLens, this is the log configuration of the application container. It's not the FireLens log router container that uses the `fluent-bit` or `fluentd` container image.

`serviceRegistries`

Type: Array of objects

Required: No

The details of the service discovery configuration for your service. For more information, see [Service discovery \(p. 522\)](#).

`registryArn`

Type: String

Required: No

The Amazon Resource Name (ARN) of the service registry. The currently supported service registry is AWS Cloud Map. For more information, see [Working with Services](#) in the *AWS Cloud Map Developer Guide*.

`port`

Type: Integer

Required: No

The port value that's 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`

Type: String

Required: No

The container name value to be used for your service discovery service. This value is specified in the task definition. 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`

Type: Integer

Required: No

The port value to be used for your service discovery service. This value is specified in the task definition. 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.

Client token

clientToken

Type: String

Required: No

The unique, case-sensitive identifier that you provide to ensure the idempotency of the request. It can be up to 32 ASCII characters long.

Service definition template

The following shows the JSON representation of an Amazon ECS service definition.

```
{  
    "cluster": "",  
    "serviceName": "",  
    "taskDefinition": "",  
    "loadBalancers": [  
        {  
            "targetGroupArn": "",  
            "loadBalancerName": "",  
            "containerName": "",  
            "containerPort": 0  
        }  
    ],  
    "serviceRegistries": [  
        {  
            "registryArn": "",  
            "port": 0,  
            "containerName": "",  
            "containerPort": 0  
        }  
    ],  
    "desiredCount": 0,  
    "clientToken": "",  
    "launchType": "FARGATE",  
    "capacityProviderStrategy": [  
        {  
            "capacityProvider": "",  
            "weight": 0,  
            "base": 0  
        }  
    ],  
    "platformVersion": "",  
    "platformFamily": "",  
    "role": "",  
    "deploymentConfiguration": {  
        "maximumPercent": 0,  
        "minimumHealthyPercent": 0  
    },  
    "placementConstraints": [  
        {  
            "type": "distinctInstance",  
            "expression": ""  
        }  
    ],  
    "placementStrategy": [  
        {  
            "type": "spread",  
            "order": 1  
        }  
    ]  
}
```

```
        "field": ""
    },
],
"networkConfiguration": {
    "awsvpcConfiguration": {
        "subnets": [
            ""
        ],
        "securityGroups": [
            ""
        ],
        "assignPublicIp": "ENABLED"
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "SERVICE"
}
```

You can create this service definition template using the following AWS CLI command.

```
aws ecs create-service --generate-cli-skeleton
```

Classic Amazon Elastic Container Service console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can use the classic console to configure your Amazon ECS resources.

Topics

- [Getting started with Amazon ECS using the classic console \(p. 863\)](#)
- [Cluster management in the classic Amazon ECS console \(p. 879\)](#)
- [Task definition management in the classic Amazon ECS console \(p. 887\)](#)
- [Task management in the classic Amazon ECS console \(p. 896\)](#)
- [Service management in the classic Amazon ECS console \(p. 899\)](#)
- [Tag management in the classic Amazon ECS console \(p. 914\)](#)
- [Account setting management in the classic Amazon ECS console \(p. 915\)](#)
- [Container instance management in the classic Amazon ECS console \(p. 916\)](#)
- [Container agent management in the classic Amazon ECS console \(p. 921\)](#)
- [Monitoring and troubleshooting in the classic Amazon ECS console \(p. 922\)](#)

Getting started with Amazon ECS using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

The following guides provide an introduction to the classic AWS Management Console to complete the common tasks to run your containers on Amazon ECS and AWS Fargate.

Contents

- [Getting started with the classic console using Linux containers on AWS Fargate \(p. 864\)](#)
- [Getting started with the classic Amazon ECS console using Windows containers on AWS Fargate \(p. 867\)](#)
- [Getting started with the classic console using Amazon EC2 \(p. 871\)](#)
- [Getting started with Windows containers using the classic console \(p. 875\)](#)

Getting started with the classic console using Linux containers on AWS Fargate

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For a broad overview on Amazon ECS on Fargate, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks. In the Regions where Amazon ECS supports AWS Fargate, the classic Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using the Fargate launch type. 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.

Complete the following steps to get started with Amazon ECS on AWS Fargate.

Prerequisites

Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or [Amazon ECS first-run wizard permissions \(p. 601\)](#) 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 [Set up to use Amazon ECS \(p. 10\)](#).
- 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 \(p. 626\)](#).

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 classic 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 [Amazon ECS on AWS Fargate \(p. 67\)](#).

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 \(p. 804\)](#).

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 \(p. 799\)](#).

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 \(p. 486\)](#).
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 takes care 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.

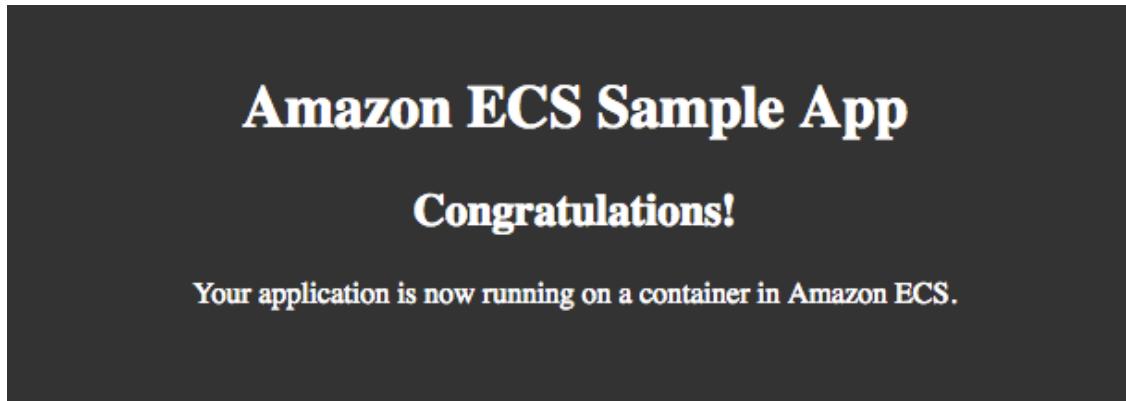
Step 4: Review

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: 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.



Step 6: Clean up

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

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

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, select the cluster to delete.
4. Choose **Delete Cluster**. At the confirmation prompt, enter **delete me** and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Getting started with the classic Amazon ECS console using Windows containers on AWS Fargate

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For a broad overview on Amazon ECS on Fargate, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks. In the Regions where Amazon ECS supports AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using the Fargate launch type. 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.

Prerequisites

Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or [Amazon ECS first-run wizard permissions \(p. 601\)](#) 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 [Set up to use Amazon ECS \(p. 10\)](#).
- 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 \(p. 626\)](#).

Step 1: Create a cluster

You can create a new cluster called windows.

To create a cluster with the AWS Management Console

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

2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create Cluster**.
4. Choose **Networking only** and choose **Next step**.
5. For **Cluster name** enter a name for your cluster (in this example, windows is the name of the cluster). Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.
 - a. To create a new VPC, select **CreateVPC**.
 - b. (Optional) If you chose to create a new VPC, for **CIDR Block**, enter a CIDR block for your VPC. For more information, see [Your VPC and Subnets](#) in the *Amazon VPC User Guide*.
 - c. For **Subnet 1** and **Subnet 2**, enter the CIDR range for each subnet.
7. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
8. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
9. Choose **Create**.

Note

It can take up to 15 minutes for your Windows container instances to register with your cluster.

Step 2: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple webpage on port 8080 of a container instance with the `mcr.microsoft.com/windows/servercore/iis` container image.

To register the sample task definition with the AWS Management Console

1. Open the Amazon ECS classic 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 compatibilities** page, choose **Fargate**, **Next step**.
5. Scroll to the bottom of the page and choose **Configure via JSON**.
6. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

Use one of the following for `operatingSystemFamily`:

- `WINDOWS_SERVER_2019_FULL`
- `WINDOWS_SERVER_2019_CORE`
- `WINDOWS_SERVER_2022_FULL`
- `WINDOWS_SERVER_2022_CORE`

```
{
```

```
    "containerDefinitions": [
```

```
{  
    "command": [  
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value  
        '<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>'; C:\\ServiceMonitor.exe  
        w3svc"  
    ],  
    "entryPoint": [  
        "powershell",  
        "-Command"  
    ],  
    "essential": true,  
    "cpu": 2048,  
    "memory": 4096,  
    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-  
    ltsc2019",  
    "name": "sample_windows_app",  
    "portMappings": [  
        {  
            "hostPort": 80,  
            "containerPort": 80,  
            "protocol": "tcp"  
        }  
    ]  
},  
    "memory": "4096",  
    "cpu": "2048",  
    "networkMode": "awsvpc",  
    "family": "windows-simple-iis-2019-core",  
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",  
    "runtimePlatform": {  
        "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"  
    },  
    "requiresCompatibilities": [  
        "FARGATE"  
    ]  
}  
}
```

7. Verify your information and choose **Create**.

To register the sample task definition with the AWS CLI

1. Create a file called `windows_fargate_sample_app.json`.
2. Open the file with your favorite text editor and add the sample JSON above to the file and save it.
3. Using the AWS CLI, run the following command to register the task definition with Amazon ECS.

Note

Make sure that your AWS CLI is configured to use the same region that your Windows cluster exists in, or add the `--region your_cluster_region` option to your command.

```
aws ecs register-task-definition --cli-input-json  
file:///windows_fargate_sample_app.json
```

Step 3: Create a service with your task definition

After you have registered your task definition, you can place tasks in your cluster with it. The following procedure creates a service with your task definition and places one task on your cluster.

To create a service from your task definition with the console

1. On the **Task Definition: windows_fargate_sample_app** registration confirmation page, choose **Actions, Create Service**.
2. On the **Create Service** page, enter the following information and then choose **Create service**.
 - **Launch type:** Fargate
 - **Platform operating system:** WINDOWS_SERVER_2019_FULL, WINDOWS_SERVER_2019_CORE, WINDOWS_SERVER_2022_FULL, or WINDOWS_SERVER_2022_CORE
 - **Cluster:** windows
 - **Service name:** windows_fargate_sample_app
 - **Service type:** REPLICA
 - **Number of tasks:** 1
 - **Deployment type:** Rolling update

To create a service from your task definition with the AWS CLI

- Using the AWS CLI, run the following command to create your service.

```
aws ecs create-service --cluster windows --task-definition windows-simple-iis --  
desired-count 1 --service-name windows_fargate_sample_app
```

Step 4: View your service

After your service has launched a task into your cluster, you can view the service and open the IIS test page in a browser to verify that the container is running.

Note

It can take up to 15 minutes for your container instance to download and extract the Windows container base layers.

To view your service

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, choose the **windows** cluster.
3. In the **Services** tab, choose the **windows_fargate_sample_app** service.
4. On the **Service: windows_fargate_sample_app** page, choose the task ID for the task in your service.
5. On the **Task** page, expand the **windows_fargate** container to view its information.
6. In the **Network bindings** of the container, you should see an **External Link** IP address and port combination link. Choose that link to open the IIS test page in your browser.

Amazon ECS Sample App

Congratulations!

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

Step 5: Clean Up

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

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

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, select the cluster to delete.
4. Choose **Delete Cluster**. At the confirmation prompt, enter **delete me** and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Getting started with the classic console using Amazon EC2

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Amazon Elastic Container Service (Amazon ECS) is a fast and highly scalable container management service that makes it easy to launch and manage your containers. For a broad overview on Amazon ECS, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS using the EC2 launch type by registering a task definition, creating a cluster, and creating a service in the classic console.

Complete the following steps to get started with Amazon ECS using the EC2 launch type.

Prerequisites

Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or the [Amazon ECS first-run wizard permissions \(p. 601\)](#) IAM policy example.

Step 1: Register a task definition

A task definition is like a blueprint for your application. Each time that 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. For more information about task definitions, see [Amazon ECS task definitions \(p. 83\)](#).

The following steps walk you through creating a task definition that will deploy a simple web application.

To register a task definition

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

2. From the navigation bar, select the Region you want to use.
3. In the navigation pane, choose **Task Definitions**, **Create new Task Definition**.
4. On the **Select launch type compatibility** page, select **EC2** and choose **Next step**.
5. On the **Configure task and container definitions** page, scroll down and choose **Configure via JSON**.
6. Copy and paste the following example task definition into the box and then choose **Save**.

```
{  
    "containerDefinitions": [  
        {  
            "entryPoint": [  
                "sh",  
                "-c"  
            ],  
            "portMappings": [  
                {  
                    "hostPort": 80,  
                    "protocol": "tcp",  
                    "containerPort": 80  
                }  
            ],  
            "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\""  
            ],  
            "cpu": 10,  
            "memory": 300,  
            "image": "httpd:2.4",  
            "name": "simple-app"  
        }  
    ],  
    "family": "console-sample-app-static"  
}
```

7. Choose **Create**.

Step 2: Create a cluster

An Amazon ECS cluster is a logical grouping of tasks, services, and container instances. When creating a cluster using the classic console, Amazon ECS creates a AWS CloudFormation stack that takes care of the Amazon EC2 instance creation, networking and IAM configuration for you. For more information about clusters, see [Amazon ECS clusters \(p. 220\)](#).

The following steps walk you through creating a cluster with one Amazon EC2 instance registered to it which will enable us to run a task on it. If a specific field is not mentioned, leave the default value the console uses.

To create a cluster

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. On the **Select cluster template** page, choose **EC2 Linux + Networking**.
6. For **Cluster name**, choose a name for your cluster.

7. In the **Instance configuration** section, do the following:
 - a. For **EC2 instance type**, choose either the **t2.micro** or **t3.micro** instance type to use for the container instance. Instance types with more CPU and memory resources can handle more tasks, but that is unnecessary for this getting started guide. For more information about the different instance types, see [Amazon EC2 Instances](#).
 - b. For **Number of instances**, type **1**. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see [Amazon EC2 Pricing](#).
 - c. For **EC2 Ami Id**, use the default value which is the Amazon Linux 2 Amazon ECS-optimized AMI. For more information about the Amazon ECS-optimized AMI, see [Amazon ECS-optimized AMI \(p. 255\)](#).
8. In the **Networking** section, for **VPC** choose either **Create a new VPC** to have Amazon ECS create a new VPC for the cluster to use, or choose an existing VPC to use. For more information, see [the section called "Create a virtual private cloud" \(p. 12\)](#).
9. In the **Container instance IAM role** section, choose **Create new role** to have Amazon ECS create a new IAM role for your container instances, or choose an existing Amazon ECS container instance (`ecsInstanceRole`) role that you have already created. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).
10. Choose **Create**.

Step 3: Create a Service

An Amazon ECS service helps you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. 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 in order to maintain the desired number of tasks in the service. For more information on services, see [Amazon ECS services \(p. 453\)](#).

To create a service

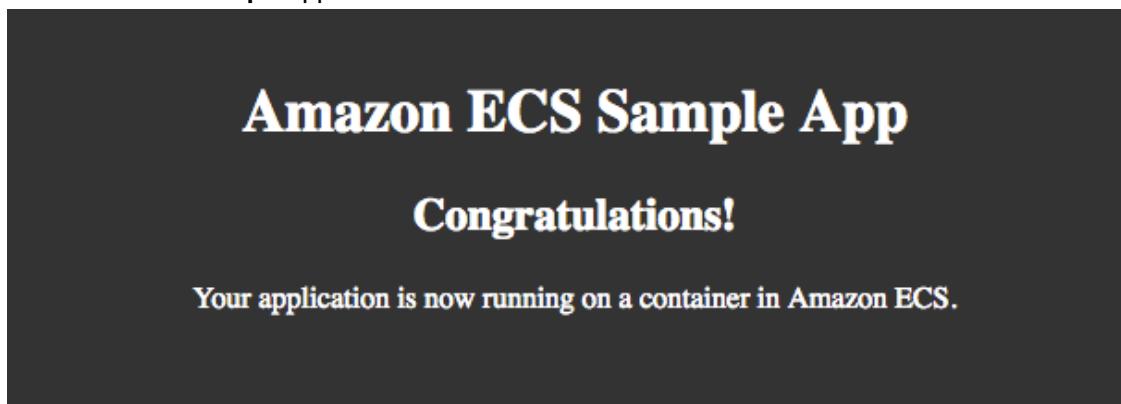
1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. Select the cluster you created in the previous step.
5. On the **Services** tab, choose **Create**.
6. In the **Configure service** section, do the following:
 - a. For **Launch type**, select **EC2**
 - b. For **Task definition**, select the **console-sample-app-static** task definition you created in step 1.
 - c. For **Cluster**, select the cluster you created in step 2.
 - d. For **Service name**, select a name for your service.
 - e. For **Number of tasks**, enter **1**.
7. Use the default values for the rest of the fields and choose **Next step**.
8. In the **Configure network** section, leave the default values and choose **Next step**.
9. In the **Set Auto Scaling** section, leave the default value and choose **Next step**.
10. Review the options and choose **Create service**.
11. Choose **View service** to review your service.

Step 4: View your Service

The service is a web-based application so you can view its containers with a web browser.

To view the service details

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. Select the cluster you created step 2.
5. On the **Services** tab, choose the service you created in step 3.
6. On the **Service: *service-name*** page, choose the **Tasks** tab.
7. Confirm that the task is in a **RUNNING** state. If it is, select the task to view the task details. If it is not in a **RUNNING** status, refresh the service details screen until it is.
8. In the **Containers** section, expand the container details. In the **Network bindings** section, for **External Link** you will see the **IPv4 Public IP** address to use to access the web application.
9. Enter the **IPv4 Public IP** address in your web browser and you should see a webpage that displays the **Amazon ECS sample** application.



Step 5: Clean Up

When you are finished using an Amazon ECS cluster, you can clean up the resources associated with it to avoid incurring charges for resources that you are not using.

The Amazon ECS resources created in this getting started guide, such as the cluster and service can be cleaned up using the Amazon ECS console.

To clean up the resources

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. Select the cluster you created step 2.
4. On the **Services** tab, select the service you created in step 3 and choose **Delete**. At the confirmation prompt, enter **delete me** and then choose **Delete**.
5. On the cluster details page, choose **Delete cluster**. At the confirmation prompt, enter **delete me** and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including the VPC and Amazon EC2 instances.

Getting started with Windows containers using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

This tutorial walks you through getting Windows containers running on either Amazon ECS with the Amazon ECS-optimized Windows Server AMI in the AWS Management Console.

You create a cluster for your Windows container instances, launch one or more container instances into your cluster, register a task definition that uses a Windows container image, create a service that uses that task definition, and then view the sample webpage that the container runs. For more information, see [Amazon EC2 Windows containers \(p. 363\)](#).

Topics

- [Step 1: Create a Windows cluster \(p. 875\)](#)
- [Step 2: Register a Windows task definition \(p. 728\)](#)
- [Step 3: Create a service with your task definition \(p. 878\)](#)
- [Step 4: View your service \(p. 878\)](#)

Step 1: Create a Windows cluster

You can create a new cluster for your Windows containers. Amazon EC2 instances using the Linux Amazon ECS-optimized AMIs cannot run Windows containers, and vice versa, so proper task placement is best accomplished by running Windows and Linux container instances in separate clusters. In this tutorial, you create a cluster called windows and register one or more Amazon EC2 instances into the cluster for your Windows containers.

To create a cluster with the AWS Management Console

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create Cluster**.
4. Choose **EC2 Windows + Networking** and choose **Next step**.
5. For **Cluster name** enter a name for your cluster (in this example, windows is the name of the cluster). Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. In the **Instance configuration** section, complete the following steps.
 - a. For **Provisioning model**, choose one of the following instance types:
 - **On-Demand Instance**– With On-Demand Instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments.
 - **Spot**– Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. For more information, see [Spot Instances](#).

Note

Spot Instances are subject to possible interruptions. We recommend that you avoid Spot Instances for applications that can't be interrupted. For more information, see [Spot Instance Interruptions](#).

- b. For Spot Instances, do the following; otherwise, skip to the next step.
 - i. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see [Spot Fleet Allocation Strategy](#).
 - ii. For **Maximum bid price (per instance/hour)**, specify a bid price. If your bid price is lower than the Spot price for the instance types that you selected, your Spot Instances are not launched.
 - c. For **EC2 instance type** page, choose the hardware configuration of your instance. The instance type that you select determines the resources available for your tasks to run on.
 - d. For **Number of instances**, choose the number of Amazon EC2 instances to launch into your cluster.
 - e. For **EC2 AMI Id**, choose the Amazon ECS-optimized AMI to use for your container instances. The available AMIs will be determined by the Region and instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).
 - f. For **EBS storage (GiB)**, choose the size of the Amazon EBS volume to use for data storage on your container instances. You can increase the size of the data volume to allow for greater image and container storage.
 - g. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for RDP access. If you do not specify a key pair, you cannot connect to your container instances with RDP. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.
7. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.
 - a. For **VPC**, create a new VPC, or select an existing VPC.
 - b. (Optional) If you chose 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*.
 - c. For **Subnets**, select the subnets to use for your VPC. If you chose to create a new VPC, you can keep the default settings or you can modify them to meet your needs. If you chose to use an existing VPC, select one or more subnets in that VPC to use for your cluster.
 - d. For **Security group**, select the security group to attach to the container instances in your cluster. If you choose to create a new security group, you can specify a CIDR block to allow inbound traffic from. The default port 0.0.0.0/0 is open to the internet. You can also select a single port or a range of contiguous ports to open on the container instance. For more complicated security group rules, you can choose an existing security group that you have already created.

Note

You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see [Amazon EC2 security groups for Windows instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

- e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the **ecsInstanceRole** that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

Important

The IAM role you use must have the **AmazonEC2ContainerServiceforEC2Role** managed policy attached to it, otherwise you will receive an error during cluster creation. If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role **ecsSpotFleetRole** is created.

8. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
9. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
10. Choose **Create**.

Note

It can take up to 15 minutes for your Windows container instances to register with your cluster.

Step 2: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple webpage on port 8080 of a container instance with the `microsoft/iis` container image.

To register the sample task definition with the AWS Management Console

1. Open the Amazon ECS classic 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 compatibilities** page, choose **EC2, Next step**.
5. Scroll to the bottom of the page and choose **Configure via JSON**.
6. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 512,
      "entryPoint": ["powershell", "-Command"],
      "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value '<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>' -Force ; C:\\ServiceMonitor.exe w3svc'],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80,
          "hostPort": 8080
        }
      ],
      "memory": 768,
      "essential": true
    }
  ]
}
```

7. Verify your information and choose **Create**.

To register the sample task definition with the AWS CLI

1. Create a file called `windows-simple-iis.json`.

2. Open the file with your favorite text editor and add the sample JSON above to the file and save it.
3. Using the AWS CLI, run the following command to register the task definition with Amazon ECS.

Note

Make sure that your AWS CLI is configured to use the same region that your Windows cluster exists in, or add the `--region` `your_cluster_region` option to your command.

```
aws ecs register-task-definition --cli-input-json file://windows-simple-iis.json
```

Step 3: Create a service with your task definition

After you have registered your task definition, you can place tasks in your cluster with it. The following procedure creates a service with your task definition and places one task on your cluster.

To create a service from your task definition with the console

1. On the **Task Definition: windows-simple-iis** registration confirmation page, choose **Actions, Create Service**.
2. On the **Create Service** page, enter the following information and then choose **Create service**.
 - **Launch type:** EC2
 - **Cluster:** windows
 - **Service name:** windows-simple-iis
 - **Service type:** REPLICA
 - **Number of tasks:** 1
 - **Deployment type:** Rolling update

To create a service from your task definition with the AWS CLI

- Using the AWS CLI, run the following command to create your service.

```
aws ecs create-service --cluster windows --task-definition windows-simple-iis --desired-count 1 --service-name windows-simple-iis
```

Step 4: View your service

After your service has launched a task into your cluster, you can view the service and open the IIS test page in a browser to verify that the container is running.

Note

It can take up to 15 minutes for your container instance to download and extract the Windows container base layers.

To view your service

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, choose the **windows** cluster.
3. In the **Services** tab, choose the **windows-simple-iis** service.
4. On the **Service: windows-simple-iis** page, choose the task ID for the task in your service.
5. On the **Task** page, expand the **iis** container to view its information.
6. In the **Network bindings** of the container, you should see an **External Link** IP address and port combination link. Choose that link to open the IIS test page in your browser.



Cluster management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can use the classic console to configure run your Amazon ECS cluster resources.

Topics

- [Creating a cluster using the classic console \(p. 879\)](#)
- [Creating an Auto Scaling group capacity provider using the classic console \(p. 882\)](#)
- [Updating an Auto Scaling group capacity provider using the classic console \(p. 883\)](#)
- [Creating a cluster with an Auto Scaling group capacity provider \(p. 885\)](#)
- [Deleting an Auto Scaling group capacity provider using the classic console \(p. 885\)](#)
- [Deleting a cluster using the classic console \(p. 886\)](#)

Creating a cluster using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can create an Amazon ECS cluster using the classic AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 10\)](#). You can register Amazon EC2 instances during cluster creation or register additional instances with the cluster after creating it.

The console cluster creation wizard provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack. It also lets you customize several common cluster configuration options. However, the wizard does not allow you to customize every resource option. For example, you can't use the wizard to customize the container instance AMI ID. If your

requirements extend beyond what is supported in this wizard, consider using our reference architecture at <https://github.com/awslabs/ecs-refarch-cloudformation>.

If you add or modify the underlying cluster resources directly after they are created by the wizard you may receive an error when attempting to delete the cluster. AWS CloudFormation refers to this as *stack drift*. For more information on detecting drift on an existing AWS CloudFormation stack, see [Detect Drift on an Entire CloudFormation Stack](#) in the *AWS CloudFormation User Guide*.

To create a cluster (AWS Management Console)

1. Open the Amazon ECS classic 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 one of the following options and then choose **Next Step**:
 - **Networking only**– This cluster template creates an empty cluster. Optionally, you can create a new VPC to use. This cluster template is typically used for workloads hosted on either AWS Fargate or external instances (ECS Anywhere). The FARGATE and FARGATE_SPOT capacity providers will be automatically associated with the cluster. For more information, see [AWS Fargate capacity providers \(p. 222\)](#).
 - **EC2 Linux + Networking**– This cluster template is used to create a cluster of Amazon EC2 instances to run Linux-based containers on. An Auto Scaling group is created for the Amazon EC2 instances.
 - **EC2 Windows + Networking** – This cluster template is used to create a cluster of Amazon EC2 instances to run Windows-based containers on. An Auto Scaling group is created for the Amazon EC2 instances. For more information, see [Amazon EC2 Windows containers \(p. 363\)](#).

Using the Networking only template

If you chose the **Networking only** cluster template, complete the following steps. Otherwise, skip to [Using the EC2 Linux + Networking or EC2 Windows + Networking template \(p. 880\)](#).

Using the Networking only cluster template

1. On the **Configure cluster** page, enter a **Cluster name**. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. In the **Networking** section, configure the VPC for your cluster. You can keep the default settings, or you can modify these settings with the following steps.
 - 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.
3. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
4. In the **CloudWatch Container Insights** section, choose whether to turn on Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
5. Choose **Create**.

Using the EC2 Linux + Networking or EC2 Windows + Networking template

If you chose the **EC2 Linux + Networking** or **EC2 Windows + Networking** templates, complete the following steps.

Using the EC2 Linux + Networking or EC2 Windows + Networking cluster template

1. For **Cluster name**, enter a name for your cluster. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. (Optional) To create a cluster with no resources, choose **Create an empty cluster**, **Create**.
3. For **Provisioning model**, choose one of the following instance types:
 - **On-Demand Instance**– With On-Demand Instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments.
 - **Spot**– Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. For more information, see [Spot Instances](#).

Note

Spot Instances are subject to possible interruptions. We recommend that you avoid Spot Instances for applications that can't be interrupted. For more information, see [Spot Instance Interruptions](#).

4. For Spot Instances, do the following; otherwise, skip to the next step.
 - a. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see [Spot Fleet Allocation Strategy](#).
 - b. For **Maximum bid price (per instance/hour)**, specify a bid price. If your bid price is lower than the Spot price for the instance types that you selected, your Spot Instances are not launched.
5. For **EC2 instance type**, choose the Amazon EC2 instance type for your container instances. The instance type that you select determines the EC2 AMI IDs and resources available for your tasks. For GPU workloads, choose an instance type from the P2 or P3 instance family. For more information, see [Working with GPUs on Amazon ECS \(p. 142\)](#).
6. For **Number of instances**, choose the number of EC2 instances to launch into your cluster. These instances are launched using the latest Amazon ECS-optimized Amazon Linux AMI required by the instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).
7. For **EC2 AMI Id**, choose the Amazon ECS-optimized AMI for your container instances. The available AMIs will be determined by the Region and EC2 instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 255\)](#).
8. For **EBS storage (GiB)**, choose the size of the Amazon EBS volume to use for data storage on your container instances. You can increase the size of the data volume to allow for greater image and container storage.
9. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for SSH access. If you do not specify a key pair, you cannot connect to your container instances with SSH. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.
10. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.
 - a. For **VPC**, create a new VPC, or select an existing VPC.
 - b. (Optional) If you chose 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*.
 - c. For **Subnets**, select the subnets to use for your VPC. If you chose to create a new VPC, you can keep the default settings or you can modify them to meet your needs. If you chose to use an existing VPC, select one or more subnets in that VPC to use for your cluster.
 - d. For **Security group**, select the security group to attach to the container instances in your cluster. If you choose to create a new security group, you can specify a CIDR block to allow inbound traffic from. The default port 0.0.0.0/0 is open to the internet. You can also select a single port or a range of contiguous ports to open on the container instance. For more complicated security group rules, you can choose an existing security group that you have already created.

Note

You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see [Amazon EC2 Security Groups for Linux Instances](#) in the [Amazon EC2 User Guide for Linux Instances](#).

- e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the **ecsInstanceRole** that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

Important

The IAM role you use must have the `AmazonEC2ContainerServiceforEC2Role` managed policy attached to it, otherwise you will receive an error during cluster creation. If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

- f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role `ecsSpotFleetRole` is created.
- g. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
- h. In the **CloudWatch Container Insights** section, choose whether to turn on Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 572\)](#).
- i. Choose **Create**.

Creating an Auto Scaling group capacity provider using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

A *capacity provider* is used in association with a cluster to determine the infrastructure that a task runs on. When creating a capacity provider, you specify the following details:

- An Auto Scaling group Amazon Resource Name (ARN)
- Whether or not to turn on managed scaling. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling policies. When managed scaling is turned off, you manage your Auto Scaling groups yourself.
- Whether or not to turn on managed termination protection. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled.

Use the following steps to create a new capacity provider for an existing Amazon ECS cluster.

To create an Auto Scaling group capacity provider (classic AWS Management Console)

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.

4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose **Capacity Providers**, and then choose **Create**.
6. For **Capacity provider name**, enter a capacity provider name.
7. For **Auto Scaling group**, select the Auto Scaling group to associate with the capacity provider. The Auto Scaling group must already be created.
8. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling policies. When managed scaling is turned off, you manage your Auto Scaling groups yourself.
9. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.
10. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
11. Choose **Create** to complete the capacity provider creation.

To create an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.
- [create-capacity-provider \(AWS CLI\)](#)

```
aws ecs create-capacity-provider \
  --name CapacityProviderName \
  --auto-scaling-group-provider
  autoScalingGroupArn="AutoScalingGroupARN",managedScaling=\{status='ENABLED|\
  DISABLED',targetCapacity=integer,minimumScalingStepSize=integer,maximumScalingStepSize=integer\},\
  DISABLED" \
  --region us-east-2
```

If you prefer to use a JSON input file with the `create-capacity-provider` command, use the following command to generate a CLI skeleton.

```
aws ecs create-capacity-provider --generate-cli-skeleton
```

Updating an Auto Scaling group capacity provider using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

A capacity provider can be updated to change its managed scaling and managed termination protection settings. Use the following steps to update an existing capacity provider.

To update an Auto Scaling group capacity provider (classic AWS Management Console)

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region the cluster the capacity provider is associated with exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose the **Capacity Providers** tab.
6. Select the capacity provider to update and choose **Update**.
7. On the **Update Capacity Provider** page, the following parameters can be updated.
 - a. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling policies. When managed scaling is turned off, you manage your Auto Scaling groups yourself.
 - b. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.
 - c. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
8. Choose **Update** to request capacity provider update.
9. To verify whether the capacity provider update was successful, check the **Update Status** column on the **Capacity Providers** tab.

To update an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.
- [update-capacity-provider \(AWS CLI\)](#)

```
aws ecs update-capacity-provider \
--name CapacityProviderName \
--auto-scaling-group-provider managedScaling=\{status='ENABLED' \
DISABLED',targetCapacity=integer,minimumScalingStepSize=integer,maximumScalingStepSize=integer\}, \
--region us-east-2
```

If you prefer to use a JSON input file with the `create-capacity-provider` command, use the following command to generate a CLI skeleton.

```
aws ecs update-capacity-provider --generate-cli-skeleton
```

Creating a cluster with an Auto Scaling group capacity provider

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

When a new Amazon ECS cluster is created, you can specify one or more capacity providers to associate with the cluster. The associated capacity providers determine the infrastructure to run your tasks on.

For AWS Management Console steps, see [the section called “Creating a cluster for the Fargate launch type using the console” \(p. 236\)](#).

Deleting an Auto Scaling group capacity provider using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

If you are finished using an Auto Scaling group capacity provider, you can delete it. Once deleted, the Auto Scaling group capacity provider will transition to the INACTIVE state. Capacity providers with an INACTIVE status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on INACTIVE capacity providers persisting.

Prior to an Auto Scaling group capacity provider being deleted, the capacity provider must be removed from the capacity provider strategy from all services. The `UpdateService` API or the update service workflow in the AWS Management Console can be used to remove a capacity provider from a service's capacity provider strategy. The force new deployment option can be used to ensure that any tasks using the Amazon EC2 instance capacity provided by the capacity provider are transitioned to use the capacity from the remaining capacity providers.

There are other prerequisites that must be performed to delete a capacity provider but they are specific to the tool used and are mentioned in the following steps.

Use the following steps to delete an Auto Scaling group capacity provider.

To delete an Auto Scaling group capacity provider (classic AWS Management Console)

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

When deleting a capacity provider using the classic AWS Management Console, the console goes through two steps. The capacity provider is first disassociated from the cluster completely and then it is deleted.

In rare cases, the capacity provider may be successfully disassociated from the cluster but is unable to be deleted. In those cases, you must use either the Amazon ECS API or the AWS CLI to view the status of the capacity provider and delete it.

Note

Only capacity providers that are currently associated with a cluster are visible in the AWS Management Console. To delete a capacity provider that is not associated with a cluster, you must use the Amazon ECS API, SDK, or AWS CLI.

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose the **Capacity Providers** tab.
6. Select the capacity provider you want to delete and then choose **Delete**.

Deleting a cluster using the classic console

The new experience is now the default in the Amazon ECS console. For more information, see [the section called "Deleting a cluster using the console" \(p. 240\)](#).

If you are finished using a cluster, you can delete it. After you delete the cluster, it transitions to the INACTIVE state. Clusters with an INACTIVE status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on INACTIVE clusters persisting.

When you delete a cluster in the Amazon ECS console, the associated resources that are deleted will vary depending on how the cluster was created. This condition is discussed in step 5 of the following procedure.

If your cluster was created with the AWS Management Console then the AWS CloudFormation stack that was created for your cluster is also deleted when you delete your cluster. If you have added or modified the underlying cluster resources you may receive an error when attempting to delete the cluster. AWS CloudFormation refers to this as *stack drift*. For more information on detecting drift on an existing AWS CloudFormation stack, see [Detect drift on an entire AWS CloudFormation stack](#) in the [AWS CloudFormation User Guide](#).

To delete a cluster using the classic console

1. Open the Amazon ECS classic 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.

Note

If your cluster has registered container instances, you must deregister or terminate them. For more information, see [Deregister an Amazon EC2 backed container instance \(p. 335\)](#).

5. In the upper-right of the page, choose **Delete Cluster**. You see one of two confirmation prompts:

- **Deleting the cluster also deletes the AWS CloudFormation stack**

EC2ContainerService-*cluster_name* – Deleting this cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

- **Deleting the cluster does not affect AWS CloudFormation resources** – Deleting this cluster does not clean up any resources that are associated with the cluster, including Auto Scaling groups, VPCs, or load balancers. Also, any container instances that are registered with this cluster must be deregistered or terminated before you can delete the cluster. For more information, see [Deregister an Amazon EC2 backed container instance \(p. 335\)](#). You can visit the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation/> to update or delete any of these resources.
6. In the confirmation box, enter **delete me**.

Task definition management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can use the classic console to configure run your Amazon ECS task definitions.

Topics

- [Creating a task definition using the classic console \(p. 887\)](#)
- [Updating a task definition using the classic console \(p. 894\)](#)
- [Deregistering a task definition revision \(p. 895\)](#)

Creating a task definition using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Important

Amazon ECS has provided a new console experience for creating task definitions. For more information, see [Creating a task definition using the console \(p. 125\)](#).

You must use the new console to create a task definition for Windows containers on Fargate with the WINDOWS_SERVER_2022_FULL, or WINDOWS_SERVER_2022_CORE operating system.

Before running Docker containers on Amazon ECS, you must first create a task definition. When you create a task definition, you can use it to define multiple containers and data volumes. For more information about the available parameters for task definitions, see [Task definition parameters \(p. 799\)](#).

To create a new task definition (Classic Amazon ECS console)

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **task definitions**, **Create new task definition**.
3. On the **Select compatibilities** page, select the launch type that your task is to use and choose **Next step**.
4. Follow the steps under one of the following tabs, according to the launch type that you chose.

Fargate launch type

Using the Fargate launch type compatibility template

If you chose **Fargate**, complete the following steps:

1. (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**.

2. For **Task Definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS API operations on your behalf. For more information, see [Task IAM role \(p. 631\)](#).

Note

Only roles that have 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 \(p. 633\)](#).

4. For **Operating system family**, choose the container operating system.
5. 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, see [Amazon ECS task execution IAM role \(p. 626\)](#).
6. 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	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) Note This option requires Linux platform 1.4.0 or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU)	Between 32 GB and 120 GB in 8 GB increments	Linux

CPU value	Memory value	Operating systems supported for AWS Fargate
<p>Note This option requires Linux platform 1.4.0 or later.</p>		

7. 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 \(p. 799\)](#).
 - c. Choose **Add** to add your container to the task definition.
8. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration, choose **Enable App Mesh integration** and then do the following:
 - a. For **Mesh name**, choose the existing App Mesh service mesh to use. If you don't see any meshes listed, then you need to create one first. For more information, see [Service meshes](#) in the *AWS App Mesh User Guide*.

Note
 This option is not available for Windows containers on Fargate.
 - b. For **App Mesh endpoints**, select one of the following options.
 - **Virtual node** – Enter or select the following information.
 - For **Application container name**, choose the container name to use for the App Mesh integration. This container must already be defined within the task definition.
 - For **Virtual node name**, choose the existing App Mesh virtual node to use. If you don't see any virtual nodes listed, then you need to create one first. For more information, see [Virtual nodes](#) in the *AWS App Mesh User Guide*.
 - For **Virtual node port** – Pre-populated with the listener port set on the virtual node in App Mesh.
 - **Virtual gateway** – Enter or select the following information.
 - For **Virtual gateway name**, choose the existing App Mesh virtual gateway to use. If you don't see any virtual gateways listed, then you need to create one first. For more information, see [Virtual gateways](#) in the *AWS App Mesh User Guide*.
 - For **Virtual gateway port** – Pre-populated with the listener port set on the virtual gateway in App Mesh.
 - c. For **Envoy image**, enter **840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod** for all regions except me-south-1 and ap-east-1. You can replace **us-west-2** with any Region except me-south-1 and ap-east-1. If your application is in one of these regions, then you also need to replace **840364872350** with the appropriate value for your Region. For more information, see [Envoy image](#) in the *AWS App Mesh User Guide*.
 - d. Choose **Apply** and then choose **Confirm**. This will add an Envoy proxy container to the task definition, as well as the settings to support it. If you selected **Virtual node**, it will also auto-populate the App Mesh **Proxy Configuration** settings for the next step. If you selected **Virtual gateway**, then the **Proxy Configuration** is disabled, because it's not used for a virtual gateway.
9. (Optional) If you selected **Virtual node** in **Service Integration**, then for **Proxy Configuration**, verify all of the pre-populated values. For more information about these fields, see the JSON tab in [Update services](#).

10. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:
 - a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the fluentbit log router type, the **Image** field pre-populates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 170\)](#).
 - c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.
11. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see [Using data volumes in tasks \(p. 100\)](#).
 - For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
12. 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](#).
13. Choose **Create**.

EC2 launch type

Using the EC2 launch type compatibility template

If you chose **EC2**, complete the following steps:

1. (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**.

2. For **task definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, 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 [Task IAM role \(p. 631\)](#).

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role. For more information, see [Amazon ECS container instance IAM role \(p. 638\)](#).

Note

Only roles that have the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For instructions on how to create an IAM role for your tasks, see [Creating an IAM role and policy for your tasks \(p. 633\)](#).

4. (Optional) For **Network Mode**, choose the Docker network mode to use for the containers in your task. The available network modes correspond to those that are described in [Network settings](#) in the Docker run reference. If you select **Enable App Mesh integration** in a following step, then you must select `awsvpc`.

The default Docker network mode is `bridge`. If the network mode is set to `none`, you can't specify port mappings in your container definitions. Moreover, the task's containers don't

have external connectivity. If the network mode is awsvpc, the task is provided with an elastic network interface. The host and awsvpc network modes offer the highest networking performance for containers. This is because they use the Amazon EC2 network stack, instead of the virtualized network stack that's provided by the bridge mode. However, exposed container ports are mapped directly to the corresponding host port. Therefore, if port mappings are used, you can't use dynamic host port mappings or run multiple instantiations of the same task on a single container instance.

5. (Optional) For **Task execution role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role. This role is specified earlier as the **Task Role**. There's no need to specify a task execution role. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).

6. (Optional) For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. The Task CPU (vCPU) values that are supported are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

Note

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

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

- a. Choose **Add container**.

- b. Enter each of the 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 \(p. 799\)](#).

- c. Choose **Add** to add your container to the task definition.

8. (Optional) For **Constraint**, you define how tasks that are created from this task definition are placed in your cluster. For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see [Amazon ECS task placement constraints \(p. 437\)](#).

9. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration, choose **Enable App Mesh integration** and then do the following:

- a. For **Mesh name**, choose the existing App Mesh service mesh to use. If you don't see any meshes listed, then you need to create one first. For more information, see [Service meshes](#) in the *AWS App Mesh User Guide*.

Note

This option is not available for Windows containers on Fargate.

- b. For **App Mesh endpoints**, select one of the following options.

- **Virtual node** – Enter or select the following information.

- For **Application container name**, choose the container name to use for the App Mesh integration. This container must already be defined within the task definition.

- For **Virtual node name**, choose the existing App Mesh virtual node to use. If you don't see any virtual nodes listed, then you need to create one first. For more information, see [Virtual nodes](#) in the *AWS App Mesh User Guide*.

- For **Virtual node port** – Pre-populated with the listener port set on the virtual node in App Mesh.

- **Virtual gateway** – Enter or select the following information.

- For **Virtual gateway name**, choose the existing App Mesh virtual gateway to use. If you don't see any virtual gateways listed, then you need to create one first. For more information, see [Virtual gateways](#) in the *AWS App Mesh User Guide*.

- For **Virtual gateway port** – Pre-populated with the listener port set on the virtual gateway in App Mesh.
- c. For **Envoy image**, enter `840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod` for all regions except me-south-1 and ap-east-1. You can replace `us-west-2` with any Region except me-south-1 and ap-east-1. If your application is in one of these regions, then you also need to replace `840364872350` with the appropriate value for your Region. For more information, see [Envoy image](#) in the [AWS App Mesh User Guide](#).
 - d. Choose **Apply** and then choose **Confirm**. This will add an Envoy proxy container to the task definition, as well as the settings to support it. If you selected **Virtual node**, it will also auto-populate the App Mesh **Proxy Configuration** settings for the next step. If you selected **Virtual gateway**, then the **Proxy Configuration** is disabled, because it's not used for a virtual gateway.
10. (Optional) If you selected **Virtual node** in **Service Integration**, then for **Proxy Configuration**, verify all of the pre-populated values. For more information about these fields, see the JSON tab in [Update services](#).
 11. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:
 - a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the fluentbit log router type, the **Image** field pre-populates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 170\)](#).
 - c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.
 12. (Optional) To define data volumes for your task, choose **Add volume**. You can create either a bind mount or Docker volume. For more information, see [Using data volumes in tasks \(p. 100\)](#).
 - a. For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - b. (Optional) To create a bind mount volume, for **Source path**, enter the path on the host container instance to present to the container. If you leave this field empty, the Docker daemon assigns a host path for you. If you specify a source path, the data volume persists at the specified location on the host container instance until you delete it manually. If the source path doesn't exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
 - c. To create a Docker volume, select **Specify a volume driver**.
 - i. For **Driver**, choose the Docker volume driver to use. The driver value must match the driver name provided by Docker. Use `docker plugin ls` on your container instance to retrieve the driver name.
 - ii. For **Scope**, choose the option that determines the lifecycle of the Docker volume. Docker volumes that are scoped to a task are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as shared persist after the task stops.
 - iii. Select **Enable auto-provisioning** to have the Docker volume created if it doesn't already exist. This option is only available for volumes that specify the shared scope.
 - iv. For **Driver options**, specify the driver-specific key values to use.
 - v. For **Volume labels**, specify the custom metadata to add to your Docker volume.
 13. 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](#).
 14. Choose **Create**.

External instance launch type

Using the external instance launch type

If you chose **External**, complete the following steps:

1. (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 file 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**.

2. For **task definition Name**, enter a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, 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 [Task IAM role \(p. 631\)](#) and [the section called "IAM permissions" \(p. 340\)](#).
4. (Optional) For **Network Mode**, choose the Docker network mode to use for the containers in your task. The available network modes correspond to those that are described in [Network settings](#) in the Docker run reference.

The default Docker network mode is bridge. If the network mode is set to none, you can't specify port mappings in your container definitions, and the task's containers don't have external connectivity. If the network mode is awsvpc, the task is allocated an elastic network interface. The host and awsvpc network modes offer the highest networking performance for containers. This is because they use the Amazon EC2 network stack, instead of the virtualized network stack that's provided by the bridge mode. However, exposed container ports are mapped directly to the corresponding host port. Therefore, you can't use dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

5. (Optional) For **Task execution role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.
6. (Optional) For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. Supported Task CPU (vCPU) values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

Note

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

7. For each container in your task definition, complete the following steps.
 - a. Choose **Add container**.
 - b. Enter each of the 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 \(p. 799\)](#).
 - c. Choose **Add** to add your container to the task definition.
8. (Optional) For **Constraint**, you define how tasks that are created from this task definition are placed in your cluster. For more information, see [Amazon ECS task placement constraints \(p. 437\)](#).
9. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:

- a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the fluentbit log router type, the **Image** field pre-populates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 170\)](#).
 - c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.
10. (Optional) To define data volumes for your task, choose **Add volume**. You can create either a bind mount or Docker volume. For more information, see [Using data volumes in tasks \(p. 100\)](#).
 - a. For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - b. (Optional) To create a bind mount volume, for **Source path**, enter the path on the host container instance to present to the container. If you leave this field empty, the Docker daemon assigns a host path for you. If you specify a source path, the data volume persists at the specified location on the host container instance until you delete it manually. If the source path doesn't exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
 - c. To create a Docker volume, select **Specify a volume driver**.
 - i. For **Driver**, choose the Docker volume driver to use. The driver value must match the driver name provided by Docker. Use `docker plugin ls` on your container instance to retrieve the driver name.
 - ii. For **Scope**, choose the option that determines the lifecycle of the Docker volume. Docker volumes that are scoped to a task are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as `shared` persist after the task stops.
 - iii. Select **Enable auto-provisioning** to have the Docker volume created if it does not already exist. This option is only available for volumes that specify the `shared` scope.
 - iv. For **Driver options**, specify the driver-specific key values to use.
 - v. For **Volume labels**, specify the custom metadata to add to your Docker volume.
 11. 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](#).
 12. Choose **Create**.

To create a new task definition (AWS CLI)

- Use the `register-task-definition` command. For more information, see [register-task-definition](#) in the *AWS Command Line Interface Reference*.

Updating a task definition using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

A *task definition revision* is a copy of the current task definition with the new parameter values replacing the existing ones. All parameters that you do not modify are in the new revision.

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 classic 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 using the console \(p. 465\)](#).

Deregistering a task definition revision

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

If you decide that you no longer need a specific task definition revision in Amazon ECS, you can deregister the task definition revision 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 revision, it's immediately marked as INACTIVE. Existing tasks and services that reference an INACTIVE task definition revision continue to run without disruption. Existing services that reference an INACTIVE task definition revision can still scale up or down by modifying the service's desired count.

You can't use an INACTIVE task definition revision to run new tasks or create new services. You also can't update an existing service to reference an INACTIVE task definition revision. This is despite that there might be up to a 10-minute window following deregistration where these restrictions have not already taken effect.

Note

You can't deregister a task definition family at one time. You can only deregister individual revisions or multiple revisions within the family. When you deregister all revisions, the task definition family is moved to the INACTIVE list. Adding a new revision of an INACTIVE task definition moves the task definition family back to the ACTIVE list.

At this time, INACTIVE task definition revisions remain discoverable in your account indefinitely. However, this behavior is subject to change in the future. Therefore, don't rely on INACTIVE task definition revisions persisting beyond the lifecycle of any associated tasks and services.

To deregister a new task definition (Classic Amazon ECS console)

1. Open the Amazon ECS classic 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 family 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 that you want to deregister.
6. Choose **Actions, Deregister**.
7. Verify the information in the **Deregister task definition** window, and then choose **Deregister** to finish.
8. (Optional) To deregister the task definition family, repeat the above steps for each ACTIVE revision.

Task management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can use the classic console to configure and run your Amazon ECS tasks.

Topics

- [Run a standalone task in the classic Amazon ECS console \(p. 896\)](#)

Run a standalone task in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

We recommend that you deploy your application as a standalone task in some situations. For example, suppose that you're developing an application but you're not ready to deploy it with the service scheduler. If your application is a one-time or periodic batch job, it doesn't make sense to keep running or restart when it finishes.

To deploy your application to run continually or to place it behind a load balancer, create an Amazon ECS service. For more information, see [Amazon ECS services \(p. 453\)](#).

Classic console

To run a standalone task using the classic console

1. Open the Amazon ECS classic 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, select the box to the left of the task definition to run.

- To run an earlier revision of a task definition, select the task definition to view all active revisions. Last, select the revision to run.
3. Choose **Actions**, **Run Task**.
 4. On the **Run Task** page, complete the following steps.
 - a. Choose either a capacity provider strategy or a launch type.
 - To use a **Capacity provider strategy** and choose **Switch to capacity provider strategy**. Then, choose whether your task uses the default capacity provider strategy that's defined for the cluster or a custom capacity provider strategy. A capacity provider must be associated with the cluster to be used in a custom capacity provider strategy. For more information, see [Amazon ECS capacity providers \(p. 220\)](#).
 - To use a **Launch type**, choose **Switch to launch type** and select either **EC2** or **EXTERNAL**. For more information about launch types, see [Amazon ECS launch types \(p. 85\)](#).
 - b. For **Cluster**, choose the cluster to use.
 - c. For **Number of tasks**, enter the number of tasks to launch with this task definition.
 - d. For **Task group**, enter the name of the task group.
 5. If your task definition uses the awsvpc network mode, complete these substeps. Otherwise, proceed to the next step.
 - a. For **Cluster VPC**, choose the VPC that your container instances reside in.
 - b. For **Subnets**, choose the available subnets for your task.

Important
Only private subnets are supported for the awsvpc network mode. Tasks don't receive public IP addresses. Therefore, a NAT gateway is required for outbound internet access, and inbound internet traffic is routed through a load balancer.
 6. (Optional) For **Task Placement**, you can specify how tasks are placed using task placement strategies and constraints. Choose from the following options:
 - **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
 - **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
 - **BinPack** - Distribute tasks based on the least available amount of CPU or memory.
 - **One Task Per Host** - Place, at most, one task from the service on each container instance.
 - **Custom** - Define your own task placement strategy.

For more information, see [Amazon ECS task placement \(p. 434\)](#).
 7. (Optional) To send command, environment variable, task IAM role, or task execution role overrides to one or more containers in your task definition, choose **Advanced Options** and complete the following steps:

Note
If you intend to use the parameter values from your task definition, you don't need to specify overrides. These fields are only used to override the values that are specified in the task definition.

 - a. For **Task Role Override**, choose an IAM role for this task to override the task IAM role that's specified in the task definition. For more information, see [Task IAM role \(p. 631\)](#).

Only roles with the `ecs-tasks.amazonaws.com` trust relationship are displayed. For instructions on how to create an IAM role for your tasks, see [Creating an IAM role and policy for your tasks \(p. 633\)](#).

- b. For **Task Execution Role Override**, choose a task execution role to override the task execution role specified in the task definition. For more information, see [Amazon ECS task execution IAM role \(p. 626\)](#).
8. (Optional) To override the container commands and environment variables, expand **Container Overrides**, and then expand the container.
 - To send a command to the container other than the task definition command, for **Command override**, enter the Docker command.

For more information about the Docker run command, see [Docker Run reference](#) in the Docker Reference Manual.

- To add an environment variable, choose **Add Environment Variable**. For **Key**, enter the name of your environment variable. For **Value**, enter a string value for your environment value (without the surrounding double quotation marks (" ")).

AWS surrounds the strings with double quotation marks (" ") and passes the string to the container in the following format:

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

9. 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.
10. 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](#).
11. Review your task information and choose **Run 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.

10. 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](#).

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

Note

If your task moves from the PENDING to the STOPPED status, your task might be stopping because of an error. This is also the case if it displays a PENDING status and then disappears from the listed tasks. For more information, see [Checking stopped tasks for errors \(p. 766\)](#) in the troubleshooting section.

Command line

Use the `run-task` command. For more information, see [run-task](#) in the *AWS Command Line Interface Reference*.

Service management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Creating an Amazon ECS service in the classic console \(p. 899\)](#)
- [Updating a service using the classic console \(p. 911\)](#)
- [Deleting a service using the classic console \(p. 913\)](#)

Creating an Amazon ECS service in the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

When you create an Amazon ECS service, you specify the parameters that define what makes up your service and how your service behaves. These parameters create a service definition. For more information, see [Service definition parameters \(p. 847\)](#).

For services that are hosted on Fargate or Amazon EC2 instances, you can optionally configure an Elastic Load Balancing load balancer to distribute traffic across the containers in your service. For more information, see [Service load balancing \(p. 486\)](#).

Note

When using a load balancer with services that are hosted on Amazon EC2 instances, verify that your instances can receive traffic from your load balancers. You can allow traffic to all ports on your instances from your load balancer's security group. This ensures that traffic can reach any containers that use dynamically assigned ports.

Creating a service using the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

The Amazon ECS console provides a create service wizard that guides you through each step to create a service.

If you have not already done so, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 900\)](#).

Topics

- [Step 1: Configuring basic service parameters \(p. 900\)](#)
- [Step 2: Configure a network \(p. 902\)](#)
- [Step 3: Configuring your service to use a load balancer \(p. 903\)](#)
- [Step 4: Configuring your service to use Service Discovery \(p. 907\)](#)
- [Step 5: Configuring your service to use Service Auto Scaling \(p. 908\)](#)
- [Step 6: Review and create your service \(p. 910\)](#)

Step 1: Configuring basic service parameters

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

All services require some configuration parameters that define the service, such as the task definition to use, which cluster that the service runs on, how many tasks are placed for the service. This is called the *service definition*. For more information about the parameters defined in a service definition, see [Service definition parameters \(p. 847\)](#).

This procedure covers how to create a service and covers the service definition parameters that are required. After you configured these parameters, you can create your service or move on to the procedures for optional service definition configuration. For example, you can move on to configuring your service to use a load balancer.

Note

If your cluster is configured with a default capacity provider strategy, you can only create a service using the default capacity provider strategy when using the console. Likewise, if no default capacity provider is defined, you can only use a launch type when creating a service using the console. It's not currently possible to have a mixed strategy using both capacity providers and launch types in the console.

To configure the basic service definition parameters

1. Open the Amazon ECS classic 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 choose the task definition to create your service from.
4. On the **Task Definition name** page, choose the revision of the task definition to create your service from.
5. Review the task definition, and then choose **Actions, Create Service**.
6. On the **Configure service** page, complete the following steps.
 - a. Choose either a capacity provider strategy or a launch type.
 - To use a **Capacity provider strategy**, first choose **Switch to capacity provider strategy**. Next, choose whether your service uses the default capacity provider strategy that's defined for the cluster or a custom capacity provider strategy. A capacity provider must already be associated with the cluster to be used in a custom capacity provider strategy. For more information, see [Amazon ECS capacity providers \(p. 220\)](#).
 - To use a **Launch type**, choose **Switch to launch type** and select **FARGATE**, **EC2**, or **EXTERNAL**. For more information about launch types, see [Amazon ECS launch types \(p. 85\)](#).

- b. For **Platform operating system**, if you chose the Fargate launch type, then select the platform operating system (for example, **LINUX**).
- c. For **Platform version**, if you chose a Fargate capacity provider or the Fargate launch type, then select the platform version to use.

Note

When the **LATEST** platform version is selected, we validate the operating system that was specified for the task, and then set the appropriate platform version.

If the Operating System is set to **Windows-Server-2019-Full** or **Windows-Server-2019-Core**, the **1.0.0** platform is used. If the operating system is Linux, the **1.4.0** platform version is used.

- d. **Cluster:** Select the cluster to create your service in.
- e. **Service name:** Enter a unique name for your service.
- f. **Service type:** Select a scheduling strategy for your service. For more information, see [Service scheduler concepts \(p. 453\)](#).
- g. **Number of tasks:** If you chose the **REPLICA** service type, enter 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 doesn't apply if your task definition uses dynamic host port mappings with the bridge network mode. For more information, see [portMappings \(p. 807\)](#).

- h. If you're using the **Rolling update** deployment type, specify the following deployment configuration parameters. For more information about how these parameters are used, see [Deployment configuration \(p. 851\)](#).
 - **Minimum healthy percent:** Enter a lower limit for the number of tasks that your service must remain in the **RUNNING** state during a deployment. Specify the number as a percentage of the desired number of tasks. This number must be a whole number.
 - **Maximum percent:** Enter an upper limit for the number of tasks that your service allows in the **RUNNING** or **PENDING** state during a deployment. Specify the number as a percentage of the desired number of tasks. This number must be a whole number.
7. For **Deployment circuit breaker**, choose the deployment circuit breaker logic. For more information, see [the section called "Deployment circuit breaker" \(p. 474\)](#).
8. On the **Deployments** page, complete the following steps.
 - a. For **Deployment type**, choose whether your service uses a rolling update deployment or a blue/green deployment using AWS CodeDeploy. For more information, see [Amazon ECS Deployment types \(p. 472\)](#).
 - b. For blue/green deployments, complete the following steps:
 - i. For **Deployment configuration** choose how traffic is shifted when your task set is updated. For more information, see [Blue/Green deployment with CodeDeploy \(p. 477\)](#)
 - ii. For **Service role for CodeDeploy** choose the IAM service role for AWS CodeDeploy. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 644\)](#)
9. (Optional) If you selected the EC2 launch type and the **REPLICA** service type, for **Task Placement**, select how tasks are placed using task placement strategies and constraints.
 - **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
 - **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
 - **BinPack** - Distribute tasks based on the least available amount of CPU or memory.

- **One Task Per Host** - Place, at most, one task from the service on each container instance.
- **Custom** - Define your own task placement strategy. See [Amazon ECS task placement \(p. 434\)](#) for examples.

10. 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.

11. 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](#).
12. Choose **Next step** and navigate to [Step 2: Configure a network \(p. 902\)](#).

Step 2: Configure a network

If your service's task definition uses the `awsvpc` network mode, you must configure a VPC, subnet, and security group for your service.

If your service's task definition uses the bridge, host, or none network modes, you can move on to the next step, [Step 3: Configuring your service to use a load balancer \(p. 903\)](#).

For tasks that are hosted on Amazon EC2 instances, the `awsvpc` network mode doesn't provide task ENIs with public IP addresses. To access the internet, tasks that are hosted on Amazon EC2 instances can be launched in a private subnet that's configured to use a NAT gateway. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC using the private IP address or DNS hostname, or routed through a load balancer from within the VPC. Tasks that are launched within public subnets don't have internet access.

To configure VPC and security group settings for your service

1. If you host tasks on EC2 instances, for **Cluster VPC**, choose the VPC that your instances are in.

If you host tasks on Fargate, or **Cluster VPC**, choose the VPC that the Amazon ECS on Fargate tasks use. The VPC cannot be configured to require dedicated hardware tenancy, because Fargate does not support the feature.
2. For **Subnets**, choose the available subnets for your service task placement.
3. For **Security groups**, choose a security group was created for your service's tasks. This security group allows HTTP traffic access 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. For tasks on Fargate, for the task to pull the container image, it must either use a public subnet and be assigned a public IP address or a private subnet that has a route to the internet or a NAT gateway that can route requests to the internet.

5. If you're configuring your service to use a load balancer or if you're using the blue/green deployment type, continue to [Step 3: Configuring your service to use a load balancer \(p. 903\)](#). If you aren't configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, [Step 5: Configuring your service to use Service Auto Scaling \(p. 908\)](#).

Step 3: Configuring your service to use a load balancer

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Services can be configured to use a load balancer to distribute incoming traffic to the tasks in your service. If your service is using the rolling update deployment type, this is optional. If your service is using the blue/green deployment type, then it is required to use either an Application Load Balancer or Network Load Balancer.

If you aren't configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, [Step 4: Configuring your service to use Service Discovery \(p. 907\)](#).

If you configured an available Elastic Load Balancing load balancer, 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 \(p. 490\)](#).

Important

Before following these procedures, you must create your Elastic Load Balancing load balancer resources.

Topics

- [Configuring a load balancer for the rolling update deployment type \(p. 903\)](#)
- [Configuring a load balancer for the blue/green deployment type \(p. 905\)](#)

Configuring a load balancer for the rolling update deployment type

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

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 2,147,483,647 seconds (about 68 years). During that time, the service scheduler ignores health check status. This grace period can prevent the 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.

To configure a health check grace period

1. If you didn't do so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 900\)](#).
2. For **Health check grace period**: Enter the period of time, in seconds, that the Amazon ECS service scheduler ignores unhealthy Elastic Load Balancing target health checks after a task has first started.

To configure your service to use a load balancer, you must choose the load balancer type to use with your service.

To choose a load balancer type

1. If you didn't do so already, follow the basic service creation procedures in [Step 1: Configuring basic service parameters \(p. 900\)](#).
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. With host port mapping allowed, you can 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. With host port mapping allowed, you can 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.

We recommend that you use Application Load Balancers for your Amazon ECS services. That way, you can use the advanced features of Application Load Balancer.

3. For **Select IAM role for service**, choose **Create new role** to create the Amazon ECS service-linked role or select your existing service-linked role.
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 that 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 \(p. 904\)](#). If you've chosen a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 905\)](#).

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 distributes 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 \(p. 491\)](#) (if applicable). Alternatively, choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Listener protocol**.
3. For **Target group name**, choose the target group that you created in [Creating an Application Load Balancer \(p. 491\)](#) (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**, a default name is provided for you.
 - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
 - For **Path pattern**, if your listener doesn't 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 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 that the load balancer sends health check pings to.
5. When you're 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 a Network Load Balancer \(p. 493\)](#) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Listener protocol**.
3. For **Target group name**, choose the target group that you created in [Creating a Network Load Balancer \(p. 493\)](#) (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**, a default name is provided for you.
 - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
 - For **Health check path**, enter the path that the load balancer sends health check pings to.
5. When you're finished configuring your Network Load Balancer, choose **Next Step**.

Configuring a load balancer for the blue/green deployment type

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

To configure your service that uses the blue/green deployment type to use a load balancer, you must use either an Application Load Balancer or a Network Load Balancer.

To choose a load balancer type

1. If you didn't do so already, follow the procedures to create the service in [Step 1: Configuring basic service parameters \(p. 900\)](#).
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. With host port mapping, you can 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. With host port mapping, you can place multiple tasks using the same port on a single container instance.

We recommend that you use Application Load Balancers for your Amazon ECS services. That way, you can use all of the Application Load Balancer features.

3. For **Load balancer name**, choose the name of the load balancer to use with your service. Only load balancers that correspond to the load balancer type that you selected earlier are visible here.
4. The next step depends on the load balancer type for your service. If you chose an Application Load Balancer, follow the steps in [To configure an Application Load Balancer \(p. 904\)](#). If you chose a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 905\)](#).

To configure an Application Load Balancer for the blue/green deployment type

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer distributes traffic to, and choose **Add to load balancer**.
2. For **Production listener port**, choose the listener port and protocol of the listener that you created in [Creating an Application Load Balancer \(p. 491\)](#) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Production listener protocol**.
3. (Optional) Select **Test listener** if you want to configure a listener port and protocol on your load balancer to test updates to your service before routing traffic to your new task set. Complete the following step:
 - For **Test listener port**, choose the listener port and protocol of the listener that you want to test traffic over, or choose **create new** to create a new test listener and then enter a port number and choose a port protocol in **Test listener protocol**.
4. For blue/green deployments, two target groups are required. Each target group binds to a separate task set in the deployment. Complete the following steps:
 - a. For **Target group 1 name**, choose the target group that you created in [Creating an Application Load Balancer \(p. 491\)](#) (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.

- b. (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.
- c. Repeat the steps for target group 2.
- d. When you are finished configuring your Application Load Balancer, choose **Next step**. Navigate to [Step 4: Configuring your service to use Service Discovery \(p. 907\)](#).

To configure a Network Load Balancer for the blue/green deployment type

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 a Network Load Balancer \(p. 493\)](#) (if applicable), or choose **Create new** to create a new listener and then enter a port number and choose a port protocol for **Listener protocol**.
3. For **Target group name**, choose the target group that you created in [Creating a Network Load Balancer \(p. 493\)](#) (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**. Navigate to [Step 4: Configuring your service to use Service Discovery \(p. 907\)](#).

Step 4: Configuring your service to use Service Discovery

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Your Amazon ECS service can optionally use service discovery integration, which allows your service to be discoverable via DNS. For more information, see [Service discovery \(p. 522\)](#).

If you are not configuring your service to use a service discovery, you can move on to the next section, [Step 5: Configuring your service to use Service Auto Scaling \(p. 908\)](#).

To configure service discovery

1. If you have not done so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 900\)](#).
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 [Supported DNS Record Types](#) in the *Amazon Route 53 Developer Guide*.

- 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** to proceed and navigate to [Step 5: Configuring your service to use Service Auto Scaling \(p. 908\)](#).

Step 5: Configuring your service to use Service Auto Scaling

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Your Amazon ECS service can optionally be configured to use Auto Scaling to adjust its desired count of tasks in your Amazon ECS service up or down in response to CloudWatch alarms.

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

- [Target tracking scaling policies \(p. 501\)](#) (Recommended)—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 \(p. 501\)](#)—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 \(p. 498\)](#).

To configure basic Service Auto Scaling parameters

1. If you have not done so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 900\)](#).
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 the `ecsAutoscaleRole`. If this role does not exist, choose **Create new role** to have the console create it for you.

7. The following procedures provide steps for creating either target tracking or step scaling policies for your service. Choose your desired scaling policy type.

These steps help you create target tracking scaling policies and CloudWatch alarms that can be used to initiate scaling activities for your service.

To configure target tracking scaling policies for 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 to track. The following metrics are available:
 - **ECSServiceAverageCPUUtilization**—Average CPU utilization of the service.
 - **ECSServiceAverageMemoryUtilization**—Average memory utilization of the service.
 - **ALBRequestCountPerTarget**—Number of requests completed per target in an Application Load Balancer target group.
4. For **Target value**, enter the metric value that the policy should maintain. For example, use a target value of 1000 for ALBRequestCountPerTarget, or a target value of 75(%) for ECSServiceAverageCPUUtilization.
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. While the scale-out cooldown period is in effect, the capacity that has been added by the previous scale-out activity that initiated the cooldown is calculated as part of the desired capacity for the next scale out. The intention is to continuously (but not excessively) scale out.
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. The scale-in cooldown period is used to block subsequent scale-in requests until it has expired. The intention is to scale in conservatively to protect your application's availability. However, if another alarm triggers a scale out activity during the cooldown period after a scale-in, Service Auto Scaling scales out your scalable target immediately.
7. (Optional) To turn off the scale-in actions for this policy, choose **Disable scale-in**. This allows you to create a separate scaling policy for scale-in later.
8. Choose **Next step**.

These steps help you create step scaling policies and CloudWatch alarms that can be used to initiate 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.

To configure step scaling policies for 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 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 initiate when your service CPU utilization exceeds 75%, you could call the alarm *service_name*-cpu-gt-75.

- b. For **ECS service metric**, choose the service metric to use for your alarm. For more information, see [Service auto scaling \(p. 498\)](#).
- 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**.
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 \(p. 910\)](#) 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 **Coldown period**, enter the number of seconds between scaling actions.
9. Repeat [Step 1 \(p. 909\)](#) through [Step 8 \(p. 910\)](#) for the **Scale in** policy and choose **Save**.
10. Choose **Next step** to proceed and navigate to [Step 6: Review and create your service \(p. 910\)](#).

Step 6: Review and create your service

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

After you have configured your basic service definition parameters and optionally configured your service's networking, load balancer, service discovery, and automatic scaling, you can review your configuration. Then, choose **Create Service** to finish creating your service.

Note

After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the ECS rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load

balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*.

Updating a service using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can update an existing service to change some of the service configuration parameters, such as the number of tasks that are maintained by a service, which task definition is used by the tasks, or if your tasks are using the Fargate launch type, you can change the platform version your service uses. A service using a Linux platform version cannot be updated to use a Windows platform version and vice versa. If you have an application that needs more capacity, you can scale up your service. If you have unused capacity to scale down, you can reduce the number of desired tasks in your service and free up resources.

If you want to use an updated container image for your tasks, you can create a new task definition revision with that image and deploy it to your service by using the **force new deployment** option in the console.

The service scheduler uses the minimum healthy percent and maximum percent parameters (in the deployment configuration for the service) to determine the deployment strategy.

If a service is using the rolling update (ECS) deployment type, the **minimum healthy percent** represents a lower limit on the number of tasks in a service 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). The parameter also applies while any container instances are in the DRAINING state if the service contains tasks using the EC2 launch type. Use this parameter 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 percent, 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 they are reported as healthy by the load balancer. The default value for minimum healthy percent is 100 percent.

If a service is using the rolling update (ECS) deployment type, the **maximum percent** parameter represents an upper limit on the number of tasks in a service that are allowed in the PENDING, RUNNING, or STOPPING state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer). The parameter also applies while any container instances are in the DRAINING state if the service contains tasks using the EC2 launch type. Use this parameter 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 percent, the scheduler may start four new tasks before stopping the four older tasks. That is provided that the cluster resources required to do this are available. The default value for the maximum percent is 200 percent.

If a service is using the blue/green (CODE_DEPLOY) deployment type and tasks that use the EC2 launch type, the **minimum healthy percent** and **maximum percent** values are set to the default values. They are only used to define the lower and upper limit on the number of the tasks in the service that remain in the RUNNING state while the container instances are in the DRAINING state. If the tasks in the service use the Fargate launch type, the minimum healthy percent and maximum percent values are not used. They are currently visible when describing your service.

When the service scheduler replaces a task during an update, the service first removes the task from the load balancer (if used) 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 the security groups for the container instances to work with the updated ports.

You can use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*.

If you update the task definition for the service, the container name and container port that are specified in the load balancer configuration must remain in the task definition.

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 classic 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, capacity provider strategy, platform version, deployment configuration, or number of desired tasks (or any combination of these). To have your service start a new deployment, which will stop and relaunch all tasks using the new configuration, select **Force new deployment**. Choose **Next step** when finished changing the service configuration.

Note

A service using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider and vice versa.

A service using a Linux platform version can't be updated to use a Windows platform version and vice versa.

8. On the **Configure deployments** page, if your service is using the blue/green deployment type, the components of your service deployment is pre-populated. Confirm the following settings.
 - a. For **Application name**, choose the CodeDeploy application of which your service is a part.
 - b. For **Deployment group name**, choose the CodeDeploy deployment group of which your service is a part.
 - c. Select the deployment lifecycle event hooks and the associated Lambda functions to execute as part of the new revision of the service deployment. The available lifecycle hooks are:
 - **BeforeInstall** – Use this deployment lifecycle event hook to invoke a Lambda function before the replacement task set is created. The result of the Lambda function at this lifecycle event does not initiate a rollback.
 - **AfterInstall** – Use this deployment lifecycle event hook to invoke a Lambda function after the replacement task set is created. The result of the Lambda function at this lifecycle event can initiate a rollback.
 - **BeforeAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function before the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can initiate a rollback.

- **AfterAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function after the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can initiate a rollback.

For more information about lifecycle hooks, see [AppSpec 'hooks' Section](#) in the *AWS CodeDeploy User Guide*.

9. Choose **Next step**.
10. On the **Configure network** page, your network information is pre-populated. In the **Load balancing** section, if your service is using the blue/green deployment type, select the listeners to associate with the target groups. Change the health check grace period (if desired) and choose **Next step**.
11. (Optional) You can use Service Auto Scaling to scale your service up and down automatically in response to CloudWatch alarms.
 - a. Under **Optional configurations**, choose **Configure Service Auto Scaling**.
 - b. Proceed to [Step 5: Configuring your service to use Service Auto Scaling \(p. 908\)](#).
 - c. Complete the steps in that section and then return.
12. Choose **Update Service** to finish and update your service.

Deleting a service using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can delete an Amazon ECS service using the console. Before deletion, the service is automatically scaled down to zero. If you have a load balancer or service discovery resources associated with the service, they are not affected by the service deletion. To delete your Elastic Load Balancing resources, see one of the following topics, depending on your load balancer type: [Delete an Application Load Balancer](#) or [Delete a Network Load Balancer](#). To delete your service, follow the procedure below.

Classic console

Use the following procedure to delete an Amazon ECS service.

1. Open the Amazon ECS classic 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** and select the name of the cluster in which your service resides.
4. On the **Cluster : *name*** page, choose **Services**.
5. Check the box to the left of the service to update and choose **Delete**.
6. Confirm the service deletion by entering the text phrase and choose **Delete**.

AWS CLI

To delete the remaining service discovery resources, you can use the AWS CLI to delete the service discovery service and service discovery namespace.

1. Ensure that 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](#).

2. Retrieve the ID of the service discovery service to delete.

```
aws servicediscovery list-services --region <region_name>
```

Note

If no service discovery service is returned, continue to step 4.

3. Using the service discovery service ID from the previous output, delete the service.

```
aws servicediscovery delete-service --id <service_discovery_service_id> --region <region_name>
```

4. Retrieve the ID of the service discovery namespace to delete.

```
aws servicediscovery list-namespaces --region <region_name>
```

5. Using the service discovery namespace ID from the previous output, delete the namespace.

```
aws servicediscovery delete-namespace --id <service_discovery_namespace_id> --region <region_name>
```

Tag management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Adding and deleting tags on an individual resource using the classic console \(p. 914\)](#)

Adding and deleting tags on an individual resource using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Amazon ECS allows you to add or delete tags that are associated with your clusters, services, tasks, and task definitions directly from the resource's page. For information about tagging your container instances, see [Adding tags to an Amazon EC2 container instance \(p. 534\)](#).

To add a tag to an individual resource

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

2. From the navigation bar, select the AWS 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 classic 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**.

Account setting management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Modifying account settings using the classic console \(p. 915\)](#)
- [Viewing account settings using the classic console \(p. 916\)](#)

Modifying account settings using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can manage your account settings using the classic console.

1. Open the Amazon ECS classic 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 your account settings.
3. Choose **Account settings**.
4. On the **Amazon ECS ARN and resource ID settings**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can select or deselect the check boxes for each account setting for the authenticated IAM user and role. Choose **Save** once finished.

Important

IAM users and IAM roles need the `ecs:PutAccountSetting` permission to perform this action.

5. On the confirmation screen, choose **Confirm** to save the selection.

Viewing account settings using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can use the classic AWS Management Console to view your account settings.

Important

The dualStackIPv6 account setting can only be viewed or changed using the AWS CLI.

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. Choose **Account settings**.
4. On the **Amazon ECS ARN and resource ID settings**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can view your opt-in status for each account setting for the authenticated IAM user and role.

Container instance management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Connect to your container Windows instance using the classic console \(p. 916\)](#)
- [Deregister an Amazon EC2 backed container instance using the classic console \(p. 917\)](#)
- [Registering an external instance to a cluster using the classic console \(p. 918\)](#)
- [Deregistering an external instance using the classic console \(p. 919\)](#)

Connect to your container Windows instance using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

You can connect to your Windows instances to perform basic administrative tasks, such as installing or updating software or accessing diagnostic logs. To connect to your instance using Remote Desktop Protocol (RDP), your Windows instance must meet the following prerequisites.

- Amazon EC2 instances created from most Windows AMIs allow you to connect using Remote Desktop Protocol (RDP). RDP allows you to connect to and use your instance in the same way you use a computer sitting in front of you. It is available on most editions of Windows and available for Mac OS.
 - Your Windows instance must have been launched with a valid Amazon EC2 key pair. Amazon EC2 instances have no password, you use a key pair for access over RDP. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance.
 - Ensure that the security group associated with your instance allows incoming RDP traffic (port 3389) from your IP address. The default security group doesn't allow incoming RDP traffic by default. For more information, see [Authorize inbound traffic for your Windows instances](#) in the *Amazon EC2 User Guide for Windows Instances*.
1. Find the public IP or DNS address for your container instance.
 - a. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
 - b. Select the cluster that hosts your container instance.
 - c. On the **Cluster** page, choose **ECS Instances**.
 - d. On the **Container Instance** column, select the container instance to connect to.
 - e. On the **Container Instance** page, record the **Public IP** or **Public DNS** for your instance.
 2. Find the default username for your container instance AMI.
 3. You can connect to your instance by using RDP. For more information, see [Connect to your Windows instance using RDP](#) in the *Amazon EC2 User Guide for Windows Instances*.

Deregister an Amazon EC2 backed container instance using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Important

This topic is for container instances created in Amazon EC2 only.

When you are finished with an Amazon EC2 backed container instance, you should deregister it from your cluster. Following deregistration, the container instance is no longer able to accept new tasks.

If you have tasks running on the container instance when you deregister it, these tasks remain running until you terminate the instance or the tasks stop through some other means. However, these tasks are orphaned which means they are no longer monitored or accounted for by Amazon ECS. If an orphaned task on your container instance is part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different container instance, if possible. Any containers in orphaned service tasks that are registered with an Application Load Balancer target group are deregistered. They begin connection draining according to the settings on the load balancer or target group. If an orphaned tasks is using the awsVpc network mode, their elastic network interfaces are deleted.

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

When deregistering a container instance, be aware of the following considerations.

- Because each container instance has unique state information, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances in the new cluster. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
- If the container instance is managed by an Auto Scaling group or a AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack. Otherwise, the Auto Scaling group or AWS CloudFormation will create a new instance after you terminate it.
- If you terminate a running container instance with a connected Amazon ECS container agent, the agent automatically deregisters the instance from your cluster. Stopped container instances or instances with disconnected agents are not automatically deregistered when terminated.
- Deregistering a container instance removes the instance from a cluster, but it does not terminate the Amazon EC2 instance. If you are finished using the instance, be sure to terminate it to stop billing. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

- Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
- From the navigation bar, choose the Region in which your container instance is registered.
- In the navigation pane, choose **Clusters** and select the cluster that hosts the container instance.
- On the **Cluster : name** page, choose the **ECS Instances** tab.

Container Instance	EC2 Instance	Agent ...	Status	Availa...	Availa...
3de21d77-d1d7-4795-a3b3-ed6e65d7d353	i-501f2599	true	ACTIVE	2048	3955

- Select the container instance ID to deregister. This takes you to the container instance detail page.
- On the **Container Instance : id** page, choose **Deregister**.
- Review the deregistration message and choose **Deregister**.
- If you are finished with the container instance, terminate the underlying Amazon EC2 instance. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

Registering an external instance to a cluster using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

For each external instance you register with an Amazon ECS cluster, it must have the SSM Agent, the Amazon ECS container agent, and Docker installed. To register the external instance to an Amazon ECS cluster, it must first be registered as an AWS Systems Manager managed instance. You can create the installation script in a few clicks on the Amazon ECS console. The installation script includes an Systems

Manager activation key and commands to install each of the required agents and Docker. The installation script must be run on your on-premises server or VM to complete the installation and registration steps.

Note

Before registering your Linux external instance with the cluster, create the `/etc/ecs/ecs.config` file on your external instance and add any container agent configuration parameters that you want. You can't do this after registering the external instance to a cluster.

1. Open the Amazon ECS classic 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 a cluster to register your external instance to.
5. Choose the **ECS Instances** tab, then choose **Register external instances**.
6. On the **Step 1: External instances activation details** page, complete the following steps.
 - a. For **Activation key duration (in days)**, enter the number of days that the activation key remains active for. After the number of days you entered pass, the key no longer works when registering an external instance.
 - b. For **Number of instances**, enter the number of external instances that you want to register to your cluster with the activation key.
 - c. For **Instance role**, choose the IAM role to associate with your external instances. If a role wasn't already created, choose **Create new role** to have Amazon ECS create a role on your behalf.
 - d. Choose **Next step**.
7. On the **Step 2: Register external instances** page, copy the registration command. This command should be run on each external instance you want to register to the cluster.

Important

The bash portion of the script must be run as root. If the command isn't run as root, an error is returned.

Deregistering an external instance using the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

We recommend that, after you finish using an external instance, you deregister the instance from both Amazon ECS and AWS Systems Manager. Following deregistration, the external instance is no longer able to accept new tasks.

If you have tasks that are running on the container instance when you deregister it, the tasks remain running until they stop through some other means. However, these tasks are no longer monitored or accounted for by Amazon ECS. If these tasks on your external instance are part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different instance, if possible.

To register an external instance to a new cluster, after the external instance has been deregistered from both Amazon ECS and Systems Manager, you can clean up the remaining AWS resources on the instance and register it with a new cluster.

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

2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.
4. On the **Cluster : *name*** page, choose the **ECS Instances** tab.



The screenshot shows the AWS ECS Instances page. At the top, there are tabs for Services, Tasks, and ECS Instances, with ECS Instances being the active tab. Below the tabs is a button to "Add additional ECS Instances using Auto Scaling or Amazon EC2". A search bar labeled "Filter in this page" is followed by a breadcrumb trail "Viewing 1-1 Container Instance". The main area displays a table with one row. The columns are: Container Instance, EC2 Instance, Agent ... Status, Availa..., and Availa.... The first column contains the ID "3de21d77-d1d7-4795-a3b3-ed6e65d7d353". The second column contains the EC2 instance ID "i-501f2599". The third column contains the status "true". The fourth column contains the status "ACTIVE". The fifth and sixth columns contain the availability counts "2048" and "3955" respectively.

5. Select the external instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance : *id*** page, choose **Deregister**.
7. Review the deregistration message. Select **Deregister from AWS Systems Manager** to also deregister the external instance as an Systems Manager managed instance. Choose **Deregister**.

Note

You can deregister the external instance as an Systems Manager managed instance in the Systems Manager console. For instructions, see [Deregistering managed instances](#) in the AWS Systems Manager User Guide.

8. If you want to clean up the remaining AWS resources from the external instance, follow these steps. The cleanup steps must be completed before you can register the external instance with a new cluster.

After you deregister the instance, clean up AWS resources on your on-premises server or VM .

Linux operating system

1. Make sure that the external instance is deregistered from both Amazon ECS and Systems Manager.
2. Stop the Amazon ECS container agent and the SSM Agent services on the instance.

```
sudo systemctl stop ecs amazon-ssm-agent
```

3. Remove the Amazon ECS and Systems Manager packages.

For CentOS 7, CentOS 8, and RHEL 7

```
sudo yum remove -y amazon-ecs-init amazon-ssm-agent
```

For SUSE Enterprise Server 15

```
sudo zypper remove -y amazon-ecs-init amazon-ssm-agent
```

For Debian and Ubuntu

```
sudo apt remove -y amazon-ecs-init amazon-ssm-agent
```

4. Remove the Amazon ECS and Systems Manager files.

```
sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ssm /var/log/ecs /var/log/amazon/ssm
```

Windows operating system

1. Make sure that the external instance is deregistered from both Amazon ECS and Systems Manager.
2. Stop the Amazon ECS container agent and the SSM Agent services on the instance.

```
Stop-Service AmazonECS
```

```
Stop-Service AmazonSSMAgent
```

3. Remove the Amazon ECS package.

```
.\ecs-anywhere-install.ps1 -Uninstall
```

Container agent management in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Updating the Amazon ECS container agent with the classic console \(p. 921\)](#)

Updating the Amazon ECS container agent with the classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Use the classic console to update your container agent.

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster that hosts the container instance or instances to check.

3. On the Cluster : **cluster-name** page, choose **ECS Instances**.
4. Select the container instance to update.
5. On the Container Instance page, choose **Update agent**.

Monitoring and troubleshooting in the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Topics

- [Viewing cluster metrics using the classic Amazon ECS console \(p. 922\)](#)
- [Viewing service metrics using the classic Amazon ECS console \(p. 922\)](#)
- [Checking stopped tasks for errors in the Amazon ECS classic console \(p. 923\)](#)

Viewing cluster metrics using the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Cluster and service metrics are available on the Amazon ECS console. The view provided for cluster metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information about cluster metrics, see [Cluster reservation \(p. 554\)](#) and [Cluster utilization \(p. 555\)](#).

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. Select the cluster that you want to view metrics for.
3. On the Cluster: **cluster-name** page, choose **Metrics**.

Viewing service metrics using the classic Amazon ECS console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

Amazon ECS service CPU and memory utilization metrics are available on 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 \(p. 556\)](#).

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. Select the cluster that contains the service that you want to view metrics for.
3. On the **Cluster: *cluster-name*** page, choose **Services**.
4. Choose the service that you want to view metrics for.
5. On the **Service: *service-name*** page, choose **Metrics**.

Checking stopped tasks for errors in the Amazon ECS classic console

The classic Amazon ECS console is reaching the end of life and will no longer be available after December 4, 2023. We recommend that you switch immediately to the new Amazon ECS console for a better experience. You can review and follow the new Amazon ECS console roadmap on GitHub. For more information, see the [container-roadmap](#) on GitHub.

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 stopped task errors like this in the Amazon ECS console by viewing the stopped task and inspecting it for error messages. If your task definition uses the awslogs log driver, the application logs that are written to Amazon CloudWatch Logs are displayed on the **Logs** tab in the Amazon ECS console as long as the stopped task appears.

Important

Stopped tasks only appear in the Amazon ECS console, AWS CLI, and AWS SDKs for at least 1 hour after the task stops. After that, the details of the stopped task expire and aren't available in Amazon ECS.

Amazon ECS also sends task state change events to Amazon EventBridge. You can't view events in EventBridge. Instead, you create rules to send the events to other persistent storage such as Amazon CloudWatch Logs. You can use the storage to view your stopped task details after it has expired from view in the Amazon ECS console. For more information, see [Task state change events \(p. 563\)](#).

For a sample EventBridge configuration to archive Amazon ECS events to Amazon CloudWatch Logs, see [ECS Stopped Tasks in CloudWatch Logs](#) on the GitHub website.

1. Open the Amazon ECS classic console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster where 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's associated with the task's service. For more information, see [Troubleshooting service load balancers \(p. 778\)](#).

Scaling activity initiated by (deployment deployment-id)

When you reduce the desired count of a stable service, some tasks must be stopped 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

If a container marked as essential in task definitions exits or dies, that can cause a task to stop. When an essential container exiting is the cause of a stopped task, the [Step 6 \(p. 924\)](#) can provide more diagnostic information about 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.

If this inspection doesn't provide enough information, you can connect to the container instance with SSH and inspect the Docker container locally. For more information, see [Inspect Docker Containers \(p. 789\)](#).

Related information

The following related resources can help you as you work with this service.

- [AWS Fargate](#) – Overview of Fargate features.
- [Windows on AWS](#) – Overview of Windows on AWS workloads and services.
- [Linux from AWS](#) – Portfolio of modern Linux-based operating systems from AWS.

Tutorials for developers

- [AWS Compute Blogs](#) – Information about new features, deep dives into features, code samples and best practices.

AWS re:Post

[AWS re:Post](#) – AWS managed question and answer (Q & A) service offering crowd-sourced, expert-reviewed answers to your technical questions.

Pricing

- [Amazon ECS pricing](#) – Pricing information for Amazon ECS.
- [AWS Fargate pricing](#) – Pricing information for Fargate.

General AWS resources

The following general resources can help you as you work with AWS.

- [Classes & Workshops](#) – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- [AWS Developer Center](#) – Explore tutorials, download tools, and learn about AWS developer events.
- [AWS Developer Tools](#) – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- [Getting Started Resource Center](#) – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- [Hands-On Tutorials](#) – Follow step-by-step tutorials to launch your first application on AWS.
- [AWS Whitepapers](#) – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- [AWS Support Center](#) – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- [AWS Support](#) – The primary webpage for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- [Contact Us](#) – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- [AWS Site Terms](#) – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Amazon ECS API reference

In addition to the AWS Management Console and the AWS Command Line Interface (AWS CLI), Amazon ECS also provides an API. You can use the API to automate tasks for managing Amazon ECS resources.

- For a list of API operations by Amazon ECS resource, see [Actions by Amazon ECS resource](#).
- For an alphabetical list of API operations, see [Actions](#).
- For an alphabetical list of data types, see [Data types](#).
- For a list of common query parameters, see [Common parameters](#).
- For descriptions of the error codes, see [Common errors](#).

For more information about the AWS CLI, see [AWS Command Line Interface reference for Amazon ECS](#).

Document history

The following table describes the major updates and new features for the *Amazon Elastic Container Service Developer Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
Additional task definition parameters in AWS Fargate	AWS Fargate adds support for <code>pidMode</code> and <code>systemControls</code> in Linux platform version 1.4.0. For more information, see Task definitions .	09 August 2023
Amazon ECS console task definition page redesign	The task definition page in the Amazon ECS console has been redesigned and contains additional options. For more information, see Creating a task definition using the console .	26 July 2023
Fargate supports lazy loading with Seekable OCI indexes	AWS Fargate is introducing Seekable OCI (SOCI) indexes. With SOCI, containers only spend a few seconds on the image pull before they can start, providing time for environment setup and application instantiation while the image is downloaded in the background. For more information, see Lazy loading container images using Seekable OCI (SOCI) in the <i>Amazon ECS User Guide for AWS Fargate</i> .	17 July 2023
Improved support for gMSA on Linux and Windows	The task definition has a new <code>credentialSpecs</code> field for gMSA for Linux and Windows. A new complete tutorial for domainless gMSA on Windows has been added, see Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI . For more information, see Using gMSAs for Linux Containers and Using gMSAs for Windows Containers .	14 July 2023
Improved ECS Agent versions documentation	The documentation for the Amazon ECS Agent versions has been updated. We recommend that you use the v20.10.13 version or newer of Docker with the latest version of the Amazon ECS container agent. The released versions and changes to the agent are available on GitHub. For more information, see Amazon ECS Linux container agent versions .	20 June 2023
Updated Region availability for Fargate ARM64 support	The Region availability for Fargate ARM64 support has been updated. For more information, see Considerations (p. 159) .	19 June 2023
Improve cluster auto scaling documentation	The documentation for Amazon ECS scaling of Amazon EC2 Auto Scaling has significant improvements in accuracy and readability. For more information, see Amazon ECS cluster auto scaling .	4 May 2023
Tagging authorization for resource creation.	Users must have permissions for actions that create the resource, such as <code>ecsCreateCluster</code> . When you create a resource and specify tags for that resource, AWS performs additional authorization to verify that there	18 April 2023

Change	Description	Date
	are permissions to create tags. For more information, see Tagging authorization and Grant permission to tag resources on creation .	
Support for gMSA for Linux containers on EC2	You can use gMSA to authenticate to Active Directory for Linux containers on EC2. For more information, see Using gMSAs for Linux Containers .	14 April 2023
Support for ephemeral storage for Windows containers on AWS Fargate	You can use ephemeral storage for Windows containers on AWS Fargate. For more information, see Fargate task storage .	14 April 2023
AWS Cost Management support for task-level CUR data	You can turn on task-level cost and resource usage in the Cost and Usage Reports. This adds Split Cost Allocation Data for tasks that run on AWS Fargate and EC2. For more information, see Task-level Cost and Usage Reports (p. 544) .	12 April 2023
Amazon Linux 2023 Amazon ECS-optimized AMI	You can deploy workloads on the Amazon Linux 2023 Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized AMI (p. 255) .	10 April 2023
AWS Fargate Federal Information Processing Standard (FIPS) 140	You can deploy workloads on Amazon ECS on AWS Fargate in a manner compliant with Federal Information Processing Standard (FIPS) 140. For more information, see AWS Fargate Federal Information Processing Standard (FIPS-140) (p. 657) .	10 April 2023
Task definition deletion	You can delete a task definition using the Amazon ECS console, SDK, and AWS CLI. For more information, see Deleting a task definition revision using the console and Task definitions .	24 February 2023
AWS Fargate service recommendations in Compute Optimizer	AWS Compute Optimizer generates task and container size recommendations based on the utilization of running tasks in Amazon ECS services on AWS Fargate. For more information, see Viewing recommendations for Amazon ECS services on Fargate .	27 January 2023
Amazon ECS console	The new Amazon ECS console is now the default console. For more information, see New Amazon ECS console .	19 January 2023
Updated AmazonECS_FullAccess IAM policy	The AmazonECS_FullAccess IAM policy is updated to include permissions to add tags to load balancers during creation. For more information, see AmazonECS_FullAccess (p. 612) .	4 January 2023
Use CloudWatch alarms to detect Amazon ECS service deployment failures	You can configure Amazon ECS to set the deployment to failed when it detects that a specified CloudWatch alarm has gone into the ALARM state. For more information, see the section called "Failure detection methods" (p. 473) .	19 December 2022
Support for container port mapping	You can set a port number range on the container that's bound to the dynamically mapped host port range. For more information, see the section called "Port mappings" (p. 807) .	15 December 2022

Change	Description	Date
General availability of Amazon ECS Service Connect	This feature adds service discovery and service mesh that is controlled by Amazon ECS service deployments. For more information, see the section called "Service Connect" (p. 503) .	27 November 2022
The new Amazon ECS console experience for task definitions is updated	The new Amazon ECS console experience now contains a JSON editor for task definitions. For more information, see the section called "Creating a task definition using the console" (p. 125) .	27 October 2022
The new Amazon ECS console experience for task definitions is updated	The new Amazon ECS console experience now contains a JSON editor for task definitions. For more information, see the section called "Creating a task definition using the console" (p. 125) .	27 October 2022
The new Amazon ECS console experience is updated	The new Amazon ECS console experience has been updated with additional service and task parameters. For more information, see the section called "Creating a service using the console" (p. 456) and the section called "Running a standalone task using the Amazon ECS console" (p. 430) .	7 October 2022
New information in task metadata endpoint version 4	The task metadata endpoint version 4 now includes the VPC ID and the service name. For more information, see the section called "Task metadata endpoint version 4" (p. 380) .	7 October 2022
New task definition sizes	Amazon ECS on Fargate now supports the 8 vCPU and 16 vCPU task sizes. For more information, see the section called "Task size" (p. 802) .	16 September 2022
ECS CLI pages archived	The ECS CLI documentation has been archived. We recommend using AWS Copilot for your command line tool needs. For more information, see Using the AWS Copilot command line interface (p. 35) .	15 September 2022
New Fargate quotas	Fargate is transitioning from task count-based quotas to vCPU-based quotas. For more information, see the section called "AWS Fargate service quotas" (p. 538) .	8 September 2022
Support for Amazon EC2 Auto Scaling warm pools.	You can now use Amazon EC2 Auto Scaling warm pools to scale out your applications faster and save costs. For more information, see Using a warm pool for your Auto Scaling group (p. 225) .	23 Mar 2022
Support for Windows instances in ECS Anywhere.	ECS Anywhere now supports Windows instances. For more information, see External instances (Amazon ECS Anywhere) (p. 336) .	03 Mar 2022
Added ECS Exec support for external instances	ECS Exec is now supported for external instances. For more information, see Using Amazon ECS Exec for debugging (p. 755) .	24 Jan 2022

Change	Description	Date
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating and deleting a cluster, updating a task definition, and deregistering a task definition. For more information, see Creating a cluster for the Fargate launch type using the console (p. 236) , Deleting a cluster using the console (p. 240) , Updating a task definition using the console (p. 139) , and Deregistering a task definition revision (p. 895) .	08-Dec 2021
Amazon ECS support for the fluentd log-driver-buffer-limit option	Amazon ECS supports the fluentd log-driver-buffer-limit option. For more information, see Custom log routing (p. 166) .	22 Nov 2021
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating a task definition. For more information, see Creating a task definition using the console (p. 125) .	23 Nov 2021
Amazon ECS supports the 64-bit ARM architecture for Linux.	Amazon ECS supports the 64-bit ARM CPU architecture for the Linux operating system. For more information, see the section called “Working with 64-bit ARM workloads on Amazon ECS” (p. 159) .	23 Nov 2021
Amazon ECS-optimized Linux AMI build script	Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized Linux AMI build script (p. 267) .	19 Nov 2021
Container instance health	Amazon ECS adds support for container instance health monitoring. For more information, see Container instance health (p. 576) .	10 Nov 2021
Support for Windows Amazon ECS Exec	Amazon ECS Exec supports Windows. For more information, see Using Amazon ECS Exec for debugging (p. 755) .	01 Nov 2021
Support for Windows containers on Fargate.	Amazon ECS supports Windows containers on Fargate. For more information, see Windows platform versions (p. 81) .	28 Oct 2021
GPU support for external instances on Amazon ECS Anywhere	Amazon ECS supports specifying GPU requirements in the task definition for tasks run on external instances. For more information, see Working with GPUs on Amazon ECS (p. 142) and Registering an external instance to a cluster (p. 342) .	8 Oct 2021
Support of awsvpc network mode on Windows	Amazon ECS supports awsvpc network mode on Windows. For more information, see awsvpc network mode (p. 91) .	15 July 2021
General availability of Bottlerocket	Amazon ECS supports an Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI. For more information, see Amazon ECS-optimized Bottlerocket AMIs (p. 268) .	30 June 2021

Change	Description	Date
Amazon ECS scheduled tasks update	Amazon EventBridge added support for additional parameters when creating rules that trigger Amazon ECS scheduled tasks. For more information, see Scheduled tasks (p. 445) .	25 June 2021
AWS managed policies for Amazon ECS	Amazon ECS added documentation of AWS managed policies for service-linked roles. For more information, see AWS managed policies for Amazon Elastic Container Service (p. 611) .	08 June 2021
Getting started with the AWS CDK	Added a getting started guide for using the AWS CDK with Amazon ECS. For more information, see Getting started with Amazon ECS using the AWS CDK (p. 45) .	27 May 2021
Amazon ECS Anywhere	Amazon ECS has added support for registering an on-premise server or virtual machine (VM) with your cluster. For more information, see External instances (Amazon ECS Anywhere) (p. 336) .	25 May 2021
Amazon ECS-optimized Windows Server 20H2 Core AMI	Amazon ECS has added support for a new Windows Amazon ECS-optimized AMI variant for Windows Server 20H2 Core. For more information, see Amazon ECS-optimized AMI (p. 255) .	19 April 2021
Amazon ECS Exec	Amazon ECS has released a new debugging tool called ECS Exec. For more information, see Using Amazon ECS Exec for debugging (p. 755) .	15 March 2021
VPC endpoint policy support	Amazon ECS now supports VPC endpoint policies. For more information, see Creating a VPC endpoint policy for Amazon ECS (p. 661) .	11 Jan 2021
New console experience	Amazon ECS has released a new console experience which supports creating or updating a service or running a standalone task. For more information, see Creating a service using the console (p. 456) and Run a standalone task in the classic Amazon ECS console (p. 896) .	28 December 2020
Capacity provider update	Amazon ECS added support for updating an existing Auto Scaling group capacity provider. For more information, see Updating an Auto Scaling group capacity provider using the classic console (p. 883) .	23 November 2020
ECS now supporting Amazon FSx for Windows File Server for Windows tasks	Amazon ECS added support for specifying Amazon FSx for Windows File Server volumes in Windows task definitions. For more information, see FSx for Windows File Server volumes (p. 105) .	11 November 2020
VPC dual-stack mode support added	Amazon ECS added support for using a VPC in dual-stack mode with tasks using the awsvpc network mode, which provides support for IPv6 addresses. For more information, see Using a VPC in dual-stack mode (p. 94) .	5 November 2020
Task metadata endpoint v4 update	Amazon ECS added additional metadata to the task metadata endpoint v4 output. For more information, see Task metadata endpoint version 4 (p. 380) .	5 November 2020

Change	Description	Date
Support for Local Zones and Wavelength Zones	Amazon ECS added support for workloads in Local Zones and Wavelength Zones. For more information, see Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts (p. 664) .	4 September 2020
Amazon ECS variant of Bottlerocket AMI	Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers. An Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI you can use when launching Amazon ECS container instances. For more information, see Amazon ECS-optimized Bottlerocket AMIs (p. 268) .	31 August 2020
Task metadata endpoint version 4 updated for network rate stats	The task metadata endpoint version 4 has been updated to provide network rate stats for Amazon ECS tasks that use the <code>awsvpc</code> or <code>bridge</code> network modes hosted on Amazon EC2 instances running at least version 1.43.0 of the container agent. For more information, see Task metadata endpoint version 4 (p. 380) .	10 August 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics which provide visibility into your accounts usage of Fargate On-Demand and Fargate Spot resources. For more information, see Usage metrics (p. 74) .	3 August 2020
AWS Copilot version 0.1.0	The new AWS Copilot CLI launched, providing high-level commands to simplify modeling, creating, releasing, and managing containerized applications on Amazon ECS from a local development environment. For more information, see Using the AWS Copilot command line interface (p. 35) .	9 July 2020
AWS Fargate platform versions deprecation schedule	The Fargate platform version deprecation schedule has been added. For more information, see AWS Fargate platform version deprecation (p. 81) .	8 July 2020
AWS Fargate Region expansion	Amazon ECS on AWS Fargate has expanded to the Europe (Milan) Region.	25 June 2020
Amazon ECS optimized Amazon Linux 2 (Neuron) AMI released	Amazon ECS released an Amazon ECS optimized Amazon Linux 2 (Neuron) AMI for inferential workloads. For more information, see Amazon ECS-optimized AMI (p. 255) .	24 June 2020
Added support for deleting capacity providers	Amazon ECS added support for deleting Auto Scaling group capacity providers. For more information, see Deleting an Auto Scaling group capacity provider using the classic console (p. 885) .	11 June 2020
AWS Fargate platform version 1.4.0 update	Beginning on May 28, 2020, any new Fargate task that is launched using platform version 1.4.0 will have its 20 GB ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key. For more information, see Fargate task storage (p. 101) .	28 May 2020

Change	Description	Date
Environment variable file support	Added support for specifying environment variable files in a task definition, which enables you to bulk add environment variables to your containers. For more information, see Passing environment variables to a container (p. 197) .	18 May 2020
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Africa (Cape Town) Region.	11 May 2020
Service quota updated	The following service quota was updated: <ul style="list-style-type: none">Clusters per account was raised from 2,000 to 10,000. For more information, see Amazon ECS service quotas (p. 536) .	17 April 2020

Change	Description	Date
AWS Fargate platform version 1.4.0	<p>AWS Fargate platform version 1.4.0 is released, which contains the following features:</p> <ul style="list-style-type: none"> • Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Amazon EFS volumes (p. 102). • The ephemeral task storage has been increased to 20 GB. For more information, see Fargate task storage (p. 101). • The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Fargate Task Networking in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>. • Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC. • CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see Amazon ECS CloudWatch Container Insights (p. 572). • Added support for the task metadata endpoint v4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see Task metadata endpoint version 4 (p. 380). • Added support for the SYS_PTRACE Linux parameter in container definitions. For more information, see Linux parameters (p. 827). • The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. This change should not have an effect on how your tasks run. • The container runtime is now using Containerd instead of Docker. This change should not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime will change from mentioning Docker to more general errors. 	8 April 2020

Change	Description	Date
	For more information, see AWS Fargate platform versions (p. 77) .	
Amazon EFS file system support for task volumes	Amazon EFS file systems can be used as data volumes for both your Amazon ECS and Fargate tasks. For more information, see Amazon EFS volumes (p. 102) .	8 April 2020
Amazon ECS Task Metadata Endpoint version 4	Beginning with Amazon ECS container agent version 1.39.0 and Fargate platform version 1.4.0, an environment variable named <code>ECS_CONTAINER_METADATA_URI_V4</code> is injected into each container in a task. When you query the task metadata version 4 endpoint, various task metadata and Docker stats are available to tasks. For more information, see Task metadata endpoint version 4 (p. 380) .	8 April 2020
Support for specific versions of Secrets Manager secrets to be injected as environment variables	Added support for specifying sensitive data using specific versions of Secrets Manager secrets. For more information, see Retrieve secrets through environment variables (p. 203) .	24 Feb 2020
Added additional CodeDeploy deployment configuration options for blue/green deployments	The CodeDeploy service added new canary and linear deployment configurations for the Amazon ECS deployment type. The ability to define custom deployment configurations is also available. For more information, see Blue/Green deployment with CodeDeploy (p. 477) .	6 Feb 2020
Added the <code>efsVolumeConfiguration</code> task definition parameter	The <code>efsVolumeConfiguration</code> task definition parameter is in public preview, which makes it easier to use Amazon EFS file systems with your Amazon ECS tasks. For more information, see Amazon EFS volumes (p. 102) .	17 Jan 2020
Amazon ECS container agent logging behavior updated	The Amazon ECS container agent logging locations and rotation behavior has been updated. For more information, see Amazon ECS Container Agent Log (p. 781) .	13 Jan 2020
Fargate Spot	Amazon ECS added support for running tasks using Fargate Spot. For more information, see AWS Fargate capacity providers (p. 222) .	3 Dec 2019
Cluster Auto Scaling	Amazon ECS cluster auto scaling enables you to have more control over how you scale tasks within a cluster. For more information, see Amazon ECS cluster auto scaling (p. 226) .	3 Dec 2019
Cluster Capacity Providers	Amazon ECS cluster capacity providers determine the infrastructure to use for your tasks. For more information, see Amazon ECS capacity providers (p. 220) .	3 Dec 2019
Creating a cluster on an AWS Outposts	Amazon ECS now supports creating clusters on an AWS Outposts. For more information, see the section called "Amazon Elastic Container Service on AWS Outposts" (p. 665) .	3 Dec 2019

Change	Description	Date
Service Action Events	Amazon ECS now sends events to Amazon EventBridge when certain service actions occur. For more information, see Service action events (p. 565) .	25 Nov 2019
Amazon ECS GPU-optimized AMI Supports G4 Instances	Amazon ECS added support for the g4 instance type family when using the Amazon ECS GPU-optimized AMI. For more information, see Working with GPUs on Amazon ECS (p. 142) .	8 Oct 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in general availability. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom log routing (p. 166) .	30 Sept 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (Paris), Europe (Stockholm), and Middle East (Bahrain) regions.	30 Sept 2019
Deep Learning Containers with Elastic Inference on Amazon ECS	Amazon ECS supports attaching Amazon Elastic Inference accelerators to your containers to make running deep learning inference workloads more efficient. For more information, see Deep Learning Containers with Elastic Inference on Amazon ECS (p. 669) .	3 Sept 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in public preview. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom log routing (p. 166) .	30 Aug 2019
CloudWatch Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Amazon ECS CloudWatch Container Insights (p. 572) .	30 Aug 2019
Container Level Swap Configuration	Amazon ECS added support for controlling the usage of swap memory space on your Linux container instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For more information, see Managing container swap space (p. 123) .	16 Aug 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Hong Kong) Region.	06 Aug 2019
Elastic Network Interface Trunking	Added additional supported Amazon EC2 instance types for ENI trunking feature. For more information, see Supported Amazon EC2 instance types (p. 287) .	1 Aug 2019
Registering Multiple Target Groups with a Service	Added support for specifying multiple target groups in a service definition. For more information, see Registering multiple target groups with a service (p. 496) .	30 July 2019

Change	Description	Date
Specifying Sensitive Data Using Secrets Manager Secrets	Added tutorial for specifying sensitive data using Secrets Manager secrets. For more information, see Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets (p. 693) .	20 July 2019
CloudWatch Container Insights	Amazon ECS has added support for CloudWatch Container Insights. For more information, see Amazon ECS CloudWatch Container Insights (p. 572) .	9 July 2019
Resource-level permissions for Amazon ECS services and tasksets	Amazon ECS has expanded resource-level permissions support for Amazon ECS services and tasks. For more information, see How Amazon Elastic Container Service works with IAM (p. 592) .	27 June 2019
New Amazon ECS-optimized AMI patched for AWS-2019-005	Amazon ECS has updated the Amazon ECS-optimized AMI to address the vulnerabilities described in AWS-2019-005 .	17 June 2019
Elastic Network Interface Trunking	Amazon ECS introduces support for launching container instances using supported Amazon EC2 instance types that have increased elastic network interface (ENI) density. Using these instance types and opting in to the awsvpcTrunking account setting provides increased ENI density on newly launched container instances which allows you to place more tasks on each container instance. For more information, see Elastic network interface trunking (p. 283) .	6 June 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports the splunk log driver in addition to the awslogs log driver. For more information, see Storage and logging (p. 819) .	1 May 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports referencing sensitive data in the log configuration of a container using the secretOptions container definition parameter. For more information, see Passing sensitive data to a container (p. 200) .	1 May 2019
AWS Fargate platform version 1.3.0 update	Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see Passing sensitive data to a container (p. 200) .	2 Apr 2019
AWS Fargate platform version 1.3.0 update	Beginning on March 27, 2019, any new Fargate task launched can use additional task definition parameters that enable you to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Proxy configuration (p. 835) , Container dependency (p. 830) , and Container timeouts (p. 831) .	27 Mar 2019

Change	Description	Date
Amazon ECS introduces the external deployment type	The <i>external</i> deployment type enables you to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service. For more information, see External deployment (p. 481) .	27 Mar 2019
AWS Deep Learning Containers on Amazon ECS	AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR. For more information, see AWS Deep Learning Containers on Amazon ECS (p. 668) .	27 Mar 2019
Amazon ECS introduces enhanced container dependency management	Amazon ECS introduces additional task definition parameters that enable you to define dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Container dependency (p. 830) .	7 Mar 2019
Amazon ECS introduces the PutAccountSettingDefault API	Amazon ECS introduces the <code>PutAccountSettingDefault</code> API that allows a user to set the default ARN/ID format opt in status for all the users and roles on the account. Previously, setting the account's default opt in status required the use of the account owner. For more information, see Amazon Resource Names (ARNs) and IDs (p. 246) .	8 Feb 2019
Amazon ECS supports GPU workloads	Amazon ECS introduces support for GPU workloads by enabling you to create clusters with GPU-enabled container instances. In a task definition you can specify the number of required GPUs and the ECS agent will pin the physical GPUs to the container. For more information, see Working with GPUs on Amazon ECS (p. 142) .	4 Feb 2019
Amazon ECS expanded secrets support	Amazon ECS expanded support for using AWS Secrets Manager secrets directly in your task definitions to inject sensitive data into your containers. For more information, see Passing sensitive data to a container (p. 200) .	21 Jan 2019
Interface VPC Endpoints (AWS PrivateLink)	Added support for configuring interface VPC endpoints powered by AWS PrivateLink. This allows you to create a private connection between your VPC and Amazon ECS without requiring access over the Internet, through a NAT instance, a VPN connection, or AWS Direct Connect. For more information, see Interface VPC Endpoints (AWS PrivateLink) .	26 Dec 2018

Change	Description	Date
AWS Fargate platform version 1.3.0	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> • Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. <p>For more information, see Passing sensitive data to a container (p. 200).</p> <ul style="list-style-type: none"> • Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. <p>For more information, see Task maintenance in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>.</p> <p>For more information, see AWS Fargate platform versions (p. 77).</p>	17 Dec 2018
Service limits updated	<p>The following service limits were updated:</p> <ul style="list-style-type: none"> • Number of clusters per Region, per account was raised from 1000 to 2000. • Number of container instances per cluster was raised from 1000 to 2000. • Number of services per cluster was raised from 500 to 1000. <p>For more information, see Amazon ECS service quotas (p. 536).</p>	14 Dec 2018
AWS Fargate region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Mumbai) and Canada (Central) Regions.</p> <p>For more information, see AWS Fargate platform versions (p. 77).</p>	07 Dec 2018
Amazon ECS blue/green deployments	<p>Amazon ECS added support for blue/green deployments using CodeDeploy. This deployment type allows you to verify a new deployment of a service before sending production traffic to it.</p> <p>For more information, see Blue/Green deployment with CodeDeploy (p. 477).</p>	27 Nov 2018
Amazon ECS-optimized Amazon Linux 2 (arm64) AMI released	<p>Amazon ECS released an Amazon ECS-optimized Amazon Linux 2 AMIs for arm64 architecture.</p> <p>For more information, see Amazon ECS-optimized AMI (p. 255).</p>	26 Nov 2018

Change	Description	Date
Added support for additional Docker flags in task definitions	<p>Amazon ECS introduced support for the following Docker flags in task definitions:</p> <ul style="list-style-type: none"> • IPC mode (p. 842) • PID mode (p. 842) 	16 Nov 2018
Amazon ECS secrets support	<p>Amazon ECS added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers.</p> <p>For more information, see Passing sensitive data to a container (p. 200).</p>	15 Nov 2018
Resource tagging	<p>Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances.</p> <p>For more information, see Resources and tags (p. 529).</p>	15 Nov 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions.</p> <p>For more information, see Amazon ECS on AWS Fargate (p. 67).</p>	07 Nov 2018
Service limits updated	<p>The following service limits were updated:</p> <ul style="list-style-type: none"> • 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. <p>For more information, see Amazon ECS service quotas (p. 536).</p>	31 Oct 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Europe (London) Region.</p> <p>For more information, see Amazon ECS on AWS Fargate (p. 67).</p>	26 Oct 2018
Amazon ECS-optimized Amazon Linux 2 AMI Released	<p>Amazon ECS vends Linux AMIs that are optimized for the service in two variants. The latest and recommended version is based on x; Amazon ECS also vends AMIs that are based on the Amazon Linux AMI, but we recommend that you migrate your workloads to the Amazon Linux 2 variant, as support for the Amazon Linux AMI will end no later than June 30, 2020.</p> <p>For more information, see Amazon ECS-optimized AMI (p. 255).</p>	18 October 2018

Change	Description	Date
Amazon ECS Task Metadata Endpoint version 3	<p>Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called <code>ECS_CONTAINER_METADATA_URI</code> into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks that use the <code>awsvpc</code> 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. 380).</p>	18 October 2018
Amazon ECS service discovery Region expansion	<p>Amazon ECS service discovery has expanded support to the Canada (Central), South America (São Paulo), Asia Pacific (Seoul), Asia Pacific (Mumbai), and Europe (Paris) Regions.</p> <p>For more information, see Service discovery (p. 522).</p>	27 September 2018
Added support for additional Docker flags in container definitions	<p>Amazon ECS introduced support for the following Docker flags in container definitions:</p> <ul style="list-style-type: none"> • System controls (p. 832) • Interactive (p. 833) • Pseudo terminal (p. 834) 	17 Sept 2018
Private registry authentication support for Amazon ECS using AWS Fargate tasks	<p>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.</p> <p>For more information, see Private registry authentication for tasks (p. 195).</p>	10 Sept 2018
Amazon ECS service discovery Region expansion	<p>Amazon ECS service discovery has expanded support to the Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), EU (Frankfurt), and Europe (London) Regions.</p> <p>For more information, see Service discovery (p. 522).</p>	30 August 2018
Scheduled tasks with Fargate tasks support	<p>Amazon ECS introduced support for scheduled tasks for the Fargate launch type.</p> <p>For more information, see Scheduled tasks (p. 445).</p>	28 August 2018
Private registry authentication using AWS Secrets Manager support	<p>Amazon ECS introduced support for 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.</p> <p>For more information, see Private registry authentication for tasks (p. 195).</p>	16 August 2018

Change	Description	Date
Docker volume support added	<p>Amazon ECS introduced support for Docker volumes.</p> <p>For more information, see Using data volumes in tasks (p. 100).</p>	9 August 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Europe (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) Regions.</p> <p>For more information, see Amazon ECS on AWS Fargate (p. 67).</p>	19 July 2018
Amazon ECS service scheduler strategies added	<p>Amazon ECS introduced the concept of service scheduler strategies.</p> <p>There are two service scheduler strategies available:</p> <ul style="list-style-type: none"> • 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. 455). • 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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 453). <p>Note Fargate tasks do not support the DAEMON scheduling strategy.</p> <p>For more information, see Service scheduler concepts (p. 453).</p>	12 June 2018

Change	Description	Date
Amazon ECS container agent v1.18.0	<p>New version of the Amazon ECS container agent released, which added the following functionality:</p> <ul style="list-style-type: none"> • Added procedure to manually install the container agent from a S3 URL on non-Amazon Linux EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Installing the Amazon ECS container agent (p. 356). • Added procedure to manually install the container agent from a S3 URL on a Windows EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Getting started with Windows containers using the classic console (p. 875). • Added support for customizing the container agent image pull behavior using the <code>ECS_IMAGE_PULL_BEHAVIOR</code> parameter. For more information, see Amazon ECS container agent configuration (p. 370). <p>For more information, see amazon-ecs-agent github.</p>	24 May 2018
Added Support for bridge and host Network Modes When Configuring Service Discovery	Added support for configuring service discovery for Amazon ECS services using task definitions that specify the bridge or host network modes. For more information, see Service discovery (p. 522) .	22 May 2018
Added support for additional Amazon ECS-optimized AMI metadata parameters	Added subparameters that allow you to programmatically retrieve the Amazon ECS-optimized AMI ID, image name, operating system, container agent version, and runtime version. Query the metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI metadata (p. 259) .	9 May 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the US East (Ohio), US West (Oregon), and EU West (Ireland) Regions.</p> <p>For more information, see Amazon ECS on AWS Fargate (p. 67).</p>	26 April 2018
Amazon ECS-optimized AMI Metadata Retrieval	Added ability to programmatically retrieve Amazon ECS-optimized AMI metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI metadata (p. 259) .	10 April 2018

Change	Description	Date
AWS Fargate platform version	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> • Added support for Amazon ECS task metadata endpoint (p. 380). • Added support for Health check (p. 811). • Added support for Service discovery (p. 522) <p>For more information, see AWS Fargate platform versions (p. 77).</p>	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. 522) .	22 March 2018
Docker shm-size and tmpfs support	<p>Added support for the Docker shm-size and tmpfs parameters in Amazon ECS task definitions.</p> <p>For more information about the updated ECS CLI syntax, see Linux parameters (p. 827).</p>	20 March 2018
Container Health Checks	Added support for Docker health checks in container definitions. For more information, see Health check (p. 811) .	08 March 2018
AWS Fargate	Added overview for Amazon ECS with AWS Fargate. For more information, see Amazon ECS on AWS Fargate (p. 67) .	22 February 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. 380) .	8 February 2018
Amazon ECS Service Auto Scaling using target tracking policies	<p>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. 501).</p> <p>Removed the previous tutorial for step scaling in the ECS first run wizard. This was replaced with the new tutorial for target tracking.</p>	8 February 2018
Docker 17.09 support	Added support for Docker 17.09. For more information, see Amazon ECS-optimized AMI (p. 255) .	18 January 2018
Elastic Load Balancing health check initialization wait period	Added ability to specify a wait period for health checks.	27 December 2017
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. 455) .	11 January 2018

Change	Description	Date
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
Task execution role	<p>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:</p> <ul style="list-style-type: none"> • Calls to Amazon ECR to pull the container image • Calls to CloudWatch to store container application logs <p>For more information, see Amazon ECS task execution IAM role (p. 626).</p>	7 December 2017
Windows containers support GA	Added support for Windows Server 2016 containers. For more information, see Amazon EC2 Windows containers (p. 363) .	5 December 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. 85) .	29 November 2017
Amazon ECS name change	Amazon Elastic Container Service is renamed (previously Amazon EC2 Container Service).	21 November 2017
Task networking	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. For more information, see Task networking for tasks that are hosted on Amazon EC2 instances (p. 90) .	14 November 2017
Amazon ECS container metadata	Amazon ECS containers are now able to access metadata such as their Docker container or image ID, networking configuration, or Amazon ARNs. For more information, see Amazon ECS container metadata file (p. 376) .	2 November 2017
Docker 17.06 support	Added support for Docker 17.06. For more information, see Amazon ECS-optimized AMI (p. 255) .	2 November 2017
Support for Docker flags: device and init	Added support for Docker's device and init features in task definitions using the <code>LinuxParameters</code> parameter (<code>devices</code> and <code>initProcessEnabled</code>). For more information, see LinuxParameters .	2 November 2017

Change	Description	Date
Support for Docker flags: cap-add and cap-drop	Added support for Docker's cap-add and cap-drop features in task definitions using the <code>LinuxParameters</code> parameter (<code>capabilities</code>). For more information, see LinuxParameters .	22 September 2017
Network Load Balancer support	Amazon ECS added support for Network Load Balancers in the Amazon ECS console. For more information, see Creating a Network Load Balancer (p. 493) .	7 September 2017
RunTask overrides	Added support for task definition overrides when running a task. This allows you to run a task while changing a task definition without the need to create a new task definition revision. For more information, see Run a standalone task in the classic Amazon ECS console (p. 896) .	27 June 2017
Amazon ECS scheduled tasks	Added support for scheduling tasks using cron. For more information, see Scheduled tasks (p. 445) .	7 June 2017
Spot Instances in the Amazon ECS console	Added support for creating Spot Fleet container instances within the Amazon ECS console. For more information, see Launching an Amazon ECS Linux container instance (p. 272) .	6 June 2017
Amazon SNS notification for new Amazon ECS-optimized AMI releases	Added ability to subscribe to SNS notifications about new Amazon ECS-optimized AMI releases.	23 March 2017
Microservices and batch jobs	Added documentation for two common use cases for Amazon ECS: microservices and batch jobs. For more information, see Common use cases in Amazon ECS (p. 5) .	February 2017
Container instance draining	Added support for container instance draining, which provides a method for removing container instances from a cluster. For more information, see Container instance draining (p. 354) .	24 January 2017
Docker 1.12 support	Added support for Docker 1.12. For more information, see Amazon ECS-optimized AMI (p. 255) .	24 January 2017
New task placement strategies	Added support for task placement strategies: attribute-based placement, bin pack, Availability Zone spread, and one per host. For more information, see Amazon ECS task placement strategies (p. 435) .	29 December 2016
Windows container support in beta	Added support for Windows Server 2016 containers (beta). For more information, see Amazon EC2 Windows containers (p. 363) .	20 December 2016
Blox OSS support	Added support for Blox OSS, which allows for custom task schedulers. For more information, see Scheduling Amazon ECS tasks (p. 429) .	1 December 2016
Amazon ECS event stream for CloudWatch Events	Amazon ECS now sends container instance and task state changes to CloudWatch Events. For more information, see Amazon ECS events and EventBridge (p. 558) .	21 November 2016

Change	Description	Date
Amazon ECS container logging to CloudWatch Logs	Added support for the awslogs driver to send container log streams to CloudWatch Logs. For more information, see Using the awslogs log driver (p. 161) .	12 September 2016
Amazon ECS services with Elastic Load Balancing support for dynamic ports	Added support for a load balancer to support multiple instance:port combinations per listener, which increases flexibility for containers. Now you can let Docker dynamically define the container's host port and the ECS scheduler registers the instance:port with the load balancer. For more information, see Service load balancing (p. 486) .	11 August 2016
IAM roles for Amazon ECS tasks	Added support for associating IAM roles with a task. This provides finer-grained permissions to containers as opposed to a single role for an entire container instance. For more information, see Task IAM role (p. 631) .	13 July 2016
Docker 1.11 support	Added support for Docker 1.11. For more information, see Amazon ECS-optimized AMI (p. 255) .	31 May 2016
Task automatic scaling	Amazon ECS added support for automatically scaling your tasks run by a service. For more information, see Service auto scaling (p. 498) .	18 May 2016
Task definition filtering on task family	Added support for filtering a list of task definition based on the task definition family. For more information, see ListTaskDefinitions .	17 May 2016
Docker container and Amazon ECS agent logging	Amazon ECS added ability to send ECS agent and Docker container logs from container instances to CloudWatch Logs to simplify troubleshooting issues.	5 May 2016
ECS-optimized AMI now supports Amazon Linux 2016.03.	The ECS-optimized AMI added support for Amazon Linux 2016.03. For more information, see Amazon ECS-optimized AMI (p. 255) .	5 April 2016
Docker 1.9 support	Added support for Docker 1.9. For more information, see Amazon ECS-optimized AMI (p. 255) .	22 December 2015
CloudWatch metrics for cluster CPU and memory reservation	Amazon ECS added custom CloudWatch metrics for CPU and memory reservation.	22 December 2015
New Amazon ECS first-run experience	The Amazon ECS console first-run experience added zero-click role creation.	23 November 2015
Task placement across Availability Zones	The Amazon ECS service scheduler added support for task placement across Availability Zones.	8 October 2015
CloudWatch metrics for Amazon ECS clusters and services	Amazon ECS added custom CloudWatch metrics for CPU and memory utilization for each container instance, service, and task definition family in a cluster. These new metrics can be used to scale container instances in a cluster using Auto Scaling groups or to create custom CloudWatch alarms.	17 August 2015

Change	Description	Date
UDP port support	Added support for UDP ports in task definitions.	7 July 2015
Environment variable overrides	Added support for <code>deregisterTaskDefinition</code> and environment variable overrides for <code>runTask</code> .	18 June 2015
Automated Amazon ECS agent updates	Added ability to see the ECS agent version that is running on a container instance. Also able to update the ECS agent from the AWS Management Console, AWS CLI, and SDK.	11 June 2015
Amazon ECS service scheduler and Elastic Load Balancing integration	Added ability to define a service and associate that service with an Elastic Load Balancing load balancer.	9 April 2015
Amazon ECS GA	Amazon ECS general availability in the US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland) Regions.	9 April 2015

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.