AWS Batch: User Guide

Copyright © 2024 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 AWS Batch?** ........................................................................................................... 1
  Components of AWS Batch ................................................................. 1
    Jobs ........................................................................................................ 1
  Job Definitions ....................................................................................... 2
  Job Queues ............................................................................................ 2
  Compute Environment ........................................................................... 2

**Getting Started** .................................................................................................................. 2

**Dashboard** ......................................................................................................................... 3
  Single job queue .................................................................................... 3
  CloudWatch Container Insights .............................................................. 4
  Job logs ..................................................................................................... 4

**Setting Up** ............................................................................................................................. 5
  Sign up for an AWS account ................................................................. 5
  Create an administrative user ............................................................... 6
  Create IAM roles for your compute environments and container instances ... 7
  Create a key pair ................................................................................... 7
  Create a VPC ........................................................................................ 9
  Create a security group ......................................................................... 10
  Install the AWS CLI ........................................................................... 12

**Getting Started** .................................................................................................................. 13
  Prerequisites .......................................................................................... 13
  Getting Started - Amazon EC2 .............................................................. 13
    Create a compute environment .......................................................... 13
    Create a job queue ............................................................................. 18
    Create a job definition ....................................................................... 18
    Create a job ........................................................................................ 22
  Review and create ................................................................................ 23

Getting Started - Fargate ............................................................................. 23
  Create a compute environment .............................................................. 23
  Create a job queue ............................................................................. 24
  Create a job definition ....................................................................... 25
  Create a job ........................................................................................ 28
  Review and create ................................................................................ 28

AWS Batch on Amazon EKS ......................................................................................... 28
Prerequisites .............................................................................................................. 29
Step 1: Preparing your Amazon EKS cluster for AWS Batch .................................. 31
Step 2: Creating an Amazon EKS compute environment ........................................ 34
Step 3: Create a job queue and attach the compute environment .............................. 36
Step 4: Create a job definition .................................................................................. 37
Step 5: Submit a job .................................................................................................. 38
(Optional) Submit a job with overrides .................................................................... 38
AWS Batch on Amazon EKS Private Clusters .......................................................... 39

Jobs .............................................................................................................................. 52
Submitting a job ......................................................................................................... 52
Job states ...................................................................................................................... 55
Job Environment Variables ....................................................................................... 57
Automated job retries ................................................................................................. 59
Job dependencies ........................................................................................................ 60
Job timeouts ................................................................................................................ 61
Amazon EKS jobs ........................................................................................................ 62
  Map a running job to a pod and a node .................................................................... 63
  How to map a running pod back to its job ............................................................... 64
Array jobs .................................................................................................................... 65
  Example array job workflow .................................................................................... 68
  Tutorial: Using array job index ................................................................................ 71
Multi-node parallel jobs .............................................................................................. 77
  Environment variables ............................................................................................... 78
  Node groups .............................................................................................................. 79
  Job lifecycle .............................................................................................................. 79
  Compute environment considerations ........................................................................ 80
GPU jobs ....................................................................................................................... 81
  To create a GPU-based job on Amazon EKS resources ............................................ 83
    To create GPU-based Kubernetes cluster on Amazon EKS ........................................ 83
    To create an Amazon EKS GPU job definition ....................................................... 85
    To run a GPU job in your Amazon EKS cluster ....................................................... 86
Search and filter AWS Batch jobs ................................................................................ 86
  Job logs .................................................................................................................... 87
  Job information ....................................................................................................... 89

Job definitions ............................................................................................................ 90
  Creating a single-node job definition ........................................................................ 91
Creating a single-node job definition on Amazon EC2 resources ........................................ 91
Creating a single-node job definition on AWS Fargate resources ........................................ 96
Creating a single-node job definition on Amazon EKS resources ........................................ 102
Creating a multi-node parallel job definition ............................................................................. 106
  Creating a multi-node parallel job definition on Amazon EC2 resources ................................. 107
Creating job definitions using ContainerProperties .............................................................. 113
  Job definition parameters for ContainerProperties .............................................................. 121
Creating job definitions using EcsProperties ............................................................................ 164
  ContainerProperties versus EcsProperties job definitions .................................................... 165
General changes to the AWS Batch APIs .................................................................................... 166
Multi-container job definitions for Amazon ECS ......................................................................... 166
Multi-container job definitions for Amazon EKS ......................................................................... 167
AWS Batch job scenarios using EcsProperties ............................................................................ 168
Using the awslogs log driver ....................................................................................................... 174
  Available awslogs log driver options ..................................................................................... 175
  Specifying a log configuration in your job definition .............................................................. 177
Specifying sensitive data ........................................................................................................... 178
  Using Secrets Manager ........................................................................................................... 179
  Using Systems Manager Parameter Store ............................................................................... 186
Private registry authentication for jobs ...................................................................................... 190
  Required IAM permissions for private registry authentication .............................................. 192
  Using private registry authentication ..................................................................................... 192
Amazon EFS volumes ................................................................................................................ 193
  Amazon EFS volume considerations ....................................................................................... 194
  Using Amazon EFS access points ........................................................................................... 195
  Specifying an Amazon EFS file system in your job definition .............................................. 195
Example job definitions ............................................................................................................ 198
  Use environment variables ...................................................................................................... 198
  Using parameter substitution .................................................................................................. 199
  Test GPU functionality .......................................................................................................... 200
  Multi-node parallel job .......................................................................................................... 201

**Job queues** ........................................................................................................................... 203
Creating a job queue .................................................................................................................. 203
  Creating a Fargate job queue .................................................................................................. 203
Creating an Amazon EC2 job queue .......................................................................................... 204
Creating an Amazon EKS job queue ......................................................................................... 205
Job scheduling ........................................................................................................ 206
Job queue template ........................................................................................................ 206
Job queue parameters ..................................................................................................... 207
Job queue name ............................................................................................................. 208
Job queue state time limit actions .................................................................................. 208
Priority ........................................................................................................................ 208
Scheduling policy ........................................................................................................... 209
State .............................................................................................................................. 209
Compute environment order ............................................................................................ 210
Tags ............................................................................................................................... 210

Job scheduling .................................................................................................................. 212
Share identifiers .............................................................................................................. 212
Fair share scheduling ...................................................................................................... 213

Compute environment ...................................................................................................... 215
Managed compute environments ................................................................................... 215
  Consideration when creating multi-node parallel jobs .................................................. 217
Unmanaged compute environments ................................................................................ 218
Compute resource AMIs .................................................................................................. 219
  Compute resource AMI specification ........................................................................... 220
  Creating a compute resource AMI .............................................................................. 222
  Using a GPU workload AMI ...................................................................................... 225
Amazon Linux deprecation .............................................................................................. 231
Launch template support .................................................................................................. 232
  Amazon EC2 user data in launch templates ................................................................ 233
Creating a compute environment .................................................................................... 237
  To create a managed compute environment using AWS Fargate resources .............. 238
  To create a managed compute environment using EC2 resources ............................ 240
  To create an unmanaged compute environment using EC2 resources ....................... 245
  To create a managed compute environment using Amazon EKS resources ......... 246
Compute environment template ....................................................................................... 249
Compute environment parameters .................................................................................. 251
  Compute environment name ....................................................................................... 251
Type ............................................................................................................................... 252
State ............................................................................................................................... 252
Compute resources ......................................................................................................... 253
Amazon EKS configuration ............................................................................................. 265
Service role ...................................................................................................................... 266
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tags</td>
<td>267</td>
</tr>
<tr>
<td>EC2 Configurations</td>
<td>267</td>
</tr>
<tr>
<td>Allocation strategies</td>
<td>268</td>
</tr>
<tr>
<td>Updating compute environments</td>
<td>269</td>
</tr>
<tr>
<td>Updating the AMI ID</td>
<td>272</td>
</tr>
<tr>
<td>Amazon EKS compute environments</td>
<td>273</td>
</tr>
<tr>
<td>Default AMI selection</td>
<td>274</td>
</tr>
<tr>
<td>Supported Kubernetes versions</td>
<td>275</td>
</tr>
<tr>
<td>Updating Kubernetes version of compute environment</td>
<td>276</td>
</tr>
<tr>
<td>Shared responsibility of the Kubernetes nodes</td>
<td>276</td>
</tr>
<tr>
<td>Running a DaemonSet on AWS Batch managed nodes</td>
<td>277</td>
</tr>
<tr>
<td>Customizing with launch templates</td>
<td>277</td>
</tr>
<tr>
<td>Memory Management</td>
<td>281</td>
</tr>
<tr>
<td>Reserving System Memory</td>
<td>282</td>
</tr>
<tr>
<td>Viewing Compute Resource Memory</td>
<td>282</td>
</tr>
<tr>
<td>Memory and vCPU considerations for AWS Batch on Amazon EKS</td>
<td>283</td>
</tr>
<tr>
<td>Scheduling policies</td>
<td>289</td>
</tr>
<tr>
<td>Creating a scheduling policy</td>
<td>289</td>
</tr>
<tr>
<td>Scheduling policy template</td>
<td>290</td>
</tr>
<tr>
<td>Scheduling policy parameters</td>
<td>291</td>
</tr>
<tr>
<td>Scheduling policy name</td>
<td>292</td>
</tr>
<tr>
<td>Fair share policy</td>
<td>292</td>
</tr>
<tr>
<td>Tags</td>
<td>294</td>
</tr>
<tr>
<td>Orchestrate AWS Batch jobs</td>
<td>295</td>
</tr>
<tr>
<td>Viewing state machine details</td>
<td>295</td>
</tr>
<tr>
<td>Editing a state machine</td>
<td>296</td>
</tr>
<tr>
<td>Running a state machine</td>
<td>296</td>
</tr>
<tr>
<td>AWS Batch on AWS Fargate</td>
<td>297</td>
</tr>
<tr>
<td>When to use Fargate</td>
<td>297</td>
</tr>
<tr>
<td>Job definitions on Fargate</td>
<td>298</td>
</tr>
<tr>
<td>Job queues on Fargate</td>
<td>300</td>
</tr>
<tr>
<td>Compute environments on Fargate</td>
<td>300</td>
</tr>
<tr>
<td>AWS Batch on Amazon EKS</td>
<td>302</td>
</tr>
<tr>
<td>Elastic Fabric Adapter</td>
<td>305</td>
</tr>
<tr>
<td>IAM policies, roles, and permissions</td>
<td>308</td>
</tr>
<tr>
<td>Policy structure</td>
<td>308</td>
</tr>
</tbody>
</table>
Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events ........................................ 359
Prerequisites ......................................................................................................................................................... 359
Step 1: Create and Subscribe to an Amazon SNS Topic ....................................................................................... 359
Step 2: Register Event Rule ................................................................................................................................. 360
Step 3: Test Your Rule ........................................................................................................................................ 362
Alternate rule: Batch Job Queue Blocked ........................................................................................................... 362

CloudWatch Logs ............................................................................................................................................... 364
Add a CloudWatch Logs IAM policy ....................................................................................................................... 364
Install and configure the CloudWatch agent ........................................................................................................... 366
View CloudWatch Logs ......................................................................................................................................... 366

Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs ................................................................. 369
Prerequisites ......................................................................................................................................................... 369
Install AWS for Fluent Bit ...................................................................................................................................... 369
Turn on Fluent Bit for AWS Batch nodes .............................................................................................................. 369

CloudWatch Container Insights ............................................................................................................................. 371
Turn on Container Insights ...................................................................................................................................... 371

CloudTrail .............................................................................................................................................................. 373
AWS Batch Information in CloudTrail ................................................................................................................... 373
Understanding AWS Batch Log File Entries .......................................................................................................... 374

Creating a virtual private cloud ............................................................................................................................ 377
Create a VPC ........................................................................................................................................................ 377
Next Steps ............................................................................................................................................................ 378

Security ................................................................................................................................................................. 379
Identity and Access Management ........................................................................................................................... 379
Audience ............................................................................................................................................................... 380
Authenticating with identities ................................................................................................................................. 381
Managing access using policies ............................................................................................................................. 384
How AWS Batch works with IAM ......................................................................................................................... 386
Execution IAM role ............................................................................................................................................. 393
Identity-based policy examples ............................................................................................................................... 395
Cross-service confused deputy prevention ........................................................................................................... 398
Troubleshooting ................................................................................................................................................ 400
Using Service-Linked Roles ................................................................................................................................ 402
AWS managed policies ...................................................................................................................................... 409
VPC endpoints ..................................................................................................................................................... 424
Considerations .................................................................................................................................................... 424
### AWS Batch User Guide

**Common errors and troubleshooting**

**Tagging your resources**

- Tag basics .................................................................................................................. 430
- Tagging your resources ............................................................................................ 431
- Tag restrictions ......................................................................................................... 432
- Working with tags using the console ........................................................................... 433
  - Adding tags on an individual resource on creation .................................................. 433
  - Adding and deleting tags on an individual resource .................................................. 433
- Working with tags using the CLI or API ........................................................................ 433

**Service Quotas** ............................................................................................................. 436

**Troubleshooting** ............................................................................................................ 437

- **AWS Batch** ............................................................................................................. 438
  - INVALID compute environment .................................................................................. 438
  - Jobs stuck in a RUNNABLE status .............................................................................. 440
  - Spot Instances not tagged on creation ...................................................................... 445
  - Spot Instances not scaling down ................................................................................ 446
  - Can't retrieve Secrets Manager secrets ...................................................................... 447
  - Can't override job definition resource requirements .................................................. 447
  - Error message when you update the desiredvCpus setting ........................................ 449
- **AWS Batch on Amazon EKS** .................................................................................... 449
  - INVALID compute environment .................................................................................. 449
  - AWS Batch on Amazon EKS job is stuck in RUNNABLE status .................................. 453
  - Verify that the aws-auth ConfigMap is configured correctly .................................... 454
  - RBAC permissions or bindings aren't configured properly ........................................ 454

**Best Practices** ................................................................................................................ 457

- When to use AWS Batch ............................................................................................... 457
- Checklist to run at scale ............................................................................................... 458
- Optimize containers and AMIs ..................................................................................... 459
- Choose the right compute environment resource ....................................................... 460
- Amazon EC2 On-Demand or Amazon EC2 Spot .......................................................... 460
- Use Amazon EC2 Spot best practices for AWS Batch .................................................. 461
- Common errors and troubleshooting .............................................................................. 463

**Document history** ......................................................................................................... 466
What Is AWS Batch?

AWS Batch helps you to run batch computing workloads on the AWS Cloud. Batch computing is a common way for developers, scientists, and engineers to access large amounts of compute resources. AWS Batch removes the undifferentiated heavy lifting of configuring and managing the required infrastructure, similar to traditional batch computing software. This service can efficiently provision resources in response to jobs submitted in order to eliminate capacity constraints, reduce compute costs, and deliver results quickly.

As a fully managed service, AWS Batch helps you to run batch computing workloads of any scale. AWS Batch automatically provisions compute resources and optimizes the workload distribution based on the quantity and scale of the workloads. With AWS Batch, there’s no need to install or manage batch computing software, so you can focus your time on analyzing results and solving problems.

Topics

- Components of AWS Batch
- Getting Started
- Dashboard

Components of AWS Batch

AWS Batch simplifies running batch jobs across multiple Availability Zones within a Region. You can create AWS Batch compute environments within a new or existing VPC. After a compute environment is up and associated with a job queue, you can define job definitions that specify which Docker container images to run your jobs. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.

Jobs

A unit of work (such as a shell script, a Linux executable, or a Docker container image) that you submit to AWS Batch. It has a name, and runs as a containerized application on AWS Fargate or Amazon EC2 resources in your compute environment, using parameters that you specify in a job definition. Jobs can reference other jobs by name or by ID, and can be dependent on the successful completion of other jobs. For more information, see Jobs.
**Job Definitions**

A job definition specifies how jobs are to be run. You can think of a job definition as a blueprint for the resources in your job. You can supply your job with an IAM role to provide access to other AWS resources. You also specify both memory and CPU requirements. The job definition can also control container properties, environment variables, and mount points for persistent storage. Many of the specifications in a job definition can be overridden by specifying new values when submitting individual Jobs. For more information, see [Job definitions](#).

**Job Queues**

When you submit an AWS Batch job, you submit it to a particular job queue, where the job resides until it's scheduled onto a compute environment. You associate one or more compute environments with a job queue. You can also assign priority values for these compute environments and even across job queues themselves. For example, you can have a high priority queue that you submit time-sensitive jobs to, and a low priority queue for jobs that can run anytime when compute resources are cheaper.

**Compute Environment**

A compute environment is a set of managed or unmanaged compute resources that are used to run jobs. With managed compute environments, you can specify desired compute type (Fargate or EC2) at several levels of detail. You can set up compute environments that use a particular type of EC2 instance, a particular model such as `c5.2xlarge` or `m5.10xlarge`. Or, you can choose only to specify that you want to use the newest instance types. You can also specify the minimum, desired, and maximum number of vCPUs for the environment, along with the amount that you're willing to pay for a Spot Instance as a percentage of the On-Demand Instance price and a target set of VPC subnets. AWS Batch efficiently launches, manages, and terminates compute types as needed. You can also manage your own compute environments. As such, you're responsible for setting up and scaling the instances in an Amazon ECS cluster that AWS Batch creates for you. For more information, see [Compute environment](#).

**Getting Started**

Get started with AWS Batch by creating a job definition, compute environment, and a job queue in the AWS Batch console.
The AWS Batch first-run wizard gives you the option of creating a compute environment and a job queue and submitting a sample Hello World job. If you already have a Docker image you want to launch in AWS Batch, you can create a job definition with that image and submit that to your queue instead. For more information, see Getting Started with AWS Batch.

Dashboard

On the AWS Batch dashboard, you can monitor recent jobs, job queues, and compute environments. By default, the following dashboard widgets are displayed:

- **Job overview** – For more information about AWS Batch jobs, see Jobs.
- **Job queue overview** – For more information about AWS Batch job queues, see Job queues.
- **Compute environment overview** – For more information about AWS Batch compute environments, see Compute environment.

You can customize the widgets that are displayed on the Dashboard page. The following sections describe additional widgets that you can install.

Single job queue

This widget displays detailed information about a single job queue.

To add this widget, follow these steps.

1. Open the AWS Batch console.
2. From the navigation bar, select the AWS Region that you want.
3. In the navigation pane, choose Dashboard.
4. Choose Add widgets.
5. For Single job queue, choose Add widget.
6. For Job queue, select the job queue that you want.
7. For Job status, choose the job statuses that you want to display.
8. (Optional) Turn off Show connected compute environments if you don't want to display the properties for compute environments.
9. For Compute environment properties, select the properties that you want.
10. Choose Add.
CloudWatch Container Insights

This widget displays aggregated metrics for AWS Batch compute environments and jobs. For more information about Container Insights, see CloudWatch Container Insights.

To add this widget, follow these steps.

1. Open the AWS Batch console.
2. From the navigation bar, select the AWS Region that you want.
3. In the navigation pane, choose Dashboard.
4. Choose Add widgets.
5. For Container insights, choose Add widget.
6. For Compute environment, choose the compute environment that you want.
7. Choose Add.

Job logs

This widget displays different logs from your jobs in one convenient location. For more information about job logs, see the section called “Job logs”.

To add this widget, follow these steps.

1. Open the AWS Batch console.
2. From the navigation bar, select the AWS Region that you want.
3. In the navigation pane, choose Dashboard.
4. Choose Add widgets.
5. For Job logs, choose Add widget.
6. For Job id, enter the job ID for the job that you want.
7. Choose Add.
Setting up with AWS Batch

If you've already signed up for Amazon Web Services (AWS) and are using Amazon Elastic Compute Cloud (Amazon EC2) or Amazon Elastic Container Service (Amazon ECS), you can soon use AWS Batch. The setup process for these services is similar. This is because AWS Batch uses Amazon ECS container instances in its compute environments. To use the AWS CLI with AWS Batch, you must use a version of the AWS CLI that supports the latest AWS Batch features. If you don't see support for an AWS Batch feature in the AWS CLI, upgrade to the latest version. For more information, see http://aws.amazon.com/cli/.

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

Complete the following tasks to get set up for AWS Batch. If you already completed any of these steps, you can skip directly to installing the AWS CLI.

Topics

• Sign up for an AWS account
• Create an administrative user
• Create IAM roles for your compute environments and container instances
• Create a key pair
• Create a VPC
• Create a security group
• Install the AWS CLI

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

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, secure your AWS account root user, enable AWS IAM Identity Center, and 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

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the AWS IAM Identity Center User Guide.

2. In IAM Identity Center, grant administrative access to an administrative user.
For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the AWS IAM Identity Center 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 IAM roles for your compute environments and container instances**

Your AWS Batch compute environments and container instances require AWS account credentials to make calls to other AWS APIs on your behalf. Create an IAM role that provides these credentials to your compute environments and container instances, then associate that role with your compute environments.

**Note**

The AWS Batch compute environment and container instance roles are automatically created for you in the console first-run experience. So, if you intend to use the AWS Batch console, you can move ahead to the next section. If you plan to use the AWS CLI instead, complete the procedures in Using service-linked roles for AWS Batch and Amazon ECS instance role before creating your first compute environment.

**Create a key pair**

AWS uses public-key cryptography to secure the login information for your instance. A Linux instance, such as an AWS Batch compute environment 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 create your compute environment, then provide the private key when you log in using SSH.
If you didn't create a key pair already, you can create one using the Amazon EC2 console. Note that, if you plan to launch instances in multiple AWS Regions, create a key pair in each Region. For more information about Regions, see Regions and Availability Zones in the Amazon EC2 User Guide for Linux Instances.

To create a key pair

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select an AWS Region for the key pair. You can select any Region that's available to you, regardless of your location: however, key pairs are specific to a Region. For example, if you plan to launch an instance in the US West (Oregon) Region, create a key pair for the instance in the same Region.
3. In the navigation pane, choose Key Pairs, Create Key Pair.
4. In the Create Key Pair dialog box, for Key pair name, enter a name for the new key pair, and choose Create. Choose a name that you can remember, such as your user name, followed by -key-pair, plus the Region name. For example, me-key-pair-uswest2.
5. The private key file is automatically downloaded by your browser. The base file name is the name that you specified as the name of your key pair, and the file name extension is .pem. Save the private key file in a safe place.

⚠️ Important

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

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

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

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

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

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


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

3. Under **Type of key to generate**, choose **RSA**. If you're using an earlier version of PuTTYgen, choose **SSH-2 RSA**.

4. Choose **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, choose the option to display files of all types.

5. Select the private key file that you created in the previous procedure and choose **Open**. Choose **OK** to dismiss the confirmation dialog box.

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

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

Create a VPC

With Amazon Virtual Private Cloud (Amazon VPC), you can launch AWS resources into a virtual network that you've defined. We strongly recommend that you launch your container instances in a VPC.
If you have a default VPC, you also can skip this section and move to the next task Create a security group. 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.

For information about how to create an Amazon VPC, see Create a VPC only in the Amazon VPC User Guide. Refer to the following table to determine what options to select.

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

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 compute environment container instances, controlling both inbound and outbound traffic at the container instance level. A security group can be used only in the VPC for which it is created.

You can add rules to a security group that enable you to connect to your container instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP and HTTPS access from anywhere. Add any rules to open ports that are required by your tasks.

Note that 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.
Note
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're connecting through an Internet service provider (ISP) or from behind a firewall without a static IP address, find out the range of IP addresses that are used by client computers.

To create a security group using the console

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Security Groups.
3. Choose Create security group.
4. Enter a name and description for the security group. You cannot change the name and description of a security group after it is created.
5. From VPC, choose the VPC.
6. (Optional) By default, new security groups start with only an outbound rule that allows all traffic to leave the resource. You must add rules to enable any inbound traffic or to restrict the outbound traffic.

AWS Batch container instances don't require any inbound ports to be open. However, you might want to add an SSH rule. That way, you can log into the container instance and examine the containers in jobs with Docker commands. If you want your container instance to host a job that runs a web server, you can also add rules for HTTP. Complete the following steps to add these optional security group rules.

On the Inbound tab, create the following rules and choose Create:

- Choose Add Rule. For Type, choose HTTP. For Source, choose Anywhere (0.0.0.0/0).
- Choose Add Rule. For Type, choose SSH. For Source, choose Custom IP, and specify the public IP address of your computer or network in Classless Inter-Domain Routing (CIDR) notation. If your company allocates addresses from a range, specify the entire range, such as 203.0.113.0/24. To specify an individual IP address in CIDR notation, choose My IP. This adds the routing prefix /32 to the public IP address.
7. You can add tags now, or you can add them later. To add a tag, choose Add new tag and enter the tag key and value.

8. Choose Create security group.

To create a security group using the command line, see create-security-group (AWS CLI)

For more information about security groups, see Work with security groups.

Install the AWS CLI

To use the AWS CLI with AWS Batch, 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.
Getting Started with AWS Batch

You can use the AWS Batch first-run wizard to get started quickly with AWS Batch. After you complete the Prerequisites, you can use the first-run wizard to create a compute environment, a job definition, and a job queue.

You can also submit a sample "Hello World" job using the AWS Batch first-run wizard to test your configuration. If you already have a Docker image that you want to launch in AWS Batch, you can use that image to create a job definition.

Prerequisites

Make sure to do the following before you start the AWS Batch first-run wizard:

- Complete the steps that are described in Setting up with AWS Batch.
- Verify that your AWS account has the required permissions.

Getting Started - Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the AWS Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.

You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

Create a compute environment

To create a compute environment for an Amazon EC2 orchestration, do the following:

1. Open the AWS Batch console first-run wizard.
2. For Select orchestration type, choose Amazon Elastic Compute Cloud(Amazon EC2).
3. Choose Next.
4. In the **Compute environment configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

5. For **Instance role**, choose an existing instance profile that has the required IAM permissions attached. This instance profile allows the Amazon ECS container instances in your compute environment to make calls to the required AWS API operations. For more information, see [Amazon ECS instance role](#).

6. **(Optional)** A tag is a label that's assigned to a resource. To add a tag or an Amazon EC2 tag, expand **Tags**, then choose **Add tag**. Enter a key-value pair, and then choose **Add tag** again.

   **Important**

   If you choose **Add tag**, you must enter a key-value pair and choose **Add tag** again or choose **Remove tag**.

7. **(Optional)** In the **Instance configuration** section for **Use Amazon EC2 Spot instances**, turn on **Enable using Spot instances**.

8. **(Spot only)** For **Maximum % on-demand price**, enter the maximum percentage of On-demand pricing that you want to pay for Spot resources.

9. **(Optional)** For **Spot fleet role**, choose an existing Amazon EC2 Spot Fleet IAM role to apply to your Spot compute environment. If you don't already have an existing Amazon EC2 Spot Fleet IAM role, you must create one first. For more information, see [Amazon EC2 spot fleet role](#).

   **Important**

   To tag your Spot Instances on creation, your Amazon EC2 Spot Fleet IAM role must use the newer **AmazonEC2SpotFleetTaggingRole** managed policy. The **AmazonEC2SpotFleetRole** managed policy doesn't have the required permissions to tag Spot Instances. For more information, see [Spot Instances not tagged on creation](#) and the section called “Tagging your resources”.

10. For **Minimum vCPUs**, choose the minimum number of EC2 vCPUs that your compute environment maintains, regardless of job queue demand.

11. For **Desired vCPUs**, choose the number of EC2 vCPUs that your compute environment launches with. As job queue demand increases, AWS Batch increases the desired number of...
vCPUs and add EC2 instances. The number of vCPUs can increase up to the maximum number of vCPUs. As demand decreases, AWS Batch decreases the desired number of vCPUs and remove instances. The number of decrease all the way to the minimum number of vCPUs.

12. For **Maximum vCPUs**, choose the maximum number of EC2 vCPUs that your compute environment can scale out to, regardless of job queue demand.

13. For **Allowed instance types**, choose the Amazon EC2 instance types that can be launched. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3). Or, you can specify specific sizes within a family (such as c5.8xlarge). Metal instance types aren't in the instance families. For example, c5 doesn't include c5.metal. You can also choose **optimal** to select instance types (from the C4, M4, and R4 instance families) that match the demand of your job queues.

**Note**

When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.

**Note**

AWS Batch scales GPUs based on the required amount in your job queues. To use GPU scheduling, the compute environment must include instance types from the p2, p3, p4, p5, g3, g3s, g4, or g5 family.

**Note**

Currently, optimal uses instance types from the C4, M4, and R4 instance families. In AWS Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

14. Expand **Additional configuration**.

15. (Optional) For **Placement group**, enter a placement group name to group resources in the compute environment.
16. (Optional) For **EC2 key pair**, choose a public and private key pair as security credentials when you connect to the instance. For more information about Amazon EC2 key pairs, see Amazon EC2 key pairs and Linux instances.

17. For **Allocation strategy**, choose the allocation strategy to use when selecting instance types from the list of allowed instance types. **BEST_FIT_PROGRESSIVE** is usually the better choice for EC2 On-Demand compute environments, and **SPOT_CAPACITY_OPTIMIZED** for EC2 Spot compute environments. For more information, see the section called “Allocation strategies”.

18. (Optional) For **EC2 configuration**, choose **Add EC2 configuration**. Choose **Image type** and **Image ID override** values to provide information for AWS Batch to select Amazon Machine Images (AMIs) for instances in the compute environment. If the **Image ID override** isn't specified for each **Image type**, AWS Batch selects a recent Amazon ECS optimized AMI. If no **Image type** is specified, the default is a **Amazon Linux 2** for non-GPU, non AWS Graviton instance.

**Important**

To use a custom AMI, choose the image type and then enter the custom AMI ID in the **Image ID override** box.

---

**Amazon Linux 2**

Default for all AWS Graviton-based instance families (for example, C6g, M6g, R6g, and T4g) and can be used for all non-GPU instance types.

**Amazon Linux 2 (GPU)**

Default for all GPU instance families (for example P4 and G4) and can be used for all non AWS Graviton-based instance types.

**Amazon Linux**

Can be used for non-GPU, non AWS Graviton instance families. The standard support for Amazon Linux AMI has ended. For more information, see Amazon Linux AMI.

---

**Note**

The AMI that you choose for a compute environment must match the architecture of the instance types that you want to use for that compute environment. For example, if
your compute environment uses A1 instance types, the compute resource AMI that you choose must support Arm instances. Amazon ECS vends both x86 and Arm versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

19. (Optional) For Launch template, select an existing Amazon EC2 launch template to configure your compute resources. The default version of the template is automatically populated. For more information, see Launch template support.

Note
In a launch template, you can specify a custom AMI that you created.

20. (Optional) For Launch template version, enter $Default, $Latest, or a specific version number to use.

Important
After the compute environment is created, the launch template version used isn't changed even if the $Default or $Latest version for the launch template is updated. To use a new launch template version, first create a new compute environment, add the new compute environment to the existing job queue. Then, remove the old compute environment from the job queue, and delete the old compute environment.

21. In the Network configuration section:
   a. For Virtual Private Cloud (VPC) ID, choose an Amazon VPC.
   b. For Subnets, the subnets for your AWS account are listed. If you want to create a custom set of subnets, choose Clear subnets, and then choose the subnets that you want.

Important
Compute resources must communicate with the Amazon ECS VPC endpoint through a VPC endpoint or multiple public IP address. For more information, see Amazon ECS interface VPC endpoints (AWS PrivateLink). If your instance doesn't have a VPC endpoint configured or a public IP address, you can use network
address translation (NAT). For more information about NAT, see [NAT gateways](#) and [Creating a virtual private cloud](#).

c. For **Security groups**, choose the Amazon EC2 security groups that you want to associate with the instance. If you want to create a custom set of security groups, choose **Clear security groups**. Then, choose the security groups that you want.

22. Choose **Next**.

### Create a job queue

A job queue stores your submitted jobs until the AWS Batch Scheduler runs the job on a resource in your compute environment. For more information, see [Job queues](#).

To create a job queue for an Amazon EC2 orchestration, do the following:

1. In the **Job queue configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

2. For **Priority**, enter an integer between 0 and 100 for the job queue.

   - **Important**
     
     Higher integer values are assigned a higher priority by the AWS Batch Scheduler.

3. Choose **Next**.

### Create a job definition

AWS Batch job definitions specify how jobs are to be run. Even though each job must reference a job definition, many of the parameters that are specified in the job definition can be overridden at runtime.

To create the job definition:

1. In the **General configuration** section:
In the **General configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. The name can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

b. (Optional) For **Execution timeout**, enter the amount of time (in seconds) that an unfinished job terminates after.

⚠️ **Important**

The minimum timeout is 60 seconds.

c. (Optional) A tag is a label that's assigned to a resource. To add a tag, expand **Tags**, then choose **Add tag**. Enter a key-value pair, and then choose **Add tag** again.

⚠️ **Important**

If you choose **Add tag**, you must enter a key-value pair and choose **Add tag** again or choose **Remove tag**.

d. (Optional) Turn on **Propagate tags** to propagate tags to the Amazon Elastic Container Service task.

2. In the **Container configuration** section:

a. For **Image**, enter the name of the image that's used to launch the container. By default, all the images in the Docker Hub registry are available. You can also specify other repositories in `repository-url/image:tag` format. The parameter can be up to 255 characters in length. The parameter can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), periods (.), forward slashes (/), and number signs (#). The parameter maps to Image in the Create a container section of the Docker Remote API and the IMAGE parameter of `docker run`.

⚠️ **Note**

Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, Arm based Docker images can only run on Arm based compute resources.
• Images in Amazon ECR Public repositories use the full registry/repository[:tag] or registry/repository[@digest] naming conventions (for example, public.ecr.aws/registry_alias/my-web-app:latest).

• Images in Amazon ECR repositories use the full registry/repository:tag naming convention (for example, aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest).

• Images in official repositories on Docker Hub use a single name (for example, ubuntu or mongo).

• Images in other repositories on Docker Hub are qualified with an organization name (for example, amazon/amazon-ecs-agent).

• Images in other online repositories are qualified further by a domain name (for example, quay.io/assemblyline/ubuntu).

b. For Command, specify the command to pass 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.

   Note
   You can use parameter substitution default values and placeholders in your command. For more information, see Parameters.

c. (Optional) For Execution role, specify an IAM role that grants the Amazon ECS container agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for tasks. For more information, see Amazon ECS task execution IAM roles in the Amazon Elastic Container Service Developer Guide.

d. (Optional) For Job Role configuration, choose an IAM role that has permissions to the AWS APIs. This feature uses Amazon ECS IAM roles for tasks. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

   Note
   Only roles that have the Amazon Elastic Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for
your AWS Batch jobs, see Creating an IAM Role and Policy for your Tasks in the Amazon Elastic Container Service Developer Guide.

e. (Optional) You can add parameters to the job definition as key-value mappings to override the job definition defaults. To add a parameter:

   - For Parameters, choose Add parameter. Enter a key-value pair and then choose Add parameter again.

   \[\text{Important}\]

   If you choose Add parameter, you must configure at least one parameter or choose Remove parameter.

f. In the **Environment configuration** section for vCPUs, specify the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to docker run. Each vCPU is equivalent to 1,024 CPU shares.

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

h. For Number of GPUs, choose the number of GPUs to reserve for the container.

i. (Optional) For **Environment variables configuration**, choose Add environment variables to add environment variables to pass to the container. This parameter maps to Env in the Create a container section of the Docker Remote API and the --env option to docker run.

j. (Optional) For **Secrets**, choose Add secret to add secrets as a name-value pairs. These secrets are exposed in the container. For more information, see secretOptions in Job definition parameters for ContainerProperties.

k. (Optional) In the **Linux configuration** section:

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

   ii. To give the job container elevated permissions on the host instance (similar to the root user), drag the Privileged slider to the right. This parameter maps to
Privileged in the Create a container section of the Docker Remote API and the --privileged option to docker run.

iii. Turn on Enable init process to run an init process inside the container. This process forwards signals and reaps processes.

l. (Optional) In the Filesystem configuration section:

   i. Turn on Enable read only filesystem to remove write access to the volume.
   
   ii. For Shared memory size, enter the size (in MiB) of the /dev/shm volume.
   
   iii. For Max swap size, enter the total amount of swap memory (in MiB) that the container can use.
   
   iv. For Swappiness enter a value between 0 and 100 to indicate the swappiness behavior of the container. If you don’t specify a value and swapping is enabled, the value defaults to 60. For more information, see swappiness in Job definition parameters for ContainerProperties.
   
   v. (Optional) Expand Additional configuration.
   
   vi. For Tmpfs, choose Add tmpfs to add a tmpfs mount.
   
   vii. For Devices, choose Add device to add a device:

       A. For Container path, specify the path of in the container instance to expose the device mapped to the host instance. If you keep this blank, the host path is used in the container.
       
       B. For Host path, specify the path of a device in the host instance.
       
       C. For Permissions, choose one or more permissions to apply to the device. The available permissions are READ, WRITE, and MKNOD.

   viii. (Optional) For Ulimits configuration, choose Add ulimit to add a ulimits value for the container. Enter Name, Soft limit, and Hard limit values, and then choose Add ulimit.

3. Choose Next.

Create a job

To create a job, do the following:
1. In the **Job configuration** section for **Name**, specify a unique name for the job. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

2. Choose **Next**.

**Review and create**

On the **Review and create** page, review the configuration steps. If you need to make changes, choose **Edit**. When you’re finished, choose **Create resources**.

**Getting Started - Fargate**

AWS Fargate launches and scales the compute to closely match the resource requirements that you specify for the container. With Fargate, you don’t need to over-provision or pay for additional servers. For more information, see [Fargate](#).

**Create a compute environment**

To create a compute environment for a Fargate orchestration, do the following:

1. Open the **AWS Batch console first-run wizard**.

2. For **Select orchestration type**, choose **Fargate**.

3. Choose **Next**.

4. In the **Compute environment configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

5. (Optional) A tag is a label that’s assigned to a resource. To add a tag, expand **Tags**, then choose **Add tag**. Enter a key-value pair, and then choose **Add tag** again.

    **Important**

    If you choose **Add tag**, you must enter a key-value pair and choose **Add tag** again or choose **Remove tag**.

6. (Optional) In the **Instance configuration** section for **Use Fargate Spot capacity**, turn on **Enable using Spot instances**.

7. For **Maximum vCPUs**, enter the maximum number of vCPUs that the instance can use.
8. In the **Network configuration** section:

a. For **Virtual Private Cloud (VPC) ID**, choose an Amazon VPC.

b. For **Subnets**, the subnets for your AWS account are listed. If you want to create a custom set of subnets, choose **Clear subnets**, and then choose the subnets that you want.

![Important]

Compute resources must communicate with the Amazon ECS VPC endpoint through a VPC endpoint or multiple public IP address. For more information, see [Amazon ECS interface VPC endpoints (AWS PrivateLink)](https://docs.aws.amazon.com/ecs/latest/userguide/vpc-networking.html). If your instance doesn't have a VPC endpoint configured or a public IP address, you can use network address translation (NAT). For more information about NAT, see [NAT gateways](https://docs.aws.amazon.com/vpc/latest/userguide/using-nat.html) and [Creating a virtual private cloud](https://docs.aws.amazon.com/vpc/latest/userguide/creating-vpc.html).

c. For **Security groups**, choose the Amazon EC2 security groups that you want to associate with the instance. If you want to create a custom set of security groups, choose **Clear security groups**. Then, choose the security groups that you want.

9. Choose **Next**.

---

### Create a job queue

A job queue stores your submitted jobs until the AWS Batch Scheduler runs the job on a resource in your compute environment. To create a job queue:

To create a job queue for a Fargate orchestration, do the following:

1. In the **Job queue configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

2. For **Priority**, enter an integer between 0 and 100 for the job queue.

![Important]

Higher integer values are assigned a higher priority by the AWS Batch Scheduler.

3. Choose **Next**.
Create a job definition

To create the job definition:

1. In the **General configuration** section:
   a. For **Name**, enter a custom job definition name.
      
      In the **General configuration** section for **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
   b. (Optional) For **Execution timeout**, enter the amount of time (in seconds) that an unfinished job terminates after.

      **Important**
      
      The minimum timeout is 60 seconds.
   c. (Optional) A tag is a label that's assigned to a resource. To add a tag, expand **Tags**, then choose **Add tag**. Enter a key-value pair, and then choose **Add tag** again.

      **Important**
      
      If you choose **Add tag**, you must enter a key-value pair and choose **Add tag** again or choose **Remove tag**.
   d. (Optional) Turn on **Propagate tags** to propagate tags to the Amazon Elastic Container Service task.

2. In the **Fargate platform configuration** section:
   a. (Optional) For **Fargate platform version**, enter the specific runtime environment that you want.
   b. For **Runtime platform**, select a **LINUX** or **Windows**.
   c. (Windows only) For **Operating System Family**, select an operating system.
   d. For **CPU architecture**, select the CPU architecture that you want.
   e. (Optional) Turn on **Assign public IP** to assign a public IP address.
   f. For **Ephemeral storage**, enter the amount of ephemeral storage that you want.
g. For **Execution role**, choose a task execution role that lets Amazon Elastic Container Service (Amazon ECS) agents make AWS calls on your behalf. For example, you can choose `ecsTaskExecutionRole`.

3. In the **Container configuration** section:

   a. For **Image**, enter the name of the image that's used to launch the container. By default, all the images in the Docker Hub registry are available. You can also specify other repositories in `repository-url/image:tag` format. The parameter can be up to 255 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), periods (.), forward slashes (/), and number signs (#). The parameter maps to `Image` in the Create a container section of the Docker Remote API and the IMAGE parameter of `docker run`.

   • Images in Amazon ECR Public repositories use the full `registry/repository[:tag]` or `registry/repository[@digest]` naming conventions (for example, `public.ecr.aws/registry_alias/my-web-app:latest`).
   
   • Images in Amazon ECR repositories use the full `registry/repository:tag` naming convention (for example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`).

   • Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).

   • Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).

:i: Note

By default, 20 GiB of ephemeral storage is used. To use additional ephemeral storage, enter a value between 21 GiB and 100 GiB.

:i: Note

Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, Arm based Docker images can only run on Arm based compute resources.
- Images in other online repositories are qualified further by a domain name (for example, quay.io/assemblyline/ubuntu).

b. For **Command**, specify the command to pass 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](https://docs.docker.com/engine/reference/builder/#cmd).

   ![Note](image)
   You can use parameter substitution default values and placeholders in your command. For more information, see [Parameters](#).

   ![Tip](image)
   Choose **Info** to review Bash and JSON code examples.

c. (Optional) You can add parameters to the job definition as key-value mappings to override the job definition defaults. To add a parameter:

   - For **Parameters**, choose **Add parameter**. Enter a key-value pair and then choose **Add parameter** again.

   ![Important](image)
   If you choose **Add parameter**, you must configure at least one parameter or choose **Remove parameter**.

d. (Optional) In the **Environment configuration** section for **Job role configuration**, choose an IAM role that provides permission to use the AWS APIs.

e. In the **Environment configuration** section for **vCPUs**, specify the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to **docker run**. Each vCPU is equivalent to 1,024 CPU shares.

f. For **Memory**, specify the hard limit (in MiB) of memory to present to the job container. If your container attempts to exceed the memory specified here, the container is stopped.
Create a job

To create a Fargate job, do the following:

1. In the Job configuration section for Name, specify a unique name for the job. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
2. Choose Next.

Review and create

On the Review and create page, review the configuration steps. If you need to make changes, choose Edit. When you’re finished, choose Create resources.

Getting started with AWS Batch on Amazon EKS

AWS Batch on Amazon EKS is a managed service for scheduling and scaling batch workloads into existing Amazon EKS clusters. AWS Batch doesn't create, administer, or perform lifecycle operations of your Amazon EKS clusters on your behalf. AWS Batch orchestration scales up and down nodes managed by AWS Batch and run pods on those nodes.

AWS Batch doesn't touch nodes, auto scaling node groups or pods lifecycles that aren't associated with AWS Batch compute environments within your Amazon EKS cluster. For AWS Batch to operate effectively, its service-linked role needs Kubernetes role-based access control (RBAC) permissions in your existing Amazon EKS cluster. For more information, see Using RBAC Authorization in the Kubernetes documentation.

AWS Batch requires a Kubernetes namespace where it can scope pods as AWS Batch jobs into. We recommend a dedicated namespace to isolate the AWS Batch pods from your other cluster workloads.
After AWS Batch has been given RBAC access and a namespace has been established, you can associate that Amazon EKS cluster to an AWS Batch compute environment using the `CreateComputeEnvironment` API operation. A job queue can be associated with this new Amazon EKS compute environment. AWS Batch jobs are submitted to the job queue based on an Amazon EKS job definition using the `SubmitJob` API operation. AWS Batch then launches AWS Batch managed nodes and place jobs from job queue as Kubernetes pods into the EKS cluster associated with an AWS Batch compute environment.

The following sections cover how to get set up for AWS Batch on Amazon EKS.

**Contents**

- **Prerequisites**
- **Step 1: Preparing your Amazon EKS cluster for AWS Batch**
- **Step 2: Creating an Amazon EKS compute environment**
- **Step 3: Create a job queue and attach the compute environment**
- **Step 4: Create a job definition**
- **Step 5: Submit a job**
- **(Optional) Submit a job with overrides**
- **Getting started with AWS Batch on Amazon EKS Private Clusters**
  - **Prerequisites**
  - **Step 1: Preparing your EKS cluster for AWS Batch**
  - **Step 2: Creating an Amazon EKS compute environment**
  - **Step 3: Create a job queue and attach the compute environment**
  - **Step 4: Create a job definition**
  - **Step 5: Submit a job**
  - **(Optional) Submit a job with overrides**
  - **Troubleshooting**

**Prerequisites**

Before starting this tutorial, you must install and configure the following tools and resources that you need to create and manage both AWS Batch and Amazon EKS resources.
• **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. This guide requires that you use version 2.8.6 or later or 1.26.0 or later. For more information, see *Installing, updating, and uninstalling the AWS CLI* in the *AWS Command Line Interface User Guide*. After installing the AWS CLI, we recommend that you also configure it. For more information, see *Quick configuration with aws configure* in the *AWS Command Line Interface User Guide*.

• **kubectl** – A command line tool for working with Kubernetes clusters. This guide requires that you use version 1.23 or later. For more information, see *Installing or updating kubectl* in the *Amazon EKS User Guide*.

• **eksctl** – A command line tool for working with Amazon EKS clusters that automates many individual tasks. This guide requires that you use version 0.115.0 or later. For more information, see *Installing or updating eksctl* in the *Amazon EKS User Guide*.

• **Required IAM permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles and service linked roles, AWS CloudFormation, and a VPC and related resources. For more information, see *Actions, resources, and condition keys for Amazon Elastic Kubernetes Service* and *Using service-linked roles* in the *IAM User Guide*. You must complete all steps in this guide as the same user.

• **Creating an Amazon EKS cluster** – For more information, see *Getting started with Amazon EKS – eksctl* in the *Amazon EKS User Guide*.

**Note**

AWS Batch only supports Amazon EKS clusters with API server endpoints that have public access, accessible to the public internet. By default, Amazon EKS clusters API server endpoints have public access. For more information, see *Amazon EKS cluster endpoint access control* in the *Amazon EKS User Guide*.

**Note**

AWS Batch doesn't provide managed-node orchestration for CoreDNS or other deployment pods. If you need CoreDNS, see *Adding the CoreDNS Amazon EKS add-on* in the *Amazon EKS User Guide*. Or, use eksctl create cluster create to create the cluster, it includes CoreDNS by default.
• **Permissions** – Users calling the CreateComputeEnvironment API operation to create a compute environment that uses Amazon EKS resources require permissions to the eks:DescribeCluster API operation. Using the AWS Management Console to create a compute resource using Amazon EKS resources requires permissions to both eks:DescribeCluster and eks:ListClusters.

**Step 1: Preparing your Amazon EKS cluster for AWS Batch**

All steps are required.

1. **Create a dedicated namespace for AWS Batch jobs**

   Use kubectl to create a new namespace.

   ```bash
   $ namespace=my-aws-batch-namespace
   $ cat - <<EOF | kubectl create -f -
   {
       "apiVersion": "v1",
       "kind": "Namespace",
       "metadata": {
           "name": "${namespace}",
           "labels": {
               "name": "${namespace}"
           }
       }
   }
   EOF
   ``

   Output:

   ```
   namespace/my-aws-batch-namespace created
   ```

2. **Enable access via role-based access control (RBAC)**

   Use kubectl to create a Kubernetes role for the cluster to allow AWS Batch to watch nodes and pods, and to bind the role. You must do this once for each EKS cluster.
**Note**

For more information about using RBAC authorization, see [Using RBAC Authorization](#) in the *Kubernetes User Guide*.

```bash
$ cat - <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: aws-batch-cluster-role
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["daemons", "deployments", "statefulsets", "replicasets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["clusterroles", "clusterrolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: aws-batch-cluster-role-binding
subjects:
  - kind: User
    name: aws-batch
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
```
name: `aws-batch-cluster-role`
apiGroup: rbac.authorization.k8s.io

EOF

Output:

```
clusterrole.rbac.authorization.k8s.io/aws-batch-cluster-role created
clusterrolebinding.rbac.authorization.k8s.io/aws-batch-cluster-role-binding created
```

Create namespace-scoped Kubernetes role for AWS Batch to manage and lifecycle pods and bind it. You must do this once for each unique namespace.

```
$ namespace=my-aws-batch-namespace
$ cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: `aws-batch-compute-environment-role`
  namespace: ${namespace}
rules:
  - apiGroups: [""
    resources: ["pods"]
    verbs: ["create", "get", "list", "watch", "delete", "patch"]
  - apiGroups: [""
    resources: ["serviceaccounts"]
    verbs: ["get", "list"]
  - apiGroups: ["rbac.authorization.k8s.io"
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: `aws-batch-compute-environment-role-binding`
  namespace: ${namespace}
subjects:
  - kind: User
    name: `aws-batch`
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: `aws-batch-compute-environment-role`
```
apiGroup: rbac.authorization.k8s.io
EOF

Output:

role.rbac.authorization.k8s.io/aws-batch-compute-environment-role created
rolebinding.rbac.authorization.k8s.io/aws-batch-compute-environment-role-binding created

Update Kubernetes aws-auth configuration map to map the preceding RBAC permissions to the AWS Batch service-linked role.

```
$ eksctl create iamidentitymapping \
   --cluster my-cluster-name \
   --arn "arn:aws:iam::<your-account>::role/AWSServiceRoleForBatch" \
   --username aws-batch
```

Output:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account>::role/AWSServiceRoleForBatch" to auth ConfigMap
```

### Note

The path aws-service-role/batch.amazonaws.com/ has been removed from the ARN of the service-linked role. This is because of an issue with the aws-auth configuration map. For more information, see [Roles with paths don't work when the path is included in their ARN in the aws-authconfigmap](#).

---

**Step 2: Creating an Amazon EKS compute environment**

AWS Batch compute environments define compute resource parameters to meet your batch workload needs. In a managed compute environment, AWS Batch helps you to manage the capacity and instance types of the compute resources (Kubernetes nodes) within your Amazon EKS cluster. This is based on the compute resource specification that you define when you create the compute environment. You can use EC2 On-Demand Instances or EC2 Spot Instances.
Now that the `AWSServiceRoleForBatch` service-linked role has access to your Amazon EKS cluster, you can create AWS Batch resources. First, create a compute environment that points to your Amazon EKS cluster.

```bash
$ cat <<EOF > ./batch-eks-compute-environment.json
{
    "computeEnvironmentName": "My-Eks-CE1",
    "type": "MANAGED",
    "state": "ENABLED",
    "eksConfiguration": {
        "eksClusterArn": "arn:aws:eks:<region>:123456789012:cluster/<cluster-name>",
        "kubernetesNamespace": "my-aws-batch-namespace"
    },
    "computeResources": {
        "type": "EC2",
        "allocationStrategy": "BEST_FIT_PROGRESSIVE",
        "minvCpus": 0,
        "maxvCpus": 128,
        "instanceTypes": [
            "m5"
        ],
        "subnets": [
            "<eks-cluster-subnets-with-access-to-internet-for-image-pull>"
        ],
        "securityGroupIds": [
            "<eks-cluster-sg>"
        ],
        "instanceRole": "<eks-instance-profile>"
    }
}
EOF
$ aws batch create-compute-environment --cli-input-json file://./batch-eks-compute-environment.json
```

**Notes**

- The `serviceRole` parameter should not be specified, then the AWS Batch service-linked role will be used. AWS Batch on Amazon EKS only supports the AWS Batch service-linked role.
- Only `BEST_FIT_PROGRESSIVE`, `SPOT_CAPACITY_OPTIMIZED`, and `SPOT_PRICE_CAPACITY_OPTIMIZED` allocation strategies are supported for Amazon EKS compute environments.
Note

We recommend that you use SPOT_PRICE_CAPACITY_OPTIMIZED rather than SPOT_CAPACITY_OPTIMIZED in most instances.

• For the instanceRole, see Creating the Amazon EKS node IAM role and Enabling IAM principal access to your cluster in the Amazon EKS User Guide. If you're using pod networking, see Configuring the Amazon VPC CNI plugin for Kubernetes to use IAM roles for service accounts in the Amazon EKS User Guide.

• A way to get working subnets for the subnets parameter is to use the Amazon EKS managed node groups public subnets that were created by eksctl when creating an Amazon EKS cluster. Otherwise, use subnets that have a network path that supports pulling images.

• The securityGroupIds parameter can use the same security group as the Amazon EKS cluster. This command retrieves the security group ID for the cluster.

```bash
$ eks describe-cluster \
   --name <cluster-name> \
   --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

• Maintenance of an Amazon EKS compute environment is a shared responsibility. For more information, see Shared responsibility of the Kubernetes nodes.

Important

It's important to confirm that the compute environment is healthy before proceeding. The DescribeComputeEnvironments API operation can be used to do this.

```bash
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1
```

Confirm that the status parameter is not INVALID. If it is, look at the statusReason parameter for the cause. For more information, see Troubleshooting AWS Batch.

Step 3: Create a job queue and attach the compute environment

```bash
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1
```
Jobs submitted to this new job queue are run as pods on AWS Batch managed nodes that joined the Amazon EKS cluster that's associated with your compute environment.

```bash
$ cat <<EOF > ./batch-eks-job-queue.json
{
    "jobQueueName": "My-Eks-JQ1",
    "priority": 10,
    "computeEnvironmentOrder": [
        {
            "order": 1,
            "computeEnvironment": "My-Eks-CE1"
        }
    ]
}
EOF
$ aws batch create-job-queue --cli-input-json file://./batch-eks-job-queue.json
```

**Step 4: Create a job definition**

```bash
$ cat <<EOF > ./batch-eks-job-definition.json
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "hostNetwork": true,
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": [
                        "sleep",
                        "60"
                    ],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    }
                }
            ]
        }
    },
    "metadata": {
        "labels": {
```

Step 4: Create a job definition
"environment": "test"

$ aws batch register-job-definition --cli-input-json file://./batch-eks-job-definition.json

Notes

- Only single container jobs are supported.
- There are considerations for the cpu and memory parameters. For more information, see Memory and vCPU considerations for AWS Batch on Amazon EKS.

Step 5: Submit a job

$ aws batch submit-job --job-queue My-Eks-JQ1 --job-definition MyJobOnEks_Sleep --job-name My-Eks-Job1
$ aws batch describe-jobs --job <jobId-from-submit-response>

Notes

- Only single container jobs are supported.
- Make sure you're familiar with all the relevant considerations for the cpu and memory parameters. For more information, see Memory and vCPU considerations for AWS Batch on Amazon EKS.
- For more information about running jobs on Amazon EKS resources, see Amazon EKS jobs.

(Optional) Submit a job with overrides

This job overrides the command passed to the container.

$ cat <<EOF > ./submit-job-override.json
{
  "jobName": "EksWithOverrides",
  "jobQueue": "My-Eks-JQ1",
  "jobDefinition": "MyJobOnEks_Sleep",
EOF
Getting started with AWS Batch on Amazon EKS Private Clusters

AWS Batch is a managed service that orchestrates batch workloads in your Amazon Elastic Kubernetes Service (Amazon EKS) clusters. This includes queuing, dependency tracking, managed job retries and priorities, pod management, and node scaling. This feature connects your existing private Amazon EKS cluster with AWS Batch to run your jobs at scale. You can use eksctl (a command line interface for Amazon EKS), the AWS console, or the AWS Command Line Interface to create a private Amazon EKS cluster with all the other necessary resources. Support for private
Amazon EKS clusters on AWS Batch is generally available in [commercial AWS Regions where AWS Batch](#) is available.

Amazon EKS private only clusters have no inbound/outbound internet access, and only have private subnets. Amazon VPC endpoints are used to enable private access to other AWS services. ekscctl supports creating fully-private clusters using a pre-existing Amazon VPC and subnets. ekscctl also creates Amazon VPC endpoints in the supplied Amazon VPC and modifies route tables for the supplied subnets.

Each subnet should have an explicit route table associated with it because ekscctl does not modify the main route table. Your cluster must pull images from a container registry that's in your Amazon VPC. As well, you can create an Amazon Elastic Container Registry in your Amazon VPC and copy container images to it for your nodes to pull from. For more information, see [Copy a container image from one repository to another repository](#). To get started with Amazon ECR private repositories, see [Amazon ECR private repositories](#).

You can optionally create a pull through cache rule with Amazon ECR. Once a pull through cache rule is created for an external public registry, you can pull an image from that external public registry using your Amazon ECR private registry uniform resource identifier (URI). Then Amazon ECR creates a repository and caches the image. When a cached image is pulled using the Amazon ECR private registry URI, Amazon ECR checks the remote registry to see if there is a new version of the image and updates your private registry up to once every 24 hours.

**Contents**

- Prerequisites
- Step 1: Preparing your EKS cluster for AWS Batch
- Step 2: Creating an Amazon EKS compute environment
- Step 3: Create a job queue and attach the compute environment
- Step 4: Create a job definition
- Step 5: Submit a job
  - (Optional) Submit a job with overrides
- Troubleshooting

**Prerequisites**

Before starting this tutorial, you must install and configure the following tools and resources that you need to create and manage both AWS Batch and Amazon EKS resources. You also need
to create all the necessary resources including VPC, subnets, route-tables, VPC endpoints, and Amazon EKS cluster. You need to use the AWS CLI.

- **AWS CLI** – A command line tool to work with AWS services, including Amazon EKS. This guide requires that you use version 2.8.6 or later or 1.26.0 or later. For more information, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide.

After installing the AWS CLI, we recommend that you configure it. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide.

- **kubect1** – A command line tool to work with Kubernetes clusters. This guide requires that you use version 1.23 or later. For more information, see Installing or updating kubect1 in the Amazon EKS User Guide.

- **eksctl** – A command line tool to work with Amazon EKS clusters that automates many individual tasks. This guide requires that you use version 0.115.0 or later. For more information, see Installing or updating eksctl in the Amazon EKS User Guide.

- **Required AWS Identity and Access Management (IAM) permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles and service linked roles, AWS CloudFormation, and a VPC and related resources. For more information, see Actions, resources, and condition keys for Amazon Elastic Kubernetes Service and Using service-linked roles in the IAM User Guide. You must complete all steps in this guide as the same user.

- **Creating an Amazon EKS cluster** – For more information, see Getting started with Amazon EKS – eksctl in the Amazon EKS User Guide.

**Note**

AWS Batch doesn't provide managed-node orchestration for CoreDNS or other deployment pods. If you need CoreDNS, see Adding the CoreDNS Amazon EKS add-on in the Amazon EKS User Guide. Or, use eksctl create cluster create to create the cluster, it includes CoreDNS by default.

- **Permissions** – Users calling the CreateComputeEnvironment API operation to create a compute environment that uses Amazon EKS resources require permissions to the eks:DescribeCluster API operation. Using the AWS Management Console to create a compute resource using Amazon EKS resources requires permissions to both eks:DescribeCluster and eks:ListClusters.

- Create a private EKS cluster in the us-east-1 region using the sample eksctl config file.
Create your resources using the command: eksctl create cluster -f clusterConfig.yaml

- Batch managed nodes must be deployed to subnets that have the VPC interface endpoints that you require. For more information, see [Private cluster requirements](#).

### Step 1: Preparing your EKS cluster for AWS Batch

All steps are required.

1. **Create a dedicated namespace for AWS Batch jobs**

   Use kubectl to create a new namespace.

   ```bash
   $ namespace=my-aws-batch-namespace
   $ cat - <<EOF | kubectl create -f -
   {
       "apiVersion": "v1",
       "kind": "Namespace",
       "metadata": {
           "name": "${namespace}",
           "labels": {
               "name": "${namespace}"
           }
       }
   }
   EOF
   ```

   Output:
namespace/my-aws-batch-namespace created

2. **Enable access via role-based access control (RBAC)**

Use `kubectl` to create a Kubernetes role for the cluster to allow AWS Batch to watch nodes and pods, and to bind the role. You must do this once for each Amazon EKS cluster.

<i>Note</i>

For more information about using RBAC authorization, see [Using RBAC Authorization](#) in the Kubernetes documentation.

```bash
$ cat - <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: aws-batch-cluster-role
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["apps"]
    resources: ["daemonsets", "deployments", "statefulsets", "replicasets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "clusterrolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
```
Create namespace-scoped Kubernetes role for AWS Batch to manage and lifecycle pods and bind it. You must do this once for each unique namespace.

```
$ namespace=my-aws-batch-namespace
$ cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: aws-batch-compute-environment-role
  namespace: ${namespace}
rules:
- apiGroups: [""
  resources: ["pods"]
  verbs: ["create", "get", "list", "watch", "delete", "patch"]
- apiGroups: [""
  resources: ["serviceaccounts"]
  verbs: ["get", "list"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: aws-batch-compute-environment-role-binding
  namespace: ${namespace}
EOF
```
subjects:
- kind: User
  name: aws-batch
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: aws-batch-compute-environment-role
  apiGroup: rbac.authorization.k8s.io
EOF

Output:

role.rbac.authorization.k8s.io/aws-batch-compute-environment-role created
rolebinding.rbac.authorization.k8s.io/aws-batch-compute-environment-role-binding created

Update Kubernetes aws-auth configuration map to map the preceding RBAC permissions to the AWS Batch service-linked role.

```
$ eksctl create iamidentitymapping \
  --cluster my-cluster-name \
  --arn "arn:aws:iam::<your-account>:role/AWSServiceRoleForBatch" \
  --username aws-batch
```

Output:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account>:role/AWSServiceRoleForBatch" to auth ConfigMap
```

**Note**

The path aws-service-role/batch.amazonaws.com/ has been removed from the ARN of the service-linked role. This is because of an issue with the aws-auth configuration map. For more information, see [Roles with paths don't work when the path is included in their ARN in the aws-authconfigmap](#).
Step 2: Creating an Amazon EKS compute environment

AWS Batch compute environments define compute resource parameters to meet your batch workload needs. In a managed compute environment, AWS Batch helps you to manage the capacity and instance types of the compute resources (Kubernetes nodes) within your Amazon EKS cluster. This is based on the compute resource specification that you define when you create the compute environment. You can use EC2 On-Demand Instances or EC2 Spot Instances.

Now that the **AWSServiceRoleForBatch** service-linked role has access to your Amazon EKS cluster, you can create AWS Batch resources. First, create a compute environment that points to your Amazon EKS cluster.

```bash
$ cat <<EOF > ./batch-eks-compute-environment.json
{
    "computeEnvironmentName": "My-Eks-CE1",
    "type": "MANAGED",
    "state": "ENABLED",
    "eksConfiguration": {
        "eksClusterArn": "arn:aws:eks:<region>:123456789012:cluster/<cluster-name>",
        "kubernetesNamespace": "my-aws-batch-namespace"
    },
    "computeResources": {
        "type": "EC2",
        "allocationStrategy": "BEST_FIT_PROGRESSIVE",
        "minvCpus": 0,
        "maxvCpus": 128,
        "instanceTypes": ["m5"
        ],
        "subnets": [
            "<eks-cluster-subnets-with-access-to-the-image-for-image-pull>"
        ],
        "securityGroupIds": ["<eks-cluster-sg>"
        ],
        "instanceRole": "<eks-instance-profile>"
    }
}
EOF
$ aws batch create-compute-environment --cli-input-json file://./batch-eks-compute-environment.json
```
Notes

- The serviceRole parameter should not be specified, then the AWS Batch service-linked role will be used. AWS Batch on Amazon EKS only supports the AWS Batch service-linked role.

- Only BEST_FIT_PROGRESSIVE, SPOT_CAPACITY_OPTIMIZED, and SPOT_PRICE_CAPACITY_OPTIMIZED allocation strategies are supported for Amazon EKS compute environments.

**Note**

We recommend that you use SPOT_PRICE_CAPACITY_OPTIMIZED rather than SPOT_CAPACITY_OPTIMIZED in most instances.

- For the instanceRole, see Creating the Amazon EKS node IAM role and Enabling IAM principal access to your cluster in the Amazon EKS User Guide. If you're using pod networking, see Configuring the Amazon VPC CNI plugin for Kubernetes to use IAM roles for service accounts in the Amazon EKS User Guide.

- A way to get working subnets for the subnets parameter is to use the Amazon EKS managed node groups public subnets that were created by eksctl when creating an Amazon EKS cluster. Otherwise, use subnets that have a network path that supports pulling images.

- The securityGroupIds parameter can use the same security group as the Amazon EKS cluster. This command retrieves the security group ID for the cluster.

```
$ eks describe-cluster \
   --name <cluster-name> \
   --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

- Maintenance of an Amazon EKS compute environment is a shared responsibility. For more information, see Security in Amazon EKS.

**Important**

It's important to confirm that the compute environment is healthy before proceeding. The DescribeComputeEnvironments API operation can be used to do this.

```
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1
```
Step 3: Create a job queue and attach the compute environment

$ aws batch describe-compute-environments --compute-environments My-Eks-CE1

Jobs submitted to this new job queue are run as pods on AWS Batch managed nodes that joined the Amazon EKS cluster that's associated with your compute environment.

$ cat <<EOF > ./batch-eks-job-queue.json
{
  "jobQueueName": "My-Eks-JQ1",
  "priority": 10,
  "computeEnvironmentOrder": [
    {
      "order": 1,
      "computeEnvironment": "My-Eks-CE1"
    }
  ]
}
EOF
$ aws batch create-job-queue --cli-input-json file:///./batch-eks-job-queue.json

Step 4: Create a job definition

In the image field of the job definition, instead of providing a link to image in a public ECR repository, provide the link to the image stored in our private ECR repository. See the following sample job-definition:

$ cat <<EOF > ./batch-eks-job-definition.json
{
  "jobDefinitionName": "MyJobOnEks_Sleep",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "hostNetwork": true,
      "containers": [
      
EOF
To run kubectl commands, you will need private access to your Amazon EKS cluster. This means all traffic to your cluster API server must come from within your cluster's VPC or a connected network.

**Step 5: Submit a job**

```bash
$ aws batch register-job-definition --cli-input-json file://./batch-eks-job-definition.json
```

```bash
$ aws batch submit-job -j My-Eks-JQ1 \
    -j MyJobOnEks_Sleep -j My-Eks-Job1
$ aws batch describe-jobs -j <jobId-from-submit-response>
```

**Notes**

- Only single container jobs are supported.
- Make sure you're familiar with all the relevant considerations for the cpu and memory parameters. For more information, see [Memory and vCPU considerations for AWS Batch on Amazon EKS](#).
- For more information about running jobs on Amazon EKS resources, see [Amazon EKS jobs](#).
(Optional) Submit a job with overrides

This job overrides the command passed to the container.

```bash
$ cat <<EOF > ./submit-job-override.json
{
  "jobName": "EksWithOverrides",
  "jobQueue": "My-Eks-JQ1",
  "jobDefinition": "MyJobOnEks_Sleep",
  "eksPropertiesOverride": {
    "podProperties": {
      "containers": [
        
        ],
      "args": [
        "-c",
        "echo hello world"
        ]
      }
    }
  }
}
EOF
$ aws batch submit-job - -cli-input-json file://./submit-job-override.json
```

Notes

- AWS Batch aggressively cleans up the pods after the jobs complete to reduce the load to Kubernetes. To examine the details of a job, logging must be configured. For more information, see [Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs](#).

- For improved visibility into the details of the operations, enable Amazon EKS control plane logging. For more information, see [Amazon EKS control plane logging](#) in the Amazon EKS User Guide.

- Daemonsets and kubelets overhead affects available vCPU and memory resources, specifically scaling and job placement. For more information, see [Memory and vCPU considerations for AWS Batch on Amazon EKS](#).
Troubleshooting

If nodes launched by AWS Batch don't have access to the Amazon ECR repository (or any other repository) that stores your image, then your jobs could remain in the STARTING state. This is because the pod will not be able to download the image and run your AWS Batch job. If you click on the pod name launched by AWS Batch you should be able to see the error message and confirm the issue. The error message should look similar to the following:

```
Failed to pull image "public.ecr.aws/amazonlinux/amazonlinux:2": rpc error: code = Unknown desc = failed to pull and unpack image
"public.ecr.aws/amazonlinux/amazonlinux:2": failed to resolve reference
"public.ecr.aws/amazonlinux/amazonlinux:2": failed to do request: Head
"https://public.ecr.aws/v2/amazonlinux/amazonlinux/manifests/2": dial tcp: i/o timeout
```

For other common troubleshooting scenarios, see Troubleshooting AWS Batch. For troubleshooting bases on pod-status, see How do I troubleshoot the pod status in Amazon EKS?
**Jobs**

Jobs are the unit of work that's started by AWS Batch. Jobs can be invoked as containerized applications that run on Amazon ECS container instances in an ECS cluster.

Containerized jobs can reference a container image, command, and parameters. For more information, see [Job definition parameters for ContainerProperties](#).

You can submit a large number of independent, simple jobs.

**Topics**

- [Submitting a job](#)
- Job states
- [AWS Batch job environment variables](#)
- Automated job retries
- Job dependencies
- Job timeouts
- Amazon EKS jobs
- Array jobs
- Multi-node parallel jobs
- GPU jobs
- [To create a GPU-based job on Amazon EKS resources](#)
- Search and filter AWS Batch jobs
- Job logs
- Job information

**Submitting a job**

After you register a job definition, you can submit it as a job to an AWS Batch job queue. You can override many of the parameters that are specified in the job definition at runtime.

**To submit a job**

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Jobs.
4. Choose Submit new job.
5. For Name, enter a unique name for your job definition. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
6. For Job definition, choose an existing job definition for your job. For more information, see Creating a single-node job definition.
7. For Job queue, choose an existing job queue. For more information, see Creating a job queue.
8. For Job dependencies, choose Add Job dependencies.
   
   • For Job id, enter the job ID for any dependencies. Then choose Add job dependencies. A job can have up to 20 dependencies. For more information, see Job dependencies.
9. (Array jobs only) For Array size, specify an array size between 2 and 10,000.
10. (Optional) Expand Tags and then choose Add tag to add tags to the resource. Enter a key and optional value, then choose Add tag.
11. Choose Next page.
12. In the Job overrides section:
   
   a. (Optional) For Scheduling priority, enter a scheduling priority value between 0 and 100. Higher values are given higher priority.
   
   b. (Optional) For Job attempts, enter the maximum number of times that AWS Batch attempts to move the job to a RUNNABLE status. You can enter a number between 1 and 10. For more information, see Automated job retries.
   
   c. (Optional) For Execution timeout, enter the timeout value (in seconds). The execution timeout is the length of time before an unfinished job is terminated. If an attempt exceeds the timeout duration, it's stopped and moves to a FAILED status. For more information, see Job timeouts. The minimum value is 60 seconds.

⚠️ Important

Don't rely on jobs that run on Fargate resources to run for more than 14 days. After 14 days, the Fargate resources might no longer be available with the job being likely terminated.
d. (Optional) Turn on Propagate tags to propagate tags from the job and job definition to the Amazon ECS task.

13. Expand Additional configuration.

14. (Optional) For Retry strategy conditions, choose Add evaluate on exit. Enter at least one parameter value and then choose an Action. For each set of conditions, Action must be set to either Retry or Exit. These actions mean the following:

- **Retry** – AWS Batch retries until the number of job attempts that you specified is reached.
- **Exit** – AWS Batch stops retrying the job.

**Important**

If you choose Add evaluate on exit, configure at least one parameter and choose either an Action or choose Remove evaluate on exit.

15. For Parameters, choose Add parameters to add parameter substitution placeholders. Then, enter a Key and an optional Value.

16. In the Container overrides section:

a. For Command, specify the command to pass to the container. For simple commands, enter the command as you do for a command prompt. For more complicated commands, for example with special characters), use JSON syntax.

**Note**

This parameter can't contain an empty string.

b. For vCPUs, enter the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to docker run. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

c. For Memory, enter the memory limit that's available to the container. If your container attempts to exceed the memory specified here, the container is stopped. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job.
**Note**

To maximize your resource utilization, prioritize memory for jobs of a specific instance type. For more information, see [Compute Resource Memory Management](#).

d. (Optional) For **Number of GPUs**, choose the number of GPUs to reserve for the container.

e. (Optional) For **Environment variables**, choose **Add environment variable** to add environment variables as name-value pairs. These variables are passed to the container.

f. Choose **Next page**.

g. For **Job review**, review the configuration steps. If you need to make changes, choose **Edit**. When you’re finished, choose **Create job definition**.

### Job states

When you submit a job to an AWS Batch job queue, the job enters the SUBMITTED state. It then passes through the following states until it succeeds (exits with code 0) or fails (exits with a non-zero code). AWS Batch jobs can have the following states:

#### SUBMITTED

A job that's submitted to the queue, and has not yet been evaluated by the scheduler. The scheduler evaluates the job to determine if it has any outstanding dependencies on the successful completion of any other jobs. If there are dependencies, the job is moved to PENDING. If there are no dependencies, the job is moved to RUNNABLE.

#### PENDING

A job that resides in the queue and isn't yet able to run due to a dependency on another job or resource. After the dependencies are satisfied, the job is moved to RUNNABLE.

#### RUNNABLE

A job that resides in the queue, has no outstanding dependencies, and is therefore ready to be scheduled to a host. Jobs in this state are started as soon as sufficient resources are available in one of the compute environments that are mapped to the job's queue. However, jobs can remain in this state indefinitely when sufficient resources are unavailable.
STARTING

These jobs have been scheduled to a host and the relevant container initiation operations are underway. After the container image is pulled and the container is up and running, the job transitions to RUNNING.

RUNNING

The job is running as a container job on an Amazon ECS container instance within a compute environment. When the job's container exits, the process exit code determines whether the job succeeded or failed. An exit code of 0 indicates success, and any non-zero exit code indicates failure. If the job associated with a failed attempt has any remaining attempts left in its optional retry strategy configuration, the job is moved to RUNNABLE again. For more information, see Automated job retries.

SUCCEEDED

The job has successfully completed with an exit code of 0. The job state for SUCCEEDED jobs is persisted in AWS Batch for at least 7 days.
Note
Logs for SUCCEEDED jobs are available in CloudWatch Logs. The log group is /aws/batch/job, and the log stream name format is as follows: first200CharsOfJobDefinitionName/default/ecs_task_id. This format may change in the future.
After a job reaches the RUNNING status, you can programmatically retrieve its log stream name with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs never expire. However, you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

Failed
The job has failed all available attempts. The job state for FAILED jobs is persisted in AWS Batch for at least 7 days.

Note
Logs for FAILED jobs are available in CloudWatch Logs. The log group is /aws/batch/job, and the log stream name format is as follows: first200CharsOfJobDefinitionName/default/ecs_task_id. This format may change in the future.
After a job reaches the RUNNING status, you can programmatically retrieve its log stream with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs never expire. However, you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

AWS Batch job environment variables
AWS Batch sets specific environment variables in container jobs. These environment variables provide introspection for the containers inside jobs. You can use the values of these variables in the
logic of your applications. All variables that AWS Batch set start with the `AWS_BATCH_` prefix. This is a protected environment variable prefix. You can't use this prefix for your own variables in job definitions or overrides.

The following environment variables are available in job containers:

**AWS_BATCH_CE_NAME**

This variable is set to the name of the compute environment where your job is placed.

**AWS_BATCH_JOB_ARRAY_INDEX**

This variable is only set in child array jobs. The array job index begins at 0, and each child job receives a unique index number. For example, an array job with 10 children has index values of 0-9. You can use this index value to control how your array job children are differentiated. For more information, see [Tutorial: Using the array job index to control job differentiation](#).

**AWS_BATCH_JOB_ARRAY_SIZE**

This variable is set to the size of the parent array job. The size of the parent array job is passed to the child array job in this variable.

**AWS_BATCH_JOB_ATTEMPT**

This variable is set to the job attempt number. The first attempt is numbered 1. For more information, see [Automated job retries](#).

**AWS_BATCH_JOB_ID**

This variable is set to the AWS Batch job ID.

**AWS_BATCH_JOB_KUBERNETES_NODE_UID**

This variable is set as the Kubernetes UID of the node object that's in the Kubernetes cluster that the pod runs on. This variable is only set for jobs that run on Amazon EKS resources. For more information, see [UIDs](#) in the Kubernetes documentation.

**AWS_BATCH_JOB_MAIN_NODE_INDEX**

This variable is only set in multi-node parallel jobs. This variable is set to the index number of the job's main node. Your application code can compare the `AWS_BATCH_JOB_MAIN_NODE_INDEX` to the `AWS_BATCH_JOB_NODE_INDEX` on an individual node to determine if it's the main node.
AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS

This variable is only set in multi-node parallel job child nodes. This variable isn’t present on the main node, but is set to the private IPv4 address of the job's main node. Your child node's application code can use this address to communicate with the main node.

AWS_BATCH_JOB_NODE_INDEX

This variable is only set in multi-node parallel jobs. This variable is set to the node index number of the node. The node index begins at 0, and each node receives a unique index number. For example, a multi-node parallel job with 10 children has index values of 0-9.

AWS_BATCH_JOB_NUM_NODES

This variable is only set in multi-node parallel jobs. This variable is set to the number of nodes that you requested for your multi-node parallel job.

AWS_BATCH_JQ_NAME

This variable is set to the name of the job queue that your job was submitted to.

Automated job retries

You can apply a retry strategy to your jobs and job definitions that allows failed jobs to be automatically retried. Possible failure scenarios include the following:

- Any non-zero exit code from a container job
- Amazon EC2 instance failure or termination
- Internal AWS service error or outage

When a job is submitted to a job queue and placed into the RUNNING state that’s considered an attempt. By default, each job is given one attempt to move to either the SUCCEEDED or FAILED job state. However, both the job definition and the job submission workflows can be used to specify a retry strategy with between 1 and 10 attempts. If evaluateOnExit is specified, it can contain up to 5 retry strategies. If evaluateOnExit is specified, but none of the retry strategies match, then the job is retried. For jobs that don't match to exit, add a final entry that exits for any reason. For example, this evaluateOnExit object has two entries that with actions of RETRY and a final entry with an action of EXIT.

"evaluateOnExit": [
At runtime, the `AWS_BATCH_JOB_ATTEMPT` environment variable is set to the container's corresponding job attempt number. The first attempt is numbered 1, and subsequent attempts are in ascending order (for example, 2, 3, 4).

For example, suppose that a job attempt fails for any reason and the number of attempts specified in the retry configuration is greater than the `AWS_BATCH_JOB_ATTEMPT` number. Then, the job is placed back in the `RUNNABLE` state. For more information, see `Job states`.

**Note**

Jobs that are cancelled or terminated aren't retried. Also, jobs that fail because of an invalid job definition aren't retried.

For more information, see `Retry strategy`, `Creating a single-node job definition`, `Submitting a job` and `Stopped tasks error codes`.

**Job dependencies**

When you submit an AWS Batch job, you can specify the job IDs that the job depends on. When you do so, the AWS Batch scheduler ensures that your job is run only after the specified dependencies have successfully completed. After they succeed, the dependent job transitions from `PENDING` to `RUNNABLE` and then to `STARTING` and `RUNNING`. If any of the job dependencies fail, the dependent job automatically transitions from `PENDING` to `FAILED`. 
For example, Job A can express a dependency on up to 20 other jobs that must succeed before it can run. You can then submit additional jobs that have a dependency on Job A and up to 19 other jobs.

For array jobs, you can specify a SEQUENTIAL type dependency without specifying a job ID so that each child array job completes sequentially, starting at index 0. You can also specify an N_TO_N type dependency with a job ID. That way, each index child of this job must wait for the corresponding index child of each dependency to complete before it can begin. For more information, see Array jobs.

To submit an AWS Batch job with dependencies, see Submitting a job.

Job timeouts

You can configure a timeout duration for your jobs so that if a job runs longer than that, AWS Batch terminates the job. For example, you might have a job that you know should only take 15 minutes to complete. Sometimes your application gets stuck in a loop and runs forever, so you can set a timeout of 30 minutes to terminate the stuck job.

⚠️ Important

By default, AWS Batch doesn't have a job timeout. If you don't define a job timeout, the job runs until the container exits.

You specify an attemptDurationSeconds parameter, which must be at least 60 seconds, either in your job definition, or when you submit the job. When this number of seconds has passed following the job attempt's startedAt timestamp, AWS Batch terminates the job. On the compute resource, your job's container receives a SIGTERM signal to give your application a chance to shut down gracefully. If the container is still running after 30 seconds, a SIGKILL signal is sent to forcefully shut down the container.

Timeout terminations are handled on a best-effort basis. You shouldn't expect your timeout termination to happen exactly when the job attempt times out (it may take a few seconds longer). If your application requires precise timeout execution, you should implement this logic within the application. If you have a large number of jobs timing out concurrently, the timeout terminations behave as a first in, first out queue, where jobs are terminated in batches.
Note

There's no maximum timeout value for an AWS Batch job.

If a job is terminated for exceeding the timeout duration, it isn't retried. If a job attempt fails on its own, then it can retry if retries are enabled, and the timeout countdown is started over for the new attempt.

Important

Jobs that run on Fargate resources can't expect to run for more than 14 days. If the timeout duration exceeds 14 days, the Fargate resources may no longer be available and the job will be terminated.

For array jobs, child jobs have the same timeout configuration as the parent job.

For information about submitting an AWS Batch job with a timeout configuration, see Submitting a job.

Amazon EKS jobs

A job is the smallest unit of work in AWS Batch. An AWS Batch job on Amazon EKS has a one-to-one mapping to a Kubernetes pod. An AWS Batch job definition is a template for an AWS Batch job. When you submit an AWS Batch job, you reference a job definition, target a job queue, and provide a name for a job. In the job definition of an AWS Batch job on Amazon EKS, the eksProperties parameter defines the set of parameters that an AWS Batch on Amazon EKS job supports. In a SubmitJob request, the eksPropertiesOverride parameter allows for overrides to some common parameters. This way, you can use templates of job definitions for multiple jobs. When a job is dispatched to your Amazon EKS cluster, AWS Batch transforms the job into a podspec (Kind: Pod). The podspec uses some additional AWS Batch parameters to ensure that jobs are scaled and scheduled correctly. AWS Batch combines labels and taints to ensure jobs run only on AWS Batch managed nodes and that other pods don't run on those nodes.
**Important**

- If the `hostNetwork` parameter isn't explicitly set in an Amazon EKS job definition, the pod networking mode in AWS Batch defaults to host mode. More specifically, the following settings are applied: `hostNetwork=true` and `dnsPolicy=ClusterFirstWithHostNet`.

- AWS Batch cleans up job pods soon after a pod completes its job. To see pod application logs, configure a logging service for your cluster. For more information, see [Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs](#).

---

**Map a running job to a pod and a node**

The `podProperties` of a running job have `podName` and `nodeName` parameters set for the current job attempt. Use the [DescribeJobs](#) API operation to view these parameters.

The following is example output.

```bash
$ aws batch describe-jobs --job 2d044787-c663-4ce6-a6fe-f2baf7e51b04
{
  "jobs": [
    {
      "status": "RUNNING",
      "jobArn": "arn:aws:batch:us-east-1:123456789012:job/2d044787-c663-4ce6-a6fe-f2baf7e51b04",
      "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/MyJobOnEks_SleepWithRequestsOnly:1",
      "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/My-Eks-JQ1",
      "jobId": "2d044787-c663-4ce6-a6fe-f2baf7e51b04",
      "eksProperties": {
        "podProperties": {
          "nodeName": "ip-192-168-55-175.ec2.internal",
          "containers": [
            {
              "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
              "resources": {
                "requests": {
                  "cpu": "1",
                  "memory": "1024Mi"
                }
              }
            }
          ]
        }
      }
    }
  ]
}
```
For a job with retries enabled, the podName and nodeName of every completed attempt is in the eksAttempts list parameter of the DescribeJobs API operation. The podName and nodeName of the current running attempt are in the podProperties object.

**How to map a running pod back to its job**

A pod has labels that indicate the jobId and uuid of the compute environment that it belongs to. AWS Batch injects environment variables so the job’s runtime can reference job information. For more information, see AWS Batch job environment variables. You can view this information by running the following command. The output is as follows.

```
$ kubectl describe pod aws-batch.14638eb9-d218-372d-ba5c-1c9ab9c7f2a1 -n my-aws-batch-namespace
Name:         aws-batch.14638eb9-d218-372d-ba5c-1c9ab9c7f2a1
Namespace:    my-aws-batch-namespace
Priority:     0
Node:         ip-192-168-45-88.ec2.internal/192.168.45.88
Start Time:   Wed, 26 Oct 2022 00:30:48 +0000
Labels:       batch.amazonaws.com/compute-environment-uuid=5c19160b-d450-31c9-8454-86cf5b30548f
                batch.amazonaws.com/job-id=f980f2cf-6309-4c77-a2b2-d83fbbaf0e9f0
                batch.amazonaws.com/node-uid=a4be5c1d-9881-4524-b967-587789094647
...  
Status:       Running
IP:           192.168.45.88
IPs:
    IP:  192.168.45.88
Containers:
    default:
        Image:        public.ecr.aws/amazonlinux/amazonlinux:2
...  
Environment:
```
Features that AWS Batch Amazon EKS jobs support

These are the AWS Batch specific features that are also common to Kubernetes jobs that run on Amazon EKS:

- Job dependencies
- Array jobs
- Job timeouts
- Automated job retries
- Fair share scheduling

Kubernetes Secrets and ServiceAccounts

AWS Batch supports referencing Kubernetes Secrets and ServiceAccounts. You can configure pods to use Amazon EKS IAM roles for service accounts. For more information, see Configuring pods to use a Kubernetes service account in the Amazon EKS User Guide.

Related documents

- Memory and vCPU considerations for AWS Batch on Amazon EKS
- To create a GPU-based job on Amazon EKS resources
- Jobs stuck in a RUNNABLE status

Array jobs

An array job is a job that shares common parameters, such as the job definition, vCPUs, and memory. It runs as a collection of related yet separate basic jobs that might be distributed across multiple hosts and might run concurrently. Array jobs are the most efficient way to run extremely parallel jobs such as Monte Carlo simulations, parametric sweeps, or large rendering jobs.
AWS Batch array jobs are submitted just like regular jobs. However, you specify an array size (between 2 and 10,000) to define how many child jobs should run in the array. If you submit a job with an array size of 1000, a single job runs and spawns 1000 child jobs. The array job is a reference or pointer to manage all the child jobs. This way, you can submit large workloads with a single query. The timeout specified in the attemptDurationSeconds parameter applies to each child job. The parent array job does not have a timeout.

When you submit an array job, the parent array job gets a normal AWS Batch job ID. Each child job has the same base ID. However, the array index for the child job is appended to the end of the parent ID, such as `example_job_ID:0` for the first child job of the array.

The parent array job can enter a SUBMITTED, PENDING, FAILED, or SUCCEEDED status. An array parent job is updated to PENDING when any child job is updated to RUNNABLE. For more information about job dependencies, see [Job dependencies](#).

At runtime, the `AWS_BATCH_JOB_ARRAY_INDEX` environment variable is set to the container's corresponding job array index number. The first array job index is numbered 0, and subsequent attempts are in ascending order (for example, 1, 2, and 3). You can use this index value to control how your array job children are differentiated. For more information, see [Tutorial: Using the array job index to control job differentiation](#).

For array job dependencies, you can specify a type for a dependency, such as SEQUENTIAL or N_TO_N. You can specify a SEQUENTIAL type dependency (without specifying a job ID) so that each child array job completes sequentially, starting at index 0. For example, if you submit an array job with an array size of 100, and specify a dependency with type SEQUENTIAL, 100 child jobs are spawned sequentially, where the first child job must succeed before the next child job starts. The figure below shows Job A, an array job with an array size of 10. Each job in Job A's child index is dependent on the previous child job. Job A:1 can't start until job A:0 finishes.
You can also specify an N_TO_N type dependency with a job ID for array jobs. That way, each index child of this job must wait for the corresponding index child of each dependency to complete before it can begin. The following figure shows Job A and Job B, two array jobs with an array size of 10,000 each. Each job in Job B's child index is dependent on the corresponding index in Job A. Job B:1 can't start until job A:1 finishes.
If you cancel or terminate a parent array job, all of the child jobs are cancelled or terminated with it. You can cancel or terminate individual child jobs (which moves them to the FAILED status) without affecting the other child jobs. However, if a child array job fails (on its own or by cancelling or terminating manually), the parent job also fails.

**Example array job workflow**

A common workflow for AWS Batch customers is to run a prerequisite setup job, run a series of commands against a large number of input tasks, and then conclude with a job that aggregates results and writes summary data to Amazon S3, DynamoDB, Amazon Redshift, or Aurora.

For example:

- **JobA**: A standard, non-array job that performs a quick listing and metadata validation of objects in an Amazon S3 bucket, **BucketA**. The [SubmitJob](https://aws.amazon.com/batch/docs/) JSON syntax is as follows.

```json
{
    "jobName": "JobA",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobA",
    "jobArguments": [
        "--input",
        "BucketAzeitig",
        "--output",
        "BucketB-output",
        "--temp",
        "BucketB-temp"
    ],
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobA",
    "jobArguments": [
        "--input",
        "BucketAzeitig",
        "--output",
        "BucketB-output",
        "--temp",
        "BucketB-temp"
    ]
}
```
• JobB: An array job with 10,000 copies that is dependent upon JobA that runs CPU-intensive commands against each object in BucketA and uploads results to BucketB. The SubmitJob JSON syntax is as follows.

```json
{
    "jobName": "JobB",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobB-CPU-Intensive-Processing:1",
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "4096"
            },
            {
                "type": "VCPU",
                "value": "32"
            }
        ]
    },
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobA_job_ID"
        }
    ]
}
```

• JobC: Another 10,000 copy array job that's dependent upon JobB with an N_TO_N dependency model, that runs memory-intensive commands against each item in BucketB, writes metadata to DynamoDB, and uploads the resulting output to BucketC. The SubmitJob JSON syntax is as follows.

```json
{
    "jobName": "JobC",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobC-Memory-Intensive-Processing:1",
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "4096"
            },
            {
                "type": "VCPU",
                "value": "32"
            }
        ]
    },
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobA_job_ID"
        }
    ]
}
```
"containerOverrides": {
  "resourceRequirements": [
    {
      "type": "MEMORY",
      "value": "32768"
    },
    {
      "type": "VCPU",
      "value": "1"
    }
  ]
}
"arrayProperties": {
  "size": 10000
},
"dependsOn": [
  {
    "jobId": "JobB_job_ID",
    "type": "N_TO_N"
  }
]

• JobD: An array job that performs 10 validation steps that each need to query DynamoDB and might interact with any of the above Amazon S3 buckets. Each of the steps in JobD run the same command. However, the behavior is different based on the value of the AWS_BATCH_JOB_ARRAY_INDEX environment variable within the job’s container. These validation steps run sequentially (for example, JobD:0 and then JobD:1). The SubmitJob JSON syntax is as follows.

```json
{
  "jobName": "JobD",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobD-Sequential-Validation:1",
  "containerOverrides": {
    "resourceRequirements": [
      {
        "type": "MEMORY",
        "value": "32768"
      },
      {
        "type": "VCPU",
        "value": "1"
      }
    ]
  }
}
```
• JobE: A final, non-array job that performs some simple cleanup operations and sends an Amazon SNS notification with a message that the pipeline has completed and a link to the output URL. The SubmitJob JSON syntax is as follows.

```json
{
    "jobName": "JobE",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobE-Cleanup-and-Notification:1",
    "parameters": {
        "SourceBucket": "s3://JobD-Output-Bucket",
        "Recipient": "pipeline-notifications@mycompany.com"
    },
    "dependsOn": [
        {
            "jobId": "JobD_job_ID"
        }
    ]
}
```

**Tutorial: Using the array job index to control job differentiation**

This tutorial describes how to use the AWS_BATCH_JOB_ARRAY_INDEX environment variable to differentiate the child jobs. Each child job is assigned to this variable. The example uses the child job’s index number to read a specific line in a file. Then, it substitutes the parameter associated
with that line number with a command inside the job's container. The result is that you can have multiple AWS Batch jobs that run the same Docker image and command arguments. However, the results are different because the array job index is used as a modifier.

In this tutorial, you create a text file that has all of the colors of the rainbow, each on its own line. Then, you create an entrypoint script for a Docker container that converts the index into a value that can be used for a line number in the color file. The index starts at zero, but line numbers start at one. Create a Dockerfile that copies the color and index files to the container image and sets ENTRYPOINT for the image to the entrypoint script. The Dockerfile and resources are built to a Docker image that's pushed to Amazon ECR. You then register a job definition that uses your new container image, submit an AWS Batch array job with that job definition, and view the results.

**Prerequisites**

This tutorial has the following prerequisites:

- An AWS Batch compute environment. For more information, see [Creating a compute environment](#).
- An AWS Batch job queue and associated compute environment. For more information, see [Creating a job queue](#).
- The AWS CLI installed on your local system. For more information, see [Installing the AWS Command Line Interface](#) in the AWS Command Line Interface User Guide.
- Docker installed on your local system. For more information, see [About Docker CE](#) in the Docker documentation.

**Step 1: Build a container image**

You can use the AWS_BATCH_JOB_ARRAY_INDEX in a job definition in the command parameter. However, we recommend that you create a container image that uses the variable in an entrypoint script instead. This section describes how to create such a container image.

**To build your Docker container image**

1. Create a new directory to use as your Docker image workspace and navigate to it.
2. Create a file named colors.txt in your workspace directory and paste the following into it.

   ```
   red
   orange
   ```
3. Create a file named `print-color.sh` in your workspace directory and paste the following into it.

```bash
#!/bin/sh
LINE=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
COLOR=$(sed -n ${LINE}p /tmp/colors.txt)
echo My favorite color of the rainbow is $COLOR.
```

**Note**

The `LINE` variable is set to the `AWS_BATCH_JOB_ARRAY_INDEX + 1` because the array index starts at 0, but line numbers start at 1. The `COLOR` variable is set to the color in `colors.txt` that's associated with its line number.

4. Create a file named `Dockerfile` in your workspace directory and paste the following content into it. This Dockerfile copies the previous files to your container and sets the entrypoint script to run when the container starts.

```bash
FROM busybox
COPY print-color.sh /tmp/print-color.sh
COPY colors.txt /tmp/colors.txt
RUN chmod +x /tmp/print-color.sh
ENTRYPOINT /tmp/print-color.sh
```

5. Build your Docker image.

```bash
$ docker build -t print-color .
```

6. Test your container with the following script. This script sets the `AWS_BATCH_JOB_ARRAY_INDEX` variable to 0 locally and then increments it to simulate what an array job with seven children does.

```bash
$ AWS_BATCH_JOB_ARRAY_INDEX=0
```
while [ $AWS_BATCH_JOB_ARRAY_INDEX -le 6 ]
  do
    docker run -e AWS_BATCH_JOB_ARRAY_INDEX=$AWS_BATCH_JOB_ARRAY_INDEX print-color
    AWS_BATCH_JOB_ARRAY_INDEX=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
  done

The following is the output.

My favorite color of the rainbow is red.
My favorite color of the rainbow is orange.
My favorite color of the rainbow is yellow.
My favorite color of the rainbow is green.
My favorite color of the rainbow is blue.
My favorite color of the rainbow is indigo.
My favorite color of the rainbow is violet.

Step 2: Push your image to Amazon ECR

Now that you built and tested your Docker container, push it to an image repository. This example uses Amazon ECR, but you can use another registry, such as DockerHub.

1. Create an Amazon ECR image repository to store your container image. This example only uses the AWS CLI, but you can also use the AWS Management Console. For more information, see Creating a Repository in the Amazon Elastic Container Registry User Guide.

   $ aws ecr create-repository --repository-name print-color

2. Tag your print-color image with your Amazon ECR repository URI that was returned from the previous step.

   $ docker tag print-color aws_account_id.dkr.ecr.region.amazonaws.com/print-color

3. Log in to your Amazon ECR registry. For more information, see Registry Authentication in the Amazon Elastic Container Registry User Guide.

   $ aws ecr get-login-password \
      --region region | docker login \
      --username AWS \
      --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
4. Push your image to Amazon ECR.

$ docker push aws_account_id.dkr.ecr.region.amazonaws.com/print-color

**Step 3: Create and register a job definition**

Now that your Docker image is in an image registry, you can specify it in an AWS Batch job definition. Then, you can use it later to run an array job. This example only uses the AWS CLI. However, you can also use the AWS Management Console. For more information, see [Creating a single-node job definition](#).

**To create a job definition**

1. Create a file named `print-color-job-def.json` in your workspace directory and paste the following into it. Replace the image repository URI with your own image's URI.

   ```json
   {
   "jobDefinitionName": "print-color",
   "type": "container",
   "containerProperties": {
   "image": "aws_account_id.dkr.ecr.region.amazonaws.com/print-color",
   "resourceRequirements": [
   {
   "type": "MEMORY",
   "value": "250"
   },
   {
   "type": "VCPU",
   "value": "1"
   }
   ]
   }
   }
   ```

2. Register the job definition with AWS Batch.

   $ aws batch register-job-definition --cli-input-json file://print-color-job-def.json
Step 4: Submit an AWS Batch array job

After you registered your job definition, you can submit an AWS Batch array job that uses your new container image.

To submit an AWS Batch array job

1. Create a file named `print-color-job.json` in your workspace directory and paste the following into it.

   ```json
   {
   "jobName": "print-color",
   "jobQueue": "existing-job-queue",
   "arrayProperties": {
   "size": 7
   },
   "jobDefinition": "print-color"
   }
   ```

   **Note**

   This example uses the job queue mentioned in the [section called “Prerequisites”](#).

2. Submit the job to your AWS Batch job queue. Note the job ID that's returned in the output.

   ```
   $ aws batch submit-job --cli-input-json file://print-color-job.json
   ```

3. Describe the job's status and wait for the job to move to SUCCEEDED.

Step 5: View your array job logs

After your job reaches the SUCCEEDED status, you can view the CloudWatch Logs from the job's container.

To view your job's logs in CloudWatch Logs

2. In the left navigation pane, choose Jobs.
3. For **Job queue**, select a queue.
4. In the **Status** section, choose **succeeded**.
5. To display all of the child jobs for your array job, select the job ID that was returned in the previous section.
6. To see the logs from the job's container, select one of the child jobs and choose **View logs**.

7. View the other child job's logs. Each job returns a different color of the rainbow.

**Multi-node parallel jobs**

You can use multi-node parallel jobs to run single jobs that span multiple Amazon EC2 instances. With AWS Batch multi-node parallel jobs, you can run large-scale, high-performance computing applications and distributed GPU model training without the need to launch, configure, and manage Amazon EC2 resources directly. An AWS Batch multi-node parallel job is compatible with any framework that supports IP-based, internode communication. Examples include Apache MXNet, TensorFlow, Caffe2, or Message Passing Interface (MPI).

Multi-node parallel jobs are submitted as a single job. However, your job definition (or job submission node overrides) specifies the number of nodes to create for the job and what node groups to create. Each multi-node parallel job contains a **main node**, which is launched first. After the main node is up, the child nodes are launched and started. The job is finished only if the main node exits. All child nodes are then stopped. For more information, see [Node groups](#).

Multi-node parallel job nodes are single-tenant. This means that only a single job container is run on each Amazon EC2 instance.

The final job status (**SUCCEEDED** or **FAILED**) is determined by the final job status of the main node. To get the status of a multi-node parallel job, describe the job by using the job ID that was returned when you submitted the job. If you need the details for child nodes, describe each child
node individually. You can address nodes using the \( N \) notation (starting with 0). For example, to access the details of the second node of a job, describe `aws_batch_job_id#1` using the AWS Batch `DescribeJobs` API operation. The started, stoppedAt, statusReason, and exit information for a multi-node parallel job is populated from the main node.

If you specify job retries, a main node failure causes another attempt to occur. Child node failures don't cause more attempts to occur. Each new attempt of a multi-node parallel job updates the corresponding attempt of its associated child nodes.

To run multi-node parallel jobs on AWS Batch, your application code must contain the frameworks and libraries that are necessary for distributed communication.

**Environment variables**

At runtime, each node is configured the standard environment variables that all AWS Batch jobs receive. In addition, the nodes are configured with the following environment variables that are specific to multi-node parallel jobs:

AWS\_BATCH\_JOB\_MAIN\_NODE\_INDEX

This variable is set to the index number of the job's main node. Your application code can compare the `AWS\_BATCH\_JOB\_MAIN\_NODE\_INDEX` to the `AWS\_BATCH\_JOB\_NODE\_INDEX` on an individual node to determine if it's the main node.

AWS\_BATCH\_JOB\_MAIN\_NODE\_PRIVATE\_IPV4\_ADDRESS

This variable is only set in multi-node parallel job child nodes. This variable isn't present on the main node. This variable is set to the private IPv4 address of the job's main node. Your child node's application code can use this address to communicate with the main node.

AWS\_BATCH\_JOB\_NODE\_INDEX

This variable is set to the node index number of the node. The node index begins at 0, and each node receives a unique index number. For example, a multi-node parallel job with 10 children has index values of 0-9.

AWS\_BATCH\_JOB\_NUM\_NODES

This variable is set to the number of nodes that you have requested for your multi-node parallel job.
Node groups

A node group is an identical group of job nodes that all share the same container properties. You can use AWS Batch to specify up to five distinct node groups for each job.

Each group can have its own container images, commands, environment variables, and so on. For example, you can submit a job that requires a single c5.xlarge instance for the main node and five c5.xlarge instance child nodes. Each of these distinct node groups may specify different container images or commands to run for each job.

Alternatively, all of the nodes in your job can use a single node group. Moreover, your application code can differentiate node roles such as the main node and child node. It does this by comparing the AWS_BATCH_JOB_MAIN_NODE_INDEX environment variable against its own value for AWS_BATCH_JOB_NODE_INDEX. You can have up to 1,000 nodes in a single job. This is the default limit for instances in an Amazon ECS cluster. You can request to increase this limit.

Note
Currently all node groups in a multi-node parallel job must use the same instance type.

Job lifecycle

When you submit a multi-node parallel job, the job enters the SUBMITTED status. Then, the job waits for any job dependencies to finish. The job also moves to the RUNNABLE status. Last, AWS Batch provisions the instance capacity that's required to run your job and launches these instances.

Each multi-node parallel job contains a main node. The main node is a single subtask that AWS Batch monitors to determine the outcome of the submitted multi node job. The main node is launched first and it moves to the STARTING status. The timeout value specified in the attemptDurationSeconds parameter applies to the whole job and not to the nodes.

When the main node reaches the RUNNING status after the node's container is running, the child nodes are launched and they also move to the STARTING status. The child nodes come up in random order. There are no guarantees on the timing or ordering of child node launch. To ensure that the all the nodes of the jobs are in the RUNNING status after the node's container is running, your application code can query the AWS Batch API to get the main node and child node information. Alternatively, the application code can wait until all nodes are online before starting any distributed processing task. The private IP address of the main node is available
as the AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS environment variable in each child node. Your application code may use this information to coordinate and communicate data between each task.

As individual nodes exit, they move to SUCCEEDED or FAILED, depending on their exit code. If the main node exits, the job is considered finished, and all of the child nodes are stopped. If a child node dies, AWS Batch doesn't take any action on the other nodes in the job. If you don't want your job to continue with a reduced number of nodes, you must factor this into your application code. Doing this terminates or cancels the job.

**Compute environment considerations**

There are several things to consider when configuring compute environments to run multi-node parallel jobs with AWS Batch.

- Multi-node parallel jobs aren't supported on UNMANAGED compute environments.
- If you want to submit multi-node parallel jobs to a compute environment, create a *cluster* placement group in a single Availability Zone and associate it with your compute resources. This keeps your multi-node parallel jobs on a logical grouping of instances close with high network flow potential. For more information, see [Placement Groups](https://docs.aws.amazon.com/AmazonEC2/latest/UserGuide/instance-termination-cluster.html) in the *Amazon EC2 User Guide for Linux Instances*.
- Multi-node parallel jobs aren't supported on compute environments that use Spot Instances.
- AWS Batch multi-node parallel jobs use the Amazon ECS awsvpc network mode, which gives your multi-node parallel job containers the same networking properties as Amazon EC2 instances. Each multi-node parallel job container gets its own elastic network interface, a primary private IP address, and an internal DNS hostname. The network interface is created in the same VPC subnet as its host compute resource. Any security groups that are applied to your compute resources are also applied to it. For more information, see [Task Networking with the awsvpc Network Mode](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-networking.html) in the *Amazon Elastic Container Service Developer Guide*.
- Your compute environment might have no more than five security groups associated with it.
- The awsvpc network mode doesn't provide the elastic network interfaces for multi-node parallel jobs with public IP addresses. To access the internet, your compute resources must be launched in a private subnet that is configured to use a NAT gateway. For more information, see [NAT Gateways](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC-NAT-Gateway-Virtual-private-cloud.html) in the *Amazon VPC User Guide*. Internode communication must use the private IP address or DNS hostname for the node. Multi-node parallel jobs that run on compute resources within public subnets don't have outbound network access. To create a VPC with private subnets and a NAT gateway, see [Creating a virtual private cloud](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/).
• The elastic network interfaces that are created and attached to your compute resources can't be detached manually or modified by your account. This is to prevent the accidental deletion of an elastic network interface that's associated with a running job. To release the elastic network interfaces for a task, terminate the job.

• Your compute environment must have enough maximum vCPUs to support your multi-node parallel job.

• Your Amazon EC2 instance quota include the number of instances that's required to run your job. For example, suppose that your job requires 30 instances, but your account can only run 20 instances in a Region. Then, your job will get stuck in RUNNABLE status.

• If you specify an instance type for a node group in a multi-node parallel job, your compute environment must launch that instance type.

GPU jobs

GPU jobs help you to run jobs that use an instance's GPUs.

The following Amazon EC2 GPU-based instance types are supported. For more information, see Amazon EC2 G3 Instances, Amazon EC2 G4 Instances, Amazon EC2 G5 Instances, Amazon EC2 P2 Instances, Amazon EC2 P3 Instances, Amazon EC2 P4d Instances, and Amazon EC2 P5 Instances.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>GPUs</th>
<th>GPU memory</th>
<th>vCPUs</th>
<th>Memory</th>
<th>Network bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>g3s.xlarge</td>
<td>1</td>
<td>8 GiB</td>
<td>4</td>
<td>30.5 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>g3.4xlarge</td>
<td>1</td>
<td>8 GiB</td>
<td>16</td>
<td>122 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>g3.8xlarge</td>
<td>2</td>
<td>16 GiB</td>
<td>32</td>
<td>244 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>g3.16xlarge</td>
<td>4</td>
<td>32 GiB</td>
<td>64</td>
<td>488 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>g4dn.xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>4</td>
<td>16 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.2xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>8</td>
<td>32 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.4xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>16</td>
<td>64 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.8xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>32</td>
<td>128 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>Instance type</td>
<td>GPUs</td>
<td>GPU memory</td>
<td>vCPUs</td>
<td>Memory</td>
<td>Network bandwidth</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
<td>------------</td>
<td>-------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>g4dn.12xlarge</td>
<td>4</td>
<td>64 GiB</td>
<td>48</td>
<td>192 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>g4dn.16xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>64</td>
<td>256 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>g5.xlarge</td>
<td>1</td>
<td>24 GiB</td>
<td>4</td>
<td>16 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>g5.2xlarge</td>
<td>1</td>
<td>24 GiB</td>
<td>8</td>
<td>32 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>g5.4xlarge</td>
<td>1</td>
<td>24 GiB</td>
<td>16</td>
<td>64 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g5.8xlarge</td>
<td>1</td>
<td>24 GiB</td>
<td>32</td>
<td>128 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>g5.16xlarge</td>
<td>1</td>
<td>24 GiB</td>
<td>64</td>
<td>256 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>g5.12xlarge</td>
<td>4</td>
<td>96 GiB</td>
<td>48</td>
<td>192 GiB</td>
<td>40 Gbps</td>
</tr>
<tr>
<td>g5.24xlarge</td>
<td>4</td>
<td>96 GiB</td>
<td>96</td>
<td>384 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>g5.48xlarge</td>
<td>8</td>
<td>192 GiB</td>
<td>192</td>
<td>768 GiB</td>
<td>100 Gbps</td>
</tr>
<tr>
<td>p2.xlarge</td>
<td>1</td>
<td>12 GiB</td>
<td>4</td>
<td>61 GiB</td>
<td>High</td>
</tr>
<tr>
<td>p2.8xlarge</td>
<td>8</td>
<td>96 GiB</td>
<td>32</td>
<td>488 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>p2.16xlarge</td>
<td>16</td>
<td>192 GiB</td>
<td>64</td>
<td>732 GiB</td>
<td>20 Gbps</td>
</tr>
<tr>
<td>p3.2xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>8</td>
<td>61 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>p3.8xlarge</td>
<td>4</td>
<td>64 GiB</td>
<td>32</td>
<td>244 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>p3.16xlarge</td>
<td>8</td>
<td>128 GiB</td>
<td>64</td>
<td>488 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>p3dn.24xlarge</td>
<td>8</td>
<td>256 GiB</td>
<td>96</td>
<td>768 GiB</td>
<td>100 Gbps</td>
</tr>
<tr>
<td>p4d.24xlarge</td>
<td>8</td>
<td>320 GiB</td>
<td>96</td>
<td>1152 GiB</td>
<td>4x100 Gbps</td>
</tr>
<tr>
<td>p5.48xlarge</td>
<td>8</td>
<td>640 GiB</td>
<td>192</td>
<td>2 TiB</td>
<td>32x100 Gbps</td>
</tr>
</tbody>
</table>
Note

Only instance types that support a NVIDIA GPU and use an x86_64 architecture are supported for GPU jobs in AWS Batch. For example, the G4ad and G5g instance families aren't supported.

The `resourceRequirements` parameter for the job definition specifies the number of GPUs to be pinned to the container. This number of GPUs isn't available to any other job that runs on that instance for the duration of that job. All instance types in a compute environment that run GPU jobs must be from the p2, p3, p4, p5, g3, g3s, g4, or g5 instance families. If this isn't done a GPU job might get stuck in the RUNNABLE status.

Jobs that don't use the GPUs can be run on GPU instances. However, they might cost more to run on the GPU instances than on similar non-GPU instances. Depending on the specific vCPU, memory, and time needed, these non-GPU jobs might block GPU jobs from running.

To create a GPU-based job on Amazon EKS resources

This section covers how to run an Amazon EKS GPU workload on AWS Batch.

Contents

- To create GPU-based Kubernetes cluster on Amazon EKS
- To create an Amazon EKS GPU job definition
- To run a GPU job in your Amazon EKS cluster

To create GPU-based Kubernetes cluster on Amazon EKS

Before you create a GPU-based Kubernetes cluster on Amazon EKS, you must have completed the steps in Getting started with AWS Batch on Amazon EKS. In addition, also consider the following:

- AWS Batch supports instance types with NVIDIA GPUs.
- By default, AWS Batch selects the Amazon EKS accelerated AMI with the Kubernetes version that matches your Amazon EKS cluster control plane version.

```
$ cat <<EOF > ./batch-eks-gpu-ce.json
```
AWS Batch doesn’t manage the NVIDIA GPU device plugin on your behalf. You must install this plugin into your Amazon EKS cluster and allow it to target the AWS Batch nodes. For more information, see [Enabling GPU Support in Kubernetes](https://github.com) on GitHub.

To configure the NVIDIA device plugin (DaemonSet) to target the AWS Batch nodes, run the following commands.

```
# pull nvidia daemonset spec
$ curl -O https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.12.2/nvidia-device-plugin.yml
# using your favorite editor, add Batch node toleration
# this will allow the DaemonSet to run on Batch nodes
- key: "batch.amazonaws.com/batch-node"
```

To create GPU-based Kubernetes cluster on Amazon EKS
We do not recommend that you mix compute-based (CPU and memory) workloads with GPU-based workloads in the same pairings of compute environment and job queue. This is because compute jobs can use up GPU capacity.

To attach job queues, run the following commands.

```bash
$ cat <<EOF > ./batch-eks-gpu-jq.json
{
  "jobQueueName": "My-Eks-GPU-JQ1",
  "priority": 10,
  "computeEnvironmentOrder": [
    {
      "order": 1,
      "computeEnvironment": "My-Eks-GPU-CE1"
    }
  ]
}
EOF
$ aws batch create-job-queue --cli-input-json file:///./batch-eks-gpu-jq.json
```

**To create an Amazon EKS GPU job definition**

Only nvidia.com/gpu is supported at this time and resource value that you set must be a whole number. You can't use fractions of GPU. For more information, see Schedule GPUs in the Kubernetes documentation.

To register a GPU job definition for Amazon EKS, run the following commands.

```bash
$ cat <<EOF > ./batch-eks-gpu-jd.json
{
  "jobDefinitionName": "MyGPUJobOnEks_Smi",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "hostNetwork": true,
      "containers": [
```
To run a GPU job in your Amazon EKS cluster

The GPU resource is non-compressible. AWS Batch creates a pod spec for GPU jobs where the value of request equals the value of limits. This is a Kubernetes requirement.

To submit a GPU job, run the following commands.

```bash
$ aws batch register-job-definition --cli-input-json file://./batch-eks-gpu-jd.json

$ aws batch submit-job --job-queue My-Eks-GPU-JQ1 --job-definition MyGPUJobOnEks_Smi --job-name My-Eks-GPU-Job

# locate information that can help debug or find logs (if using Amazon CloudWatch Logs with Fluent Bit)
$ aws batch describe-jobs --job <job-id> | jq '.jobs[].eksProperties.podProperties | {podName, nodeName}'
```

Search and filter AWS Batch jobs

You can list the jobs in a job queue using the AWS Batch console. However, if there are many jobs in the job queue, it might be difficult to find a specific job.
You can use **Searching and filtering** to list jobs that match search criteria that you specify.

1. Open the [AWS Batch console](https://aws.amazon.com/batch).  
2. Choose **Jobs**.  
3. Turn on **Searching and filtering**.

   ❌ **Note**  
   If you have several jobs, this process might take a few minutes.

4. In the **Please select a job queue** box, choose the job queue that you want to search.  
5. In the **Filter resources by property or value** box, choose one of the listed properties.  
6. Choose the operator that you want to use. For example, choose **Status** =.  

   ❌ **Tip**  
   To use a different property or operator, close the current criteria. Then, choose the property and operator that you want.

7. Enter or choose a property value. For example, enter all or part of a job name or choose **Status** = **RUNNABLE**.  
8. Choose the job that you want in the filtered list.  

   ❌ **Tip**  
   If you don’t see the job that you want, scroll the filtered list.

### Job logs

You can configure your AWS Batch jobs to send log information to CloudWatch Logs. This way, you can view different logs from your jobs in one convenient location. For more information, see [Using CloudWatch Logs with AWS Batch](https://aws.amazon.com/batch).  

You can also use **Job logs** in the AWS Batch console to monitor or troubleshoot an AWS Batch job.
1. Open the AWS Batch console.

2. Choose Jobs.

3. For Job queue, choose the job queue that you want.

   **Tip**
   If there are several jobs in the job queue, you can turn on Searching and filtering to find a job faster. For more information, see Search and filter AWS Batch jobs.

4. For Status, choose the job status that you want.

5. Choose the job that you want.

6. On the Details page, scroll down to Job Logs.

7. Choose Retrieve logs.

8. For Authorization required, enter OK, and then choose Authorize to accept Amazon CloudWatch charges.

   **Note**
   To revoke your authorization for CloudWatch charges:
   1. In the left navigation pane, choose Permissions.
   2. For Job logs, choose Edit.
   3. Clear the Authorize Batch to use CloudWatch check box.
   4. Choose Save changes.

9. Review the log data for the AWS Batch job.

   **Tip**
   You can filter the log based on Keywords, Max results, and Sorting. You can also choose one of the default time intervals or create a custom interval to customize the results.
Job information

You can review AWS Batch job information such as status, job definition and container information.

1. Open the [AWS Batch console](https://aws.amazon.com/batch/).
2. Choose Jobs.
3. For Job queue, choose the job queue that you want.

   **Tip**
   If there are several jobs in the job queue, you can turn on **Searching and filtering** to find a job faster. For more information, see [Search and filter AWS Batch jobs](https://docs.aws.amazon.com/batch/latest/userguide/).  

4. Choose the job that you want.

   **Note**
   You can also use the AWS Command Line Interface (AWS CLI) to view details about an AWS Batch job. For more information, see [describe-jobs](https://docs.aws.amazon.com/cli/latest/reference/batch/describe-jobs) in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/batch/describe-jobs).
Job definitions

AWS Batch job definitions specify how jobs are to be run. While each job must reference a job definition, many of the parameters that are specified in the job definition can be overridden at runtime.

Contents

- Creating a single-node job definition
- Creating a multi-node parallel job definition
- Creating job definitions using ContainerProperties
- Creating job definitions using EcsProperties
- Using the awslogs log driver
- Specifying sensitive data
- Private registry authentication for jobs
- Amazon EFS volumes
- Example job definitions

Some of the attributes specified in a job definition include:

- Which Docker image to use with the container in your job
- How many vCPUs and how much memory to use with the container
- The command the container should run when it is started
- What (if any) environment variables should be passed to the container when it starts
- Any data volumes that should be used with the container
- What (if any) IAM role your job should use for AWS permissions

For a complete description of the parameters available in a job definition, see Job definition parameters for ContainerProperties.
Creating a single-node job definition

Before you can run jobs in AWS Batch, you must create a job definition. This process varies slightly between single-node and multi-node parallel jobs. This topic covers specifically how to create a job definition for an AWS Batch job that's not a multi-node parallel job.

You can create a multi-node parallel job definition on Amazon Elastic Container Service resources. For more information, see the section called “Creating a multi-node parallel job definition”.

Topics

- Creating a single-node job definition on Amazon EC2 resources
- Creating a single-node job definition on AWS Fargate resources
- Creating a single-node job definition on Amazon EKS resources

Creating a single-node job definition on Amazon EC2 resources

To create a new job definition on Amazon EC2 resources:

2. From the navigation bar, choose the AWS Region to use.
3. In the left navigation pane, choose Job definitions.
4. Choose Create.
5. For Orchestration type, choose Amazon Elastic Compute Cloud (Amazon EC2).
6. For EC2 platform configuration, turn off Enable multi-node parallel processing.
7. For Name, enter a unique name for your job definition. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
8. (Optional) For Execution timeout, enter the timeout value (in seconds). The execution timeout is the length of time before an unfinished job is terminated. If an attempt exceeds the timeout duration, the attempt is stopped and moves to a FAILED status. For more information, see Job timeouts. The minimum value is 60 seconds.
9. (Optional) Turn on Scheduling priority. Enter a scheduling priority value between 0 and 100. Higher values are given higher priority.
10. (Optional) For Job attempts, enter the number of times that AWS Batch attempts to move the job to RUNNABLE status. Enter a number between 1 and 10.
11. (Optional) For **Retry strategy conditions**, choose **Add evaluate on exit**. Enter at least one parameter value and then choose an **Action**. For each set of conditions, **Action** must be set to either **Retry** or **Exit**. These actions mean the following:

- **Retry** – AWS Batch retries until the number of job attempts that you specified is reached.
- **Exit** – AWS Batch stops retrying the job.

**Important**

If you choose **Add evaluate on exit**, you must configure at least one parameter and either choose an **Action** or choose **Remove evaluate on exit**.

12. (Optional) Expand **Tags** and then choose **Add tag** to add tags to the resource. Enter a key and optional value, then choose **Add tag**.

13. (Optional) Turn on **Propagate tags** to propagate tags from the job and job definition to the Amazon ECS task.

14. Choose **Next page**.

15. In the **Container configuration** section:

   a. For **Image**, choose the Docker image to use for your job. By default, images in the Docker Hub registry are available. You can also specify other repositories with `repository-url/image:tag`. The name can be up to 225 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), forward slashes (/), and number signs (#). This parameter maps to **Image** in the **Create a container** section of the **Docker Remote API** and the **IMAGE** parameter of **docker run**.

   **Note**

   Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, Arm based Docker images can only run on Arm based compute resources.

   - Images in Amazon ECR Public repositories use the full `registry/repository[:tag]` or `registry/repository[@digest]` naming conventions (for example, `public.ecr.aws/registry_alias/my-web-app:latest`).

Creating a single-node job definition on Amazon EC2 resources
Images in Amazon ECR repositories use the full registry/repository[:tag] naming convention (for example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`).

Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).

Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).

Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

b. For **Command syntax**, choose **Bash** or **JSON**.

c. For **Command**, specify the command to pass to the container. For simpler commands, enter the command as you do for a command prompt. Then, verify that the JSON result is correct and passed to the Docker daemon. For more complicated commands (for example, with special characters), use **JSON** syntax.

### Tip
Choose **Info** to view Bash and JSON code samples.

This parameter maps to `Cmd` in the **Create a container** section of the [Docker Remote API](https://docs.docker.com/engine/reference/builder/#cmd) and the `COMMAND` parameter to `docker run`. For more information about the Docker `CMD` parameter, see [https://docs.docker.com/engine/reference/builder/#cmd](https://docs.docker.com/engine/reference/builder/#cmd).

### Note
You can use default values for parameter substitution and placeholders in your command. For more information, see [Parameters](https://docs.aws.amazon.com/batch/latest/userguide/parameters.html).

d. (Optional) For **Execution role**, specify an IAM role that grants the Amazon ECS container agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for tasks. For more information, see [Amazon ECS task execution IAM roles](https://docs.aws.amazon.com/ecs/latest/userguide/iam-roles-for-tasks.html) in the *Amazon Elastic Container Service Developer Guide*.

e. For **Job Role configuration**, choose an IAM role that has permissions to the AWS APIs. This feature uses Amazon ECS IAM roles for tasks. For more information, see [IAM Roles for Tasks](https://docs.aws.amazon.com/ecs/latest/userguide/iam-roles-for-tasks.html) in the *Amazon Elastic Container Service Developer Guide*. 
16. For **Parameters**, choose **Add parameters** to add parameter substitution placeholders as **Key** and optional **Value** pairs.

17. In the **Environment configuration** section:
   
a. For **vCPUs**, enter the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the **Create a container** section of the **Docker Remote API** and the --cpu-shares option to **docker run**. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

   b. For **Memory**, enter the memory limit available to the container. If your container attempts to exceed the amount of memory that you specify here, the container is stopped. This parameter maps to Memory in the **Create a container** section of the **Docker Remote API** and the --memory option to **docker run**. You must specify at least 4 MiB of memory for a job.

   c. For **Number of GPUs**, choose the number of GPUs to reserve for the container.

   d. (Optional) For **Environment variables**, choose **Add environment variable** to add environment variables as name-value pairs. These variables are passed to the container.

   e. (Optional) For **Secrets**, choose **Add secret** to add secrets as a name-value pairs. These secrets are exposed in the container. For more information, see **secretOptions** in **Job definition parameters for ContainerProperties**.

18. Choose **Next page**.

19. In the **Linux configuration** section:
a. For **User**, enter the user name to use inside the container. This parameter maps to `User` in the Create a container section of the Docker Remote API and the `--user` option to `docker run`.

b. (Optional) To give the job container elevated permissions on the host instance (similar to the root user), drag the **Privileged** slider to the right. This parameter maps to Privileged in the Create a container section of the Docker Remote API and the `--privileged` option to `docker run`.

c. (Optional) Turn on **Enable init process** to run an init process inside the container. This process forwards signals and reaps processes.

20. (Optional) In the **Filesystem configuration** section:

   a. Turn on **Enable read only filesystem** to remove write access to the volume.

   b. For **Shared memory size**, enter the size (in MiB) of the `/dev/shm` volume.

   c. For **Max swap size**, enter the total amount of swap memory (in MiB) that the container can use.

   d. For **Swappiness** enter a value between 0 and 100 to indicate the swappiness behavior of the container. If you don't specify a value and swapping is enabled, the value defaults to 60. For more information, see `swappiness` in Job definition parameters for ContainerProperties.

   e. (Optional) Expand **Additional configuration**.

   f. (Optional) For **Tmpfs**, choose Add tmpfs to add a tmpfs mount.

   g. (Optional) For **Devices**, choose Add device to add a device:

      i. For **Container path**, specify the path of in the container instance to expose the device mapped to the host instance. If you keep this blank, the host path is used in the container.

      ii. For **Host path**, specify the path of a device in the host instance.

      iii. For **Permissions**, choose one or more permissions to apply to the device. The available permissions are **READ**, **WRITE**, and **MKNOD**.

   h. (Optional) For **Volumes configuration**, choose Add volume to create a list of volumes to pass to the container. Enter **Name** and **Source path** for the volume and then choose Add volume. You can also choose to turn on **Enable EFS**.

   i. (Optional) For **Mount points**, choose Add mount points configuration to add mount points for data volumes. You must specify the source volume and container path. These
mount points are passed to the Docker daemon on a container instance. You can also choose to make the volume Read only.

j. (Optional) For **Ulimits configuration**, choose **Add ulimit** to add a ulimits value for the container. Enter **Name**, **Soft limit**, and **Hard limit** values, and then choose **Add ulimit**.

21. (Optional) In the **Logging configuration** section:

a. For **Log driver**, choose the log driver to use. For more information about the available log drivers, see **logDriver** in **Job definition parameters for ContainerProperties**.

   ![Note](image)

   > By default, the awslogs log driver is used.

b. For **Options**, choose **Add option** to add an option. Enter a name-value pair, and then choose **Add option**.

c. For **Secrets**, choose **Add secret**. Enter a name-value pair and then choose **Add secret** to add a secret.

   ![Tip](image)

   > For more information, see **secretOptions** in **Job definition parameters for ContainerProperties**.

22. Choose **Next page**.

23. For **Job definition review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create job definition**.

### Creating a single-node job definition on AWS Fargate resources

**To create a new job definition on AWS Fargate resources:**


2. From the top navigation bar, choose the AWS Region to use.

3. In the left navigation pane, choose **Job definitions**.

4. Choose **Create**.
5. For **Orchestration type**, choose **Fargate**. For more information, see [AWS Batch on AWS Fargate](#).

6. For **Name**, enter a unique name for your job definition. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

7. (Optional) For **Execution timeout**, enter the timeout value (in seconds). The execution timeout is the length of time before an unfinished job is terminated. If an attempt exceeds the timeout duration, the attempt is stopped and moves to a FAILED status. For more information, see [Job timeouts](#). The minimum value is 60 seconds.

8. (Optional) Turn on **Scheduling priority**. Enter a scheduling priority value between 0 and 100. Higher values are given higher priority over lower values.

9. (Optional) Expand **Tags**, and then choose **Add tag** to add tags to the resource. Turn on **Propagate tags** to propagate tags from the job and job definition.

10. In the **Fargate platform configuration** section:

    a. For **Runtime platform**, choose the compute environment architecture.

    b. For **Operating System Family**, choose the operating system for the compute environment.

    c. For **CPU Architecture**, choose the vCPU architecture.

    d. For **Fargate platform version**, enter LATEST or a specific runtime environment version.

    e. (Optional) Turn on **Assign public IP** to assign a public IP address to a Fargate job network interface. For a job that's running in a private subnet to send outbound traffic to the internet, the private subnet requires a NAT gateway be attached to route requests to the internet. You might want to do this so that you can pull container images. For more information, see [Amazon ECS task networking](#) in the [Amazon Elastic Container Service Developer Guide](#).

    f. (Optional) For **Ephemeral storage**, enter the amount of ephemeral storage to allocate to the task. The amount of ephemeral storage must be between 21 GiB and 200 GiB. By default, 20 GiB of ephemeral storage is allocated if you don’t enter a value.

    > **Note**
    > Ephemeral storage requires Fargate platform version 1.4 or later.
g. For **Execution role**, specify an IAM role that grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for task functionality. For more information including configuration prerequisites, see [Amazon ECS task execution IAM roles](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-execution-iam-roles.html) in the *Amazon Elastic Container Service Developer Guide*.

h. For **Job attempts**, enter the number of times that AWS Batch attempts to move the job to a **RUNNABLE** status. Enter a number between 1 and 10.

i. Optional) For **Retry strategy conditions**, choose **Add evaluate on exit**. Enter at least one parameter value and then choose an **Action**. For each set of conditions, **Action** must be set to either **Retry** or **Exit**. These actions mean the following:

- **Retry** – AWS Batch retries until the number of job attempts that you specified is reached.
- **Exit** – AWS Batch stops retrying the job.

⚠️ **Important**

If you choose **Add evaluate on exit**, you must configure at least one parameter and choose an **Action** or choose **Remove evaluate on exit**.

11. Choose **Next page**.

12. In the **Container configuration** section:

a. For **Image**, choose the Docker image to use for your job. By default, images in the Docker Hub registry are available. You can also specify other repositories with `repository-url/image:tag`. The name can be up to 225 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), periods (.), forward slashes (/), and number signs (#). This parameter maps to Image in the **Create a container** section of the [Docker Remote API](https://docs.docker.com/engine/reference/api/docker_remote_api/) and the IMAGE parameter of `docker run`.

ℹ️ **Note**

Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, Arm based Docker images can only run on Arm based compute resources.
• Images in Amazon ECR Public repositories use the full `registry/repository[:tag]` or `registry/repository[@digest]` naming conventions (for example, `public.ecr.aws/registry_alias/my-web-app:latest`).

• Images in Amazon ECR repositories use the full `registry/repository[:tag]` naming convention (for example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`).

• Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).

• Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).

• Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

b. For **Command syntax**, choose **Bash** or **JSON**.

c. For **Command**, specify the command to pass to the container. For simple commands, enter the command as you do for a command prompt and then verify that the JSON result is correct. It's passed to the Docker daemon. For more complicated commands (for example, with special characters), use **JSON** syntax.

   ![Tip](image)
   
   Choose **Info** to view Bash and JSON code samples.

This parameter maps to `Cmd` in the **Create a container** section of the [Docker Remote API](https://docs.docker.com/engine/reference/builder/#cmd) and the `COMMAND` parameter to **docker run**. For more information about the Docker `CMD` parameter, see [https://docs.docker.com/engine/reference/builder/#cmd](https://docs.docker.com/engine/reference/builder/#cmd).

![Note](image)

You can use default values for parameter substitution and placeholders in your command. For more information, see [Parameters](#parameters).

d. (Optional) Add parameters to the job definition as name-value mappings to override the job definition defaults. To add a parameter:
• For **Parameters**, choose **Add parameters**, enter a name-value pair, then choose **Add parameter**.

⚠ **Important**
If you choose **Add parameter**, you must either configure at least one parameter or choose **Remove parameter**

e. In the **Environment configuration** section:

i. For **Job role configuration**, choose an IAM role that has permissions to the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information, see [IAM Roles for Tasks](#) in the *Amazon Elastic Container Service Developer Guide*.

⚠ **Note**
Only roles that have the **Amazon Elastic Container Service Task Role** trust relationship are shown here. For more information about how to create an IAM role for your AWS Batch jobs, see [Creating an IAM Role and Policy for your Tasks](#) in the *Amazon Elastic Container Service Developer Guide*.

ii. For **vCPUs**, enter the number of vCPUs to reserve for the container. This parameter maps to `CpuShares` in the **Create a container** section of the *Docker Remote API* and the `--cpu-shares` option to `docker run`. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

iii. For **Memory**, enter the memory limit that's available to the container. If your container attempts to exceed the memory specified here, the container is stopped. This parameter maps to `Memory` in the **Create a container** section of the *Docker Remote API* and the `--memory` option to `docker run`. You must specify at least 4 MiB of memory for a job.

If you use GuardDuty Runtime Monitoring, there is a slight memory overhead for the GuardDuty security agent. Therefore the memory limit must include the size of the GuardDuty security agent. For information about the GuardDuty security agent memory limits, see [CPU and memory limits](#) in the *GuardDuty User Guide*. For information about the best practices, see [How do I remediate out of memory errors](#).
on my Fargate tasks after enabling Runtime Monitoring in the Amazon ECS Developer Guide.

Note
To maximize your resource utilization, prioritize memory for jobs of a specific instance type. For more information, see Compute Resource Memory Management.

f. (Optional) For Environment variables, choose Add environment variable to add environment variables as name-value pairs. These variables are passed to the container.

g. (Optional) For Secrets, choose Add secret to add secrets as a name-value pairs. These secrets are exposed in the container. For more information, see secretOptions in Job definition parameters for ContainerProperties.

h. Choose Next page.

13. (Optional) In the Linux configuration section:

a. For User, enter a user name to use inside the container.

b. Turn on Enable init process to run an init process inside the container. This process forwards signals and reaps processes.

c. Turn on Enable read only filesystem to remove write access to the volume.

d. (Optional) Expand Additional configuration.

e. For Mount points configuration, choose Add mount points configuration to add mount points for data volumes. You must specify the source volume and container path. These mount points are passed to the Docker daemon on a container instance.

f. For Volumes configuration, choose Add volume to create a list of volumes to pass to the container. Enter a Name and Source path for the volume, and then choose Add volume.

g. In the Logging configuration section:

i. (Optional) For Log driver, choose the log driver to use. For more information about the available log drivers, see logDriver in Job definition parameters for ContainerProperties.
**Note**

By default, the awslogs log driver is used.

ii.  
(Optional) For **Options**, choose **Add option** to add an option. Enter a name-value pair, and then choose **Add option**.

iii.  
(Optional) For **Secrets**, choose **Add secret** to add a secret. Then, enter a name-value pair, and choose **Add secret**.

**Tip**

For more information, see `secretOptions` in **Job definition parameters for ContainerProperties**.

14. Choose **Next page**.

15. For **Job definition review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create job definition**.

---

### Creating a single-node job definition on Amazon EKS resources

**To create a new job definition on Amazon Elastic Kubernetes Service resources:**


2. From the top navigation bar, choose the AWS Region to use.

3. In the left navigation pane, choose **Job definitions**.

4. Choose **Create**.

5. For **Orchestration type**, choose **Elastic Kubernetes Service (EKS)**.

6. For **Name**, enter a unique name for your job definition. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

7. (Optional) For **Execution timeout**, enter the timeout value (in seconds). The execution timeout is the length of time before an unfinished job is terminated. If an attempt exceeds the timeout duration, the attempt is stopped and moves to a FAILED status. For more information, see **Job timeouts**. The minimum value is 60 seconds.
8.  (Optional) Turn on **Scheduling priority**. Enter a scheduling priority value between 0 and 100. Higher values are given higher priority over lower values.

9.  (Optional) Expand **Tags**, and then choose **Add tag** to add tags to the resource.

10. Choose **Next page**.

11. In the **EKS pod properties** section:

    a. For **Service account name**, enter an account that provides an identity for processes that run in a pod.

    b. Turn **Host network** on to use the Kubernetes pod network model and open a listening port for incoming connections. Turn this setting off for outgoing communications only.

    c. For **DNS policy**, choose one of the following:

        • **No value (null)** – The pod ignores the DNS settings from the Kubernetes environment.
        • **Default** – The pod inherits the name resolution configuration from the node that it runs on.

        **Note**
        If a DNS policy isn't specified, **Default** isn't the default DNS policy. Instead, **ClusterFirst** is used.

        • **ClusterFirst** – Any DNS query that doesn't match the configured cluster domain suffix is forwarded to the upstream nameserver that's inherited from the node.
        • **ClusterFirstWithHostNet** – Use if **Host network** is turned on.

    d. (Optional) For **Pod labels**, choose **Add pod labels**, then enter a name-value pair.

        **Important**
        The prefix for a pod label can't contain `kubernetes.io/`, `k8s.io/`, or `batch.amazonaws.com/`.

    e. Choose **Next page**.

    f. In the **Container configuration** section:

        i. For **Name**, enter a unique name for the container. The name must start with a letter or number, and can be up to 63 characters long. It can contain uppercase and lowercase letters, numbers, and hyphens (-).
ii. For **Image**, choose the Docker image to use for your job. By default, images in the Docker Hub registry are available. You can also specify other repositories with `repository-url/image:tag`. The name can be up to 255 characters long. It can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), periods (.), forward slashes (/), and number signs (#). This parameter maps to Image in the [Create a container](#) section of the [Docker Remote API](#) and the IMAGE parameter of `docker run`.

```plaintext
Note

Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, Arm based Docker images can only run on Arm based compute resources.
```

- Images in Amazon ECR Public repositories use the full `registry/repository[:tag]` or `registry/repository[@digest]` naming conventions (for example, `public.ecr.aws/registry_alias/my-web-app:latest`).

- Images in Amazon ECR repositories use the full `registry/repository[:tag]` naming convention (for example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`).

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).

- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).

- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

iii. (Optional) For **Image pull policy**, choose when images are pulled.

iv. (Optional) For **Command**, enter a Bash or JSON command to pass to the container.

v. (Optional) For **Arguments**, enter arguments to pass to the container. If an argument isn't provided, the container image command is used.

g. (Optional) You can add parameters to the job definition as name-value mappings to override the job definition defaults. To add a parameter:

- For **Parameters**, enter a name-value pair, then choose **Add parameter**.
Important
If you choose Add parameter, you must configure at least one parameter or choose Remove parameter.

h. In the Environment configuration section:

i. For vCPUs, enter the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to docker run. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

ii. For Memory, enter the memory limit available to the container. If your container attempts to exceed the memory specified here, the container is stopped. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job.

Note
To maximize your resource utilization, prioritize memory for jobs of a specific instance type. For more information, see Compute Resource Memory Management.

i. (Optional) For Environment variables, choose Add environment variable to add environment variables as name-value pairs. These variables are passed to the container.

j. (Optional) For Volume mount:

i. Choose Add volume mount.

ii. Enter a Name, and then enter a Mount path in the container where the volume is mounted.

iii. Choose Read only to remove write permissions to the volume.

iv. Choose Add volume mount.

k. (Optional) For Run as user, enter a user ID to run the container process.
l. (Optional) For Run as group, enter a group ID to run the container process runtime.

m. (Optional) To give your job's container elevated permissions on the host instance (similar to the root user), drag the Privileged slider to the right. This parameter maps to Privileged in the Create a container section of the Docker Remote API and the --privileged option to docker run.

n. (Optional) Turn on Read-only root filesystem to remove write access to the root filesystem.

o. (Optional) Turn on Run as non-root to run the containers in the pod as a non-root user.

p. Choose Next page.

12. For Job definition review, review the configuration steps. If you need to make changes, choose Edit. When you're finished, choose Create job definition.

Creating a multi-node parallel job definition

Before you can run jobs in AWS Batch, you must create a job definition. This process varies slightly between single-node and multi-node parallel jobs. This topic covers specifically how to create a job definition for an AWS Batch multi-node parallel job. For more information, see Multi-node parallel jobs.
Note
AWS Fargate doesn't support multi-node parallel jobs.

Creating a multi-node parallel job definition on Amazon EC2 resources

To create a single-node job definition, see Creating a single-node job definition .

To create a multi-node parallel job definition on Amazon Elastic Compute Cloud resources:

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Job definitions.
4. Choose Create.
5. For Orchestration type, choose Amazon Elastic Compute Cloud (Amazon EC2).
6. For Enable multi-node parallel, turn on multi-node parallel.
7. For Name, enter a unique name for your job definition. The name can be up to 128 characters long, and can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
8. (Optional) For Execution timeout, specify the maximum number of seconds that you want job attempts to run. If an attempt exceeds the timeout duration, the attempt is stopped and moves to a FAILED status. For more information, see Job timeouts.
9. (Optional) Turn on Scheduling priority. Enter a scheduling priority value between 0 and 100. Higher values are given higher priority over lower values.
10. (Optional) For Job attempts, enter the number of times that AWS Batch attempts to move the job to RUNNABLE status. Enter a number between 1 and 10.
11. (Optional) For Retry strategy conditions, choose Add evaluate on exit. Enter at least one parameter value and then choose an Action. For each set of conditions, Action must be set to either Retry or Exit. These actions mean the following:
   - Retry – AWS Batch retries until the number of job attempts that you specified is reached.
   - Exit – AWS Batch stops retrying the job.
Important

If you choose **Add evaluate on exit**, you must configure at least one parameter and either choose an **Action** or choose **Remove evaluate on exit**.

12. (Optional) Expand **Tags** and then choose **Add tag** to add tags to the resource. Enter a key and optional value, and then choose **Add tag**. You can also turn on **Propagate tags** to propagate tags from the job and job definition to the Amazon ECS task.

13. Choose **Next page**.

14. For **Number of nodes**, enter the total number of nodes to use for your job.

15. For **Main node**, enter the node index to use for the main node. The default main node index is 0.

16. For **Instance type**, choose an instance type.

Note

The instance type that you choose applies to all nodes.

17. For **Parameters**, choose **Add parameters** to add parameter substitution placeholders as **Key** and optional **Value** pairs.

18. In the **Node ranges** section:

   a. Select **Add node range**. This creates a **Node range** section.

   b. For **Target nodes**, specify the range for your node group, using **range_start**:**range_end** notation.

   You can create up to five node ranges for the nodes that you specified for your job. Node ranges use the index value for a node, and the node index begins at 0. Make sure that range end index value of your final node group is one less than the number of nodes that you specified. For example, suppose that you specified 10 nodes, and you want to use a single node group. Then, your end range is 9.

   c. For **Image**, choose the Docker image to use for your job. By default, images in the Docker Hub registry are available. You can also specify other repositories with **repository-url/image:tag**. The name can be up to 225 characters long. It can contain uppercase and lowercase letters, numbers, hyphens (-), underscores (_), colons (:), forward slashes (/),
and number signs (#). This parameter maps to Image in the Create a container section of
the Docker Remote API and the IMAGE parameter of docker run.

Note

Docker image architecture must match the processor architecture of the compute
resources that they're scheduled on. For example, Arm based Docker images can
only run on Arm based compute resources.

- Images in Amazon ECR Public repositories use the full registry/repository[:tag]
or registry/repository[@digest] naming conventions (for example,
- Images in Amazon ECR repositories use the full registry/
  repository[:tag] naming convention. For example,
  aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest
- Images in official repositories on Docker Hub use a single name (for example, ubuntu or
  mongo).
- Images in other repositories on Docker Hub are qualified with an organization name (for
  example, amazon/amazon-ecs-agent).
- Images in other online repositories are qualified further by a domain name (for example,
  quay.io/assemblyline/ubuntu).

d. For Command syntax, choose Bash or JSON.

e. For Command, specify the command to pass to the container. For simple commands,
you can enter the command as you do at a command prompt in the Space delimited
  tab. Then, verify that the JSON result is correct. The JSON result is passed to the Docker
daemon. For more complicated commands (for example, with special characters), you can
switch to the JSON tab and enter the string array equivalent there.

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.
f. For **vCPUs**, specify the number of vCPUs to reserve for the container. This parameter maps to CpuShares in the *Create a container* section of the Docker Remote API and the `--cpu-shares` option to `docker run`. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

g. For **Memory**, specify the hard limit (in MiB) of memory to present to the job's container. If your container attempts to exceed the memory specified here, the container is stopped. This parameter maps to `Memory` in the *Create a container* section of the Docker Remote API and the `--memory` option to `docker run`. You must specify at least 4 MiB of memory for a job.

h. (Optional) For **Number of GPUs**, specify the number of GPUs your job uses. The job runs on a container with the specified number of GPUs that are pinned to that container.

i. (Optional) For **Job role**, you can specify an IAM role that provides the container in your job with permissions to use the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information including configuration prerequisites, see **IAM Roles for Tasks** in the *Amazon Elastic Container Service Developer Guide*.

Note

For jobs that are running on Fargate resources, a job role is required.
j. (Optional) For **Execution role**, specify an IAM role that grants the Amazon ECS container agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for task functionality. For more information, see [Amazon ECS task execution IAM roles](https://docs.aws.amazon.com/elasticcontainerservice/latest/developerguide/task-execution-iam-roles.html) in the *Amazon Elastic Container Service Developer Guide*.

19. (Optional) Expand **Additional configuration**:

a. For **Environment variables**, choose **Add environment variable** to add environment variables as name-value pairs. These variables are passed to the container.

b. For **Job role configuration**, you can specify an IAM role that provides the container in your job with permissions to use the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information including configuration prerequisites, see [IAM Roles for Tasks](https://docs.aws.amazon.com/elasticcontainerservice/latest/developerguide/iam-roles-tasks.html) in the *Amazon Elastic Container Service Developer Guide*.

**Note**

For jobs that are running on Fargate resources, a job role is required.

**Note**

Only roles that have the **Amazon Elastic Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your AWS Batch jobs, see [Creating an IAM Role and Policy for your Tasks](https://docs.aws.amazon.com/batch/latest/userguide/using-iam-roles.html) in the *Amazon Elastic Container Service Developer Guide*.

c. For **Execution role**, specify an IAM role that grants the Amazon ECS container agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for task functionality. For more information, see [Amazon ECS task execution IAM roles](https://docs.aws.amazon.com/elasticcontainerservice/latest/developerguide/task-execution-iam-roles.html) in the *Amazon Elastic Container Service Developer Guide*. 
20. In the **Security Configuration** section:

   a. (Optional) To give your job's container elevated privileges on the host instance (similar to the root user), turn on **Privileged**. This parameter maps to Privileged in the Create a container section of the **Docker Remote API** and the --privileged option to **docker run**.

   b. (Optional) For **User**, enter the user name to use inside the container. This parameter maps to User in the Create a container section of the **Docker Remote API** and the --user option to **docker run**.

   c. (Optional) For **Secrets**, choose **Add secret** to add secrets as a name-value pairs. These secrets are exposed in the container. For more information, see secretOptions in **Job definition parameters for ContainerProperties**.

21. In the **Linux configuration** section:

   a. Turn on **Enable read only filesystem** to remove write access to the volume.

   b. (Optional) Turn on **Enable init process** to run an init process inside the container. This process forwards signals and reaps processes.

   c. For **Shared memory size**, enter the size (in MiB) of the /dev/shm volume.

   d. For **Max swap size**, enter the total amount of swap memory (in MiB) that the container can use.

   e. For **Swappiness** enter a value between 0 and 100 to indicate the swappiness behavior of the container. If you don't specify a value and swapping is enabled, value defaults to 60. For more information, see swappiness in **Job definition parameters for ContainerProperties**.

   f. (Optional) For **Devices**, choose **Add device** to add a device:

      i. For **Container path**, specify the path of in the container instance to expose the device mapped to the host instance. If you keep this blank, the host path is used in the container.

      ii. For **Host path**, specify the path of a device in the host instance.

      iii. For **Permissions**, choose one or more permissions to apply to the device. The available permissions are READ, WRITE, and MKNOD.

22. (Optional) For **Mount points**, choose **Add mount points configuration** to add mount points for data volumes. You must specify the source volume and container path. These mount points are passed to the Docker daemon on a container instance. You can also choose to make the volume **Read only**.
23. (Optional) For **Ulimits configuration**, choose **Add ulimit** to add a ulimits value for the container. Enter **Name**, **Soft limit**, and **Hard limit** values, and then choose **Add ulimit**.

24. (Optional) For **Volumes configuration**, choose **Add volume** to create a list of volumes to pass to the container. Enter **Name** and **Source path** for the volume and then choose **Add volume**. You can also choose to turn on **Enable EFS**.

25. (Optional) For **Tmpfs**, choose **Add tmpfs** to add a tmpfs mount.

26. (Optional) In the **Logging configuration** section:

   a. For **Log driver**, choose the log driver to use. For more information about the available log drivers, see **logDriver** in **Job definition parameters for ContainerProperties**.

      ![](image)
      **Note**
      By default, the awslogs log driver is used.

   b. For **Options**, choose **Add option** to add an option. Enter a name-value pair, and then choose **Add option**.

   c. For **Secrets**, choose **Add secret**. Enter a name-value pair and then choose **Add secret** to add a secret.

      ![](image)
      **Tip**
      For more information, see **secretOptions** in **Job definition parameters for ContainerProperties**.

27. Choose **Next page**.

28. For **Job definition review**, review the configuration steps. If you need to make changes, choose **Edit**. When you’re finished, choose **Create job definition**.

### Creating job definitions using ContainerProperties

The following is an empty job definition template that includes a single container. You can use this template to create your job definition, which can then be saved to a file and used with the AWS CLI **--cli-input-json** option. For more information about these parameters, see **Job definition parameters for ContainerProperties**.

```
{
```
Creating job definitions using ContainerProperties

```
"jobDefinitionName": "",
"type": "container",
"parameters": {
    "KeyName": ""
},
"schedulingPriority": 0,
"containerProperties": {
    "image": "",
    "vcpus": 0,
    "memory": 0,
    "command": [
        ""
    ],
    "jobRoleArn": "",
    "executionRoleArn": "",
    "volumes": [
        {
            "host": {
                "sourcePath": ""
            },
            "name": "",
            "efsVolumeConfiguration": {
                "fileSystemId": "",
                "rootDirectory": "",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": 0,
                "authorizationConfig": {
                    "accessPointId": "",
                    "iam": "DISABLED"
                }
            }
        }
    ],
    "environment": [
        {
            "name": "",
            "value": ""
        }
    ],
    "mountPoints": [
        {
            "containerPath": "",
            "readOnly": true,
            "sourceVolume": ""
        }
    ]
```
"readonlyRootFilesystem": true,
"privileged": true,
"ulimits": [
    {
        "hardLimit": 0,
        "name": ",
        "softLimit": 0
    }
],
"user": ",
"instanceType": ",
"resourceRequirements": [
    {
        "value": ",
        "type": "MEMORY"
    }
],
"linuxParameters": {
    "devices": [
        {
            "hostPath": ",
            "containerPath": ",
            "permissions": [
                "WRITE"
            ]
        }
    ],
    "initProcessEnabled": true,
    "sharedMemorySize": 0,
    "tmpfs": [
        {
            "containerPath": ",
            "size": 0,
            "mountOptions": [
                ""
            ]
        }
    ],
    "maxSwap": 0,
    "swappiness": 0
},
"logConfiguration": {
"logDriver": "syslog",
"options": {
   "KeyName": "",
},
"secretOptions": [
   {
   "name": "",
   "valueFrom": ""
   }
],
"secrets": [
   {
   "name": "",
   "valueFrom": ""
   }
],
"networkConfiguration": {
   "assignPublicIp": "DISABLED"
},
"fargatePlatformConfiguration": {
   "platformVersion": ""
}
},
"nodeProperties": {
   "numNodes": 0,
   "mainNode": 0,
   "nodeRangeProperties": [
   {
   "targetNodes": "",
   "container": {
      "image": "",
      "vcpus": 0,
      "memory": 0,
      "command": [""
      ],
      "jobRoleArn": "",
      "executionRoleArn": "",
      "volumes": [
      {
      "host": {
      "sourcePath": ""
      },
   },

Creating job definitions using ContainerProperties
Creating job definitions using ContainerProperties
"devices": [
    {
        "hostPath": "",
        "containerPath": "",
        "permissions": [
            "WRITE"
        ]
    }
],
"initProcessEnabled": true,
"sharedMemorySize": 0,
"tmpfs": [
    {
        "containerPath": "",
        "size": 0,
        "mountOptions": [
            ""
        ]
    }
],
"maxSwap": 0,
"swappiness": 0
},
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "KeyName": ""
    },
    "secretOptions": [
        {
            "name": "",
            "valueFrom": ""
        }
    ]
},
"secrets": [
    {
        "name": "",
        "valueFrom": ""
    }
],
"networkConfiguration": {
    "assignPublicIp": "DISABLED"
}
"fargatePlatformConfiguration": {
    "platformVersion": "",
}
},
"retryStrategy": {
    "attempts": 0,
    "evaluateOnExit": [
        {
            "onStatusReason": "",
            "onReason": "",
            "onExitCode": "",
            "action": "RETRY"
        }
    ]
},
"propagateTags": true,
"timeout": {
    "attemptDurationSeconds": 0
},
"tags": {
    "KeyName": ""
},
"platformCapabilities": ["EC2"
],
"eksProperties": {
    "podProperties": {
        "serviceAccountName": "",
        "hostNetwork": true,
        "dnsPolicy": "",
        "containers": [
            {
                "name": "",
                "image": "",
                "imagePullPolicy": "",
                "command": [
                    ""
                ],
                "args": [
                    ""
                ],
            }
        ]
    }
}
"env": [ 
    {
        "name": "",
        "value": ""
    } 
],
"resources": {
    "limits": {
        "KeyName": ""
    },
    "requests": {
        "KeyName": ""
    }
},
"volumeMounts": [ 
    {
        "name": "",
        "mountPath": "",
        "readOnly": true
    }
],
"securityContext": {
    "runAsUser": 0,
    "runAsGroup": 0,
    "privileged": true,
    "readOnlyRootFilesystem": true,
    "runAsNonRoot": true
}
],
"volumes": [ 
    {
        "name": "",
        "hostPath": {
            "path": ""
        },
        "emptyDir": {
            "medium": "",
            "sizeLimit": ""
        },
        "secret": {
            "secretName": "",
            "optional": true
        }
    }
]
You can generate a single-container job definition template with the following AWS CLI command:

```
$ aws batch register-job-definition --generate-cli-skeleton
```

## Job definition parameters for ContainerProperties

Job definitions that use `ContainerProperties` are split into several parts:

- the job definition name
- the type of the job definition
- the parameter substitution placeholder defaults
- the container properties for the job
- the Amazon EKS properties for the job definition that are necessary for jobs run on Amazon EKS resources
- the node properties that are necessary for a multi-node parallel job
- the platform capabilities that are necessary for jobs run on Fargate resources
- the default tag propagation details of the job definition
- the default retry strategy for the job definition
- the default scheduling priority for the job definition
- the default tags for the job definition
- the default timeout for the job definition

### Contents

- **Job definition name**
Job definition name

jobDefinitionName

When you register a job definition, you specify a name. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_). The first job definition that's registered with that name is given a revision of 1. Any subsequent job definitions that are registered with that name are given an incremental revision number.

Type: String

Required: Yes

Type

type

When you register a job definition, you specify the type of job. If the job runs on Fargate resources, then multinode isn't supported. For more information about multi-node parallel jobs, see the section called “Creating a multi-node parallel job definition”.

Type: String
Valid values: container | multinode

Required: Yes

Parameters

parameters

When you submit a job, you can specify parameters that replace the placeholders or override the default job definition parameters. Parameters in job submission requests take precedence over the defaults in a job definition. This means that you can use the same job definition for multiple jobs that use the same format. You can also programmatically change values in the command at submission time.

Type: String to string map

Required: No

When you register a job definition, you can use parameter substitution placeholders in the command field of a job's container properties. The syntax is as follows.

```
"command": [
    "ffmpeg",
    "-i",
    "Ref::inputfile",
    "-c",
    "Ref::codec",
    "-o",
    "Ref::outputfile"
]
```

In the above example, there are `Ref::inputfile`, `Ref::codec`, and `Ref::outputfile` parameter substitution placeholders in the command. You can use the parameters object in the job definition to set default values for these placeholders. For example, to set a default for the `Ref::codec` placeholder, you specify the following in the job definition:

```
"parameters" : {"codec" : "mp4"}
```

When this job definition is submitted to run, the `Ref::codec` argument in the command for the container is replaced with the default value, mp4.
Container properties

When you register a job definition, specify a list of container properties that are passed to the Docker daemon on a container instance when the job is placed. The following container properties are allowed in a job definition. For single-node jobs, these container properties are set at the job definition level. For multi-node parallel jobs, container properties are set in the Node properties level, for each node group.

command

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.

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

Type: String array

Required: No

evironment

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 don't recommend that you use plaintext environment variables for sensitive information, such as credential data.

⚠️ Note

Environment variables must not start with AWS_BATCH. This naming convention is reserved for variables that are set by the AWS Batch service.

Type: Array of key-value pairs

Required: No
name

The name of the environment variable.
Type: String
Required: Yes, when environment is used.

value

The value of the environment variable.
Type: String
Required: Yes, when environment is used.

"environment" : [
  { "name" : "envName1", "value" : "envValue1" },
  { "name" : "envName2", "value" : "envValue2" }
]

executionRoleArn

When you register a job definition, you can specify an IAM role. The role provides the Amazon ECS container agent with permissions to call the API actions that are specified in its associated policies on your behalf. Jobs that run on Fargate resources must provide an execution role. For more information, see AWS Batch execution IAM role.

Type: String
Required: No

fargatePlatformConfiguration

The platform configuration for jobs that run on Fargate resources. Jobs that run on EC2 resources must not specify this parameter.

Type: FargatePlatformConfiguration object
Required: No

platformVersion

The AWS Fargate platform version use for the jobs, or LATEST to use a recent, approved version of the AWS Fargate platform.
image

The image used to start a job. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. You can also specify other repositories with `repository-url/image:tag`. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to Image in the Create a container section of the Docker Remote API and the IMAGE parameter of `docker run`.

- Images in Amazon ECR Public repositories use the full `registry/repository[:tag]` or `registry/repository[@digest]` naming conventions (for example, `public.ecr.aws/registry_alias/my-web-app:latest`).
- Images in Amazon ECR repositories use the full `registry/repository:[tag]` naming convention. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`.
- Images in official repositories on Docker Hub use a single name (for example, ubuntu or mongo).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

Type: String

Required: Yes
instanceType

The instance type to use for a multi-node parallel job. All node groups in a multi-node parallel job must use the same instance type. This parameter isn't valid for single-node container jobs or for jobs that run on Fargate resources.

Type: String

Required: No

jobRoleArn

When you register a job definition, you can specify an IAM role. The role provides the job container with permissions to call the API actions that are specified in its associated policies on your behalf. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

Type: String

Required: No

linuxParameters

Linux-specific modifications that are applied to the container, such as details for device mappings.

```
"linuxParameters": {
  "devices": [
    {
      "hostPath": "string",
      "containerPath": "string",
      "permissions": [
        "READ", "WRITE", "MKNOD"
      ]
    }
  ],
  "initProcessEnabled": true|false,
  "sharedMemorySize": 0,
  "tmpfs": [
    {
      "containerPath": "string",
      "size": integer,
      "mountOptions": [
        "string"
      ]
    }
  ]
```

Job definition parameters for ContainerProperties
```
{}
```

Type: `LinuxParameters` object

Required: No

devices

List of devices mapped into 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**

This parameter isn't applicable to jobs that run on Fargate resources.

Type: Array of `Device` objects

Required: No

hostPath

Path where the device available in the host container instance is.

Type: String

Required: Yes

containerPath

Path where the device is exposed in the container is. If this isn't specified, the device is exposed at the same path as the host path.

Type: String

Required: No

permissions

Permissions for the device in the container. If this isn't specified the permissions are set to READ, WRITE, and MKNOD.
Type: Array of strings

Required: No

Valid values: READ | WRITE | MKNOD

initProcessEnabled

If true, 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. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: `sudo docker version | grep "Server API version"

Type: Boolean

Required: No

maxSwap

The total amount of swap memory (in MiB) a job 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. For more information, see `--memory-swap details` in the Docker documentation.

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 runs on. A maxSwap value must be set for the swappiness parameter to be used.

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.

Type: Integer

Required: No

sharedMemorySize

The value for the size (in MiB) of the /dev/shm volume. This parameter maps to the --shm-size option to `docker run`. 

Job definition parameters for ContainerProperties
**swappiness**

You can use this to tune a container's memory swappiness behavior. A swappiness value of 0 causes swapping to not happen unless absolutely necessary. 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`, then this parameter is ignored. If `maxSwap` is set to 0, the container doesn't use swap. This parameter maps to the `--memory-swappiness` option to `docker run`.

Consider the following when you use a per-container swap configuration.

- Swap space must be enabled and allocated on the container instance for the containers to use.

**Note**

The Amazon ECS optimized AMIs don't have swap enabled by default. You must enable swap on the instance to use this feature. For more information, see [Instance Store Swap Volumes](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/swap.html) 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?](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/swap.html).

- The swap space parameters are only supported for job definitions using EC2 resources.
- If the `maxSwap` and `swappiness` parameters are omitted from a job definition, each container has a default `swappiness` value of 60. The total swap usage is limited to two times the memory reservation of the container.

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.
**tmpfs**

The container path, mount options, and size of the tmpfs mount.

Type: Array of **Tmpfs** objects

**Note**

This parameter isn’t applicable to jobs that run on Fargate resources.

**containerPath**

The absolute file path in the container where the tmpfs volume is mounted.

Type: String

Required: Yes

**mountOptions**

The list of tmpfs volume mount options.

Valid values: "defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirmount" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostreamatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"

Type: Array of strings

Required: No

**size**

The size (in MiB) of the tmpfs volume.

Type: Integer
Required: Yes

logConfiguration

The log configuration specification for the job.

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 can 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 either configured on the container instance or on another log server to provide remote logging options. For more information about the options for different supported log drivers, see Configure logging drivers in the Docker documentation.

ℹ️ Note

AWS Batch currently supports a subset of the logging drivers available to the Docker daemon (shown in the LogConfiguration data type).

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: sudo docker version | grep "Server API version"

```
"logConfiguration": {
  "devices": [ {
    "logDriver": "string",
    "options": { "optionName1": "optionValue1", "optionName2": "optionValue2" }
  },
  "secretOptions": [
    { "name": "secretOptionName1", "valueFrom": "secretOptionArn1" },
    { "name": "secretOptionName2", "valueFrom": "secretOptionArn2" }
  ]
}
```
Type: LogConfiguration object

Required: No

logDriver

The log driver to use for the job. By default, AWS Batch enables the awslogs log driver. The valid values that are listed for this parameter are log drivers that the Amazon ECS container agent can communicate with by default.

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, jobs use the same logging driver that the Docker daemon uses. However, the job can use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the job definition. If you want to specify another logging driver for a job, the log system must be configured on the container instance in the compute environment. Or, alternatively, configure it on another log server to provide remote logging options. For more information about the options for different supported log drivers, see Configure logging drivers in the Docker documentation.

Note

AWS Batch currently 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.

The supported log drivers are awslogs, fluentd, gelf, json-file, journald, logentries, syslog, and splunk.

Note

Jobs that run on Fargate resources are restricted to the awslogs and splunk log drivers.
This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: `sudo docker version | grep "Server API version"

**Note**

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. Otherwise, the containers placed on that instance can't use these log configuration options. For more information, see [Amazon ECS Container Agent Configuration](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-agent-logging.html) in the *Amazon Elastic Container Service Developer Guide*.

awslogs

Specifies the Amazon CloudWatch Logs logging driver. For more information, see [Using the awslogs log driver](https://docs.docker.com/config/logging/aws/) and [Amazon CloudWatch Logs logging driver](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-agent-logging.html) in the Docker documentation.

fluentd

Specifies the Fluentd logging driver. For more information including usage and options, see [Fluentd logging driver](https://docs.docker.com/config/logging/containers/) in the Docker documentation.

gelf

Specifies the Graylog Extended Format (GELF) logging driver. For more information including usage and options, see [Graylog Extended Format logging driver](https://docs.docker.com/config/logging/containers/) in the Docker documentation.

journald

Specifies the journald logging driver. For more information including usage and options, see [Journald logging driver](https://docs.docker.com/config/logging/containers/) in the Docker documentation.

json-file

Specifies the JSON file logging driver. For more information including usage and options, see [JSON File logging driver](https://docs.docker.com/config/logging/containers/) in the Docker documentation.
splunk

    Specifies the Splunk logging driver. For more information including usage and options, see [Splunk logging driver](https://docs.docker.com/config/connect/logging/driver/splunk/) in the Docker documentation.

syslog

    Specifies the syslog logging driver. For more information including usage and options, see [Syslog logging driver](https://docs.docker.com/config/connect/logging/driver/syslog/) in the Docker documentation.

Type: String

Required: Yes

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

**Note**

If you have a custom driver that's not listed earlier that you would like to work with the Amazon ECS container agent, you can fork the Amazon ECS container agent project that's [available on GitHub](https://github.com/aws/amazon-ecs-container-agent) and customize it to work with that driver. We encourage you to submit pull requests for changes that you want to have included. However, Amazon Web Services doesn't currently support requests that run modified copies of this software.

**options**

Log configuration options to send to a log driver for the job.

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

Type: String to string map

Required: No

**secretOptions**

An object that represents the secret to pass to the log configuration. For more information, see [Specifying sensitive data](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/parameters.html).

Type: object array
Required: No

name

The name of the log driver option to set in the job.

Type: String

Required: Yes

valueFrom

The Amazon Resource Name (ARN) of the secret to expose to the log configuration of the container. The supported values are either the full ARN of the Secrets Manager secret or the full ARN of the parameter in the SSM Parameter Store.

**Note**

If the SSM Parameter Store parameter 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 parameter. If the parameter exists in a different Region, then the full ARN must be specified.

Type: String

Required: Yes

memory

_This parameter is deprecated, use `resourceRequirements` instead._

The number of MiB of memory reserved for the job.

As an example for how to use `resourceRequirements`, if your job definition contains syntax that's similar to the following.

```
"containerProperties": {
    "memory": 512
}
```

The equivalent syntax using `resourceRequirements` is as follows.
"containerProperties": { 
  "resourceRequirements": [ 
    { 
      "type": "MEMORY",
      "value": "512"
    } 
  ]
}

Type: Integer

Required: Yes

mountPoints

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.

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

Type: Object array

Required: No

sourceVolume

The name of the volume to mount.

Type: String

Required: Yes, when mountPoints is used.

containerPath

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

Type: String
Required: Yes, when mountPoints is used.

**readOnly**

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.

Type: Boolean

Required: No

Default: False

**networkConfiguration**

The network configuration for jobs that run on Fargate resources. Jobs that run on EC2 resources must not specify this parameter.

```
"networkConfiguration": {
   "assignPublicIp": "string"
}
```

Type: Object array

Required: No

**assignPublicIp**

Indicates whether the job has a public IP address. This is required if the job needs outbound network access.

Type: String

Valid values: ENABLED | DISABLED

Required: No

Default: DISABLED

**privileged**

When this parameter is true, the container is given elevated permissions on the host container instance (similar to the root user). This parameter maps to Privileged in the Create a container section of the Docker Remote API and the --privileged option to docker run. This
parameter isn't applicable to jobs that run on Fargate resources. Don't provide it or specify it as false.

"privileged": true/false

Type: Boolean
Required: No

readonlyRootFilesystem

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

"readonlyRootFilesystem": true/false

Type: Boolean
Required: No

resourceRequirements

The type and amount of a resource to assign to a container. The supported resources include GPU, MEMORY, and VCPU.

"resourceRequirements" : [
    {
        "type": "GPU",
        "value": "number"
    }
]

Type: Object array
Required: No

The type of resource to assign to a container. The supported resources include GPU, MEMORY, and VCPU.

Type: String
Required: Yes, when `resourceRequirements` is used.

**value**

The quantity of the specified resource to reserve for the container. The values vary based on the type specified.

**type="GPU"**

The number of physical GPUs to reserve for the container. The number of GPUs reserved for all containers in a job cannot exceed the number of available GPUs on the compute resource that the job is launched on.

**type="MEMORY"**

The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to `Memory` in the Create a container section of the Docker Remote API and the `--memory` option to `docker run`. You must specify at least 4 MiB of memory for a job. This is required but can be specified in several places for multi-node parallel (MNP) jobs. It must be specified for each node at least once. This parameter maps to `Memory` in the Create a container section of the Docker Remote API and the `--memory` option to `docker run`.

![Note](https://example.com/note.png)

If you're trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, see Compute Resource Memory Management.

For jobs that run on Fargate resources, then `value` must match one of the supported values. Moreover, the VCPU values must be one of the values that's supported for that memory value.

<table>
<thead>
<tr>
<th>VCPU</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25 vCPU</td>
<td>512, 1024, and 2048 MiB</td>
</tr>
<tr>
<td>0.5 vCPU</td>
<td>1024-4096 MiB in 1024 MiB increments</td>
</tr>
<tr>
<td>1 vCPU</td>
<td>2048-8192 MiB in 1024 MiB increments</td>
</tr>
</tbody>
</table>
The number of vCPUs reserved for the job. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to docker run. Each vCPU is equivalent to 1,024 CPU shares. For jobs that run on EC2 resources, you must specify at least one vCPU. This is required but can be specified in several places. It must be specified for each node at least once.

For jobs that run on Fargate resources, value must match one of the supported values and the MEMORY values must be one of the values that's supported for that VCPU value. The supported values are 0.25, 0.5, 1, 2, 4, 8, and 16.

The default for the Fargate On-Demand vCPU resource count quota is 6 vCPUs. For more information about Fargate quotas, see AWS Fargate quotas in the Amazon Web Services General Reference.

Type: String

Required: Yes, when resourceRequirements is used.

secrets

The secrets for the job that are exposed as environment variables. For more information, see Specifying sensitive data.

"secrets": [
  {
    "name": "secretName1",
    "valueFrom": "secretArn1"
  },
  
]
"name": "secretName2",
"valueFrom": "secretArn2"
}
...
]

Type: Object array

Required: No

name

The name of the environment variable that contains the secret.

Type: String

Required: Yes, when secrets is used.

valueFrom

The secret to expose to the container. The supported values are either the full Amazon Resource Name (ARN) of the Secrets Manager secret or the full ARN of the parameter in the SSM Parameter Store.

⚠️ Note

If the SSM Parameter Store parameter exists in the same AWS Region as the job you're launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

Type: String

Required: Yes, when secrets is used.

ulimits

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

"ulimits": [
{
    "name": string,
...}
"softLimit": integer,
"hardLimit": integer
}
...
]

Type: Object array

Required: No

name

The type of the ulimit.

Type: String

Required: Yes, when ulimits is used.

hardLimit

The hard limit for the ulimit type.

Type: Integer

Required: Yes, when ulimits is used.

softLimit

The soft limit for the ulimit type.

Type: Integer

Required: Yes, when ulimits is used.

user

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

"user": "string"

Type: String

Required: No
vcpus

*This parameter is deprecated, use `resourceRequirements` instead.*

The number of vCPUs reserved for the container.

As an example for how to use `resourceRequirements`, if your job definition contains lines similar to this:

```
"containerProperties": {
    "vcpus": 2
}
```

The equivalent lines using `resourceRequirements` is as follows.

```
"containerProperties": {
    "resourceRequirements": [
        {
            "type": "VCPU",
            "value": "2"
        }
    ]
}
```

Type: Integer

Required: Yes

volumes

When you register a job definition, you can specify a list of volumes that are passed to the Docker daemon on a container instance. The following parameters are allowed in the container properties:

```
"volumes": [
    {
        "name": "string",
        "host": {
            "sourcePath": "string"
        },
        "efsVolumeConfiguration": {
            "authorizationConfig": {
                "accessPointId": "string"
            }
        }
    }
]
```
name

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`.

Type: String

Required: No

host

The contents of the `host` parameter determine whether your data volume persists on the host container instance and where it's stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume. However, the data isn't guaranteed to persist after the container associated with it stops running.

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.

Type: Object

Required: No

sourcePath

The path on the host container instance that's presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you.

If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host container instance until you delete it.
manually. If the sourcePath value 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.

Type: String
Required: No

efsVolumeConfiguration

This parameter is specified when you're using an Amazon Elastic File System file system for task storage. For more information, see Amazon EFS volumes.

Type: Object
Required: No

authorizationConfig

The authorization configuration details for the Amazon EFS file system.

Type: String
Required: No

accessPointId

The Amazon EFS access point ID to use. If an access point is specified, the root directory value that's specified in the EFSVolumeConfiguration must either be omitted or set to /.. This enforces the path that's 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.

Type: String
Required: No

iam

Determines whether to use the AWS Batch job IAM role defined in a job 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 Using Amazon EFS access points.
Type: String

Valid values: ENABLED | DISABLED

Required: No

fileSystemId

The Amazon EFS file system ID to use.

Type: String

Required: No

rootDirectory

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. If you specify /, it has the same effect as omitting this parameter. The maximum length is 4,096 characters.

⚠️ Important

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

Type: String

Required: No

transitEncryption

Determines whether 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.

Type: String

Valid values: ENABLED | DISABLED
transitEncryptionPort

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. The value must be between 0 and 65,535. For more information, see EFS Mount Helper in the Amazon Elastic File System User Guide.

Type: Integer

Required: No

Amazon EKS properties

An object with various properties that are specific to Amazon EKS based jobs. This must not be specified for Amazon ECS based job definitions.

podProperties

The properties for the Kubernetes pod resources of a job.

Type: EksPodProperties object

Required: No

containers

The properties of the container that's used on the Amazon EKS pod.

Type: EksContainer object

Required: No

args

An array of arguments to the entrypoint. If this isn't specified, the CMD of the container image is used. This corresponds to the args member in the Entrypoint portion of the Pod in Kubernetes. Environment variable references are expanded using the container's environment.

If the referenced environment variable doesn't exist, the reference in the command isn't changed. For example, if the reference is to "$(NAME1)" and the NAME1 environment
variable doesn't exist, the command string will remain "$(NAME1)." $$ is replaced with $, and the resulting string isn't expanded. For example, $$\$(VAR\_NAME)$$ is passed as $$\$(VAR\_NAME)$$ whether or not the VAR\_NAME environment variable exists. For more information, see CMD in the Dockerfile reference and Define a command and arguments for a pod in the Kubernetes documentation.

Type: Array of strings

Required: No

command

The entrypoint for the container. This isn't run within a shell. If this isn't specified, the ENTRYPPOINT of the container image is used. Environment variable references are expanded using the container's environment.

If the referenced environment variable doesn't exist, the reference in the command isn't changed. For example, if the reference is to "$(NAME1)" and the NAME1 environment variable doesn't exist, the command string will remain "$(NAME1)." $$ is replaced with $ and the resulting string isn't expanded. For example, $$\$(VAR\_NAME)$$ will be passed as $$\$(VAR\_NAME)$$ whether or not the VAR\_NAME environment variable exists. The entrypoint can't be updated. For more information, see ENTRYPPOINT in the Dockerfile reference and Define a command and arguments for a container and Entrypoint in the Kubernetes documentation.

Type: Array of strings

Required: No

env

The environment variables to pass to a container.

Note

Environment variables cannot start with "AWS\_BATCH". This naming convention is reserved for variables that AWS Batch sets.

Type: Array of EksContainerEnvironmentVariable objects
Required: No
name

The name of the environment variable.
Type: String
Required: Yes
value

The value of the environment variable.
Type: String
Required: No
image

The Docker image used to start the container.
Type: String
Required: Yes
imagePullPolicy

The image pull policy for the container. Supported values are Always, IfNotPresent, and Never. This parameter defaults to IfNotPresent. However, if the :latest tag is specified, it defaults to Always. For more information, see Updating images in the Kubernetes documentation.
Type: String
Required: No
name

The name of the container. If the name isn't specified, the default name "Default" is used. Each container in a pod must have a unique name.
Type: String
Required: No
resources

The type and amount of resources to assign to a container. The supported resources include memory, cpu, and nvidia.com/gpu. For more information, see Resource management for pods and containers in the Kubernetes documentation.

Type: EksContainerResourceRequirements object

Required: No

limits

The type and quantity of the resources to reserve for the container. The values vary based on the name that's specified. Resources can be requested using either the limits or the requests objects.

memory

The memory hard limit (in MiB) for the container, using whole integers, with a "Mi" suffix. If your container attempts to exceed the memory specified, the container is terminated. You must specify at least 4 MiB of memory for a job. memory can be specified in limits, requests, or both. If memory is specified in both places, then the value that's specified in limits must be equal to the value that's specified in requests.

Note

To maximize your resource utilization, provide your jobs with as much memory as possible for the specific instance type that you are using. To learn how, see Compute Resource Memory Management.

cpu

The number of CPUs that's reserved for the container. Values must be an even multiple of 0.25. cpu can be specified in limits, requests, or both. If cpu is specified in both places, then the value that's specified in limits must be at least as large as the value that's specified in requests.

nvidia.com/gpu

The number of GPUs that's reserved for the container. Values must be a whole integer. memory can be specified in limits, requests, or both. If memory is
specified in both places, then the value that's specified in limits must be equal to the value that's specified in requests.

Type: String to string map

Value Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

requests

The type and quantity of the resources to request for the container. The values vary based on the name that's specified. Resources can be requested by using either the limits or the requests objects.

memory

The memory hard limit (in MiB) for the container, using whole integers, with a "Mi" suffix. If your container attempts to exceed the memory specified, the container is terminated. You must specify at least 4 MiB of memory for a job. memory can be specified in limits, requests, or both. If memory is specified in both, then the value that's specified in limits must be equal to the value that's specified in requests.

Note

If you're trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, see Compute Resource Memory Management.

cpu

The number of CPUs that are reserved for the container. Values must be an even multiple of 0.25. cpu can be specified in limits, requests, or both. If cpu is specified in both, then the value that's specified in limits must be at least as large as the value that's specified in requests.

nvidia.com/gpu

The number of GPUs that are reserved for the container. Values must be a whole integer. nvidia.com/gpu can be specified in limits, requests, or both. If
nvidia.com/gpu is specified in both, then the value that's specified in limits must be equal to the value that's specified in requests.

Type: String to string map

Value Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

securityContext

The security context for a job. For more information, see Configure a security context for a pod or container in the Kubernetes documentation.

Type: EksContainerSecurityContext object

Required: No

privileged

When this parameter is true, the container is given elevated permissions on the host container instance. The level of permissions is similar to the root user permissions. The default value is false. This parameter maps to privileged policy in the Privileged pod security policies in the Kubernetes documentation.

Type: Boolean

Required: No

readOnlyRootFilesystem

When this parameter is true, the container is given read-only access to its root file system. The default value is false. This parameter maps to ReadOnlyRootFilesystem policy in the Volumes and file systems pod security policies in the Kubernetes documentation.

Type: Boolean

Required: No

runAsGroup

When this parameter is specified, the container is run as the specified group ID (gid). If this parameter isn't specified, the default is the group that's specified in the image.
metadata. This parameter maps to RunAsGroup and MustRunAs policy in the Users and groups pod security policies in the Kubernetes documentation.

Type: Long

Required: No

runAsNonRoot

When this parameter is specified, the container is run as a user with a uid other than 0. If this parameter isn't specified, so such rule is enforced. This parameter maps to RunAsUser and MustRunAsNonRoot policy in the Users and groups pod security policies in the Kubernetes documentation.

Type: Long

Required: No

runAsUser

When this parameter is specified, the container is run as the specified user ID (uid). If this parameter isn't specified, the default is the user that's specified in the image metadata. This parameter maps to RunAsUser and MustRanAs policy in the Users and groups pod security policies in the Kubernetes documentation.

Type: Long

Required: No

volumeMounts

The volume mounts for a container for an Amazon EKS job. For more information about volumes and volume mounts in Kubernetes, see Volumes in the Kubernetes documentation.

Type: Array of EksContainerVolumeMount objects

Required: No

mountPath

The path on the container where the volume is mounted.

Type: String
name

The name the volume mount. This must match the name of one of the volumes in the pod.

Type: String

readOnly

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

Type: Boolean

dnsPolicy

The DNS policy for the pod. The default value is ClusterFirst. If the hostNetwork parameter is not specified, the default is ClusterFirstWithHostNet. ClusterFirst indicates that any DNS query that does not match the configured cluster domain suffix is forwarded to the upstream nameserver inherited from the node. If no value was specified for dnsPolicy in the RegisterJobDefinition API operation, then no value is returned for dnsPolicy by either of DescribeJobDefinitions or DescribeJobs API operations. The pod spec setting will contain either ClusterFirst or ClusterFirstWithHostNet, depending on the value of the hostNetwork parameter. For more information, see Pod's DNS policy in the Kubernetes documentation.

Valid values: Default | ClusterFirst | ClusterFirstWithHostNet

Type: String

hostNetwork

Indicates if the pod uses the hosts' network IP address. The default value is true. Setting this to false enables the Kubernetes pod networking model. Most AWS Batch workloads are egress-only and don't require the overhead of IP allocation for each pod for incoming...
connections. For more information, see Host namespaces and Pod networking in the Kubernetes documentation.

Type: Boolean

Required: No

serviceAccountName

The name of the service account that's used to run the pod. For more information, see Kubernetes service accounts and Configure a Kubernetes service account to assume an IAM role in the Amazon EKS User Guide and Configure service accounts for pods in the Kubernetes documentation.

Type: String

Required: No

volumes

Specifies the volumes for a job definition that uses Amazon EKS resources.

Type: Array of EksVolume objects

Required: No

easytDir

Specifies the configuration of a Kubernetes emptyDir volume. An emptyDir volume is first created when a pod is assigned to a node. It exists as long as that pod runs on that node. The emptyDir volume is initially empty. All containers in the pod can read and write the files in the emptyDir volume. However, the emptyDir volume can be mounted at the same or different paths in each container. When a pod is removed from a node for any reason, the data in the emptyDir is deleted permanently. For more information, see emptyDir in the Kubernetes documentation.

Type: EksEmptyDir object

Required: No

medium

The medium to store the volume. The default value is an empty string, which uses the storage of the node.
(Default) Use the disk storage of the node.

"Memory"

Use the tmpfs volume that's backed by the RAM of the node. Contents of the volume are lost when the node reboots, and any storage on the volume counts against the container's memory limit.

Type: String

Required: No

sizeLimit

The maximum size of the volume. By default, there's no maximum size defined.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

hostPath

Specifies the configuration of a Kubernetes hostPath volume. A hostPath volume mounts an existing file or directory from the host node's filesystem into your pod. For more information, see hostPath in the Kubernetes documentation.

Type: EksHostPath object

Required: No

path

The path of the file or directory on the host to mount into containers on the pod.

Type: String

Required: No

name

The name of the volume. The name must be allowed as a DNS subdomain name. For more information, see DNS subdomain names in the Kubernetes documentation.
Type: String

Required: Yes

secret

Specifies the configuration of a Kubernetes secret volume. For more information, see secret in the Kubernetes documentation.

Type: EksSecret object

Required: No

optional

Specifies whether the secret or the secret's keys must be defined.

Type: Boolean

Required: No

secretName

The name of the secret. The name must be allowed as a DNS subdomain name. For more information, see DNS subdomain names in the Kubernetes documentation.

Type: String

Required: Yes

Platform capabilities

platformCapabilities

The platform capabilities that's required by the job definition. If no value is specified, it defaults to EC2. For jobs that run on Fargate resources, FARGATE is specified.

Note

If the job runs on Amazon EKS resources, then you must not specify platformCapabilities.
Type: String

Valid values: EC2 | FARGATE

Required: No

**Propagate tags**

propagateTags

Specifies whether to propagate the tags from the job or job definition to the corresponding Amazon ECS task. If no value is specified, the tags aren't propagated. Tags can only be propagated to the tasks when the task is created. For tags with the same name, job tags are given priority over job definitions tags. If the total number of combined tags from the job and job definition is over 50, the job's moved to the FAILED state.

**Note**

If the job runs on Amazon EKS resources, then you must not specify propagateTags.

Type: Boolean

Required: No

**Node properties**

dnodeProperties

When you register a multi-node parallel job definition, you must specify a list of node properties. These node properties define the number of nodes to use in your job, the main node index, and the different node ranges to use. If the job runs on Fargate resources, then you can't specify nodeProperties. Instead, use containerProperties. The following node properties are allowed in a job definition. For more information, see [Multi-node parallel jobs](#).

**Note**

If the job runs on Amazon EKS resources, then you must not specify nodeProperties.
Type: **NodeProperties** object

Required: No

**mainNode**

Specifies the node index for the main node of a multi-node parallel job. This node index value must be smaller than the number of nodes.

Type: Integer

Required: Yes

**numNodes**

The number of nodes that are associated with a multi-node parallel job.

Type: Integer

Required: Yes

**nodeRangeProperties**

A list of node ranges and their properties that are associated with a multi-node parallel job.

**Note**

A node group is an identical group of job nodes that all share the same container properties. You can use AWS Batch to specify up to five distinct node groups for each job.

Type: Array of **NodeRangeProperty** objects

Required: Yes

**targetNodes**

The range of nodes, using node index values. A range of 0:3 indicates nodes with index values of 0 through 3. If the starting range value is omitted (:n), then 0 is used to start the range. If the ending range value is omitted (n:), then the highest possible node index is used to end the range. Your accumulative node ranges must account for all nodes.
(0 : n). You can nest node ranges, for example 0:10 and 4:5. For this case, the 4:5 range properties override the 0:10 properties.

Type: String

Required: No

circle

The container details for the node range. For more information, see Container properties.

Type: ContainerProperties object

Required: No

**Retry strategy**

retryStrategy

When you register a job definition, you can optionally specify a retry strategy to use for failed jobs that are submitted with this job definition. Any retry strategy that's specified during a SubmitJob operation overrides the retry strategy defined here. By default, each job is attempted one time. If you specify more than one attempt, the job is retried if it fails. Examples of a fail attempt include the job returns a non-zero exit code or the container instance is terminated. For more information, see Automated job retries.

Type: RetryStrategy object

Required: No

attempts

The number of times to move a job to the RUNNABLE status. You can specify between 1 and 10 attempts. If attempts is greater than one, the job is retried that many times if it fails, until it has moved to RUNNABLE.

"attempts": integer

Type: Integer

Required: No
evaluateOnExit

Array of up to 5 objects that specify conditions under which the job is retried or failed. If this parameter is specified, then the attempts parameter must also be specified. If evaluateOnExit is specified but none of the entries match, then the job is retried.

```
"evaluateOnExit": [
  {
    "action": "string",
    "onExitCode": "string",
    "onReason": "string",
    "onStatusReason": "string"
  }
]
```

Type: Array of EvaluateOnExit objects

Required: No

action

Specifies the action to take if all of the specified conditions (onStatusReason, onReason, and onExitCode) are met. The values aren't case sensitive.

Type: String

Required: Yes

Valid values: RETRY | EXIT

onExitCode

Contains a glob pattern to match against the decimal representation of the ExitCode that's returned for a job. The pattern can be up to 512 characters in length. It can contain only numbers. It cannot contain letters or special characters. It can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

Type: String

Required: No

onReason

Contains a glob pattern to match against the Reason that's returned for a job. The pattern can be up to 512 characters in length. It can contain letters, numbers, periods (.),
colons (:), and white space (spaces, tabs). It can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

**Type:** String

**Required:** No

**onStatusReason**

Contains a glob pattern to match against the StatusReason that's returned for a job. The pattern can be up to 512 characters in length. It can contain letters, numbers, periods (.), colons (:), and white space (spaces, tabs). It can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

**Type:** String

**Required:** No

**Scheduling priority**

**schedulingPriority**

The scheduling priority for jobs that are submitted with this job definition. This only affects jobs in job queues with a fair share policy. Jobs with a higher scheduling priority are scheduled before jobs with a lower scheduling priority.

The minimum supported value is 0 and the maximum supported value is 9999.

**Type:** Integer

**Required:** No

**Tags**

**tags**

Key-value pair tags to associate with the job definition. For more information, see [Tagging your AWS Batch resources](#).

**Type:** String to string map
Timeout

timeout

You can configure a timeout duration for your jobs so that if a job runs longer than that, AWS Batch terminates the job. For more information, see Job timeouts. If a job is terminated because of a timeout, it isn't retried. Any timeout configuration that's specified during a SubmitJob operation overrides the timeout configuration defined here. For more information, see Job timeouts.

Type: JobTimeout object

Required: No

attemptDurationSeconds

The time duration in seconds (measured from the job attempt's startedAt timestamp) after AWS Batch terminates unfinished jobs. The minimum value for the timeout is 60 seconds.

For array jobs, the timeout applies to the child jobs, not to the parent array job.

For multi-node parallel (MNP) jobs, the timeout applies to the whole job, not to the individual nodes.

Type: Integer

Required: No

Creating job definitions using EcsProperties

With AWS Batch job definitions using EcsProperties, you can model hardware, sensors, 3D environments and other simulations in separate containers. You can use this feature to logically organize your workload components, and separate them from the main application. This feature can be used with AWS Batch on Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and AWS Fargate.
ContainerProperties versus EcsProperties job definitions

You can choose to use ContainerProperties or EcsProperties job definitions as your use case dictates. At a high-level, running AWS Batch jobs with EcsProperties is similar to running jobs with a ContainerProperties.

The legacy job definition structure, using ContainerProperties, remains supported. If you currently have workflows using this structure, you can continue to run them.

The main difference is that there is a new object added to the job definition to accommodate EcsProperties-based definitions.

For example, a job definition that uses ContainerProperties on Amazon ECS and Fargate has the following structure:

```json
{
   "containerProperties": {
      ...,
      "image": "my_ecr_image1",
      ...
   },
   ...
}
```

A job definition that uses EcsProperties on Amazon ECS and Fargate has the following structure:

```json
{
   "ecsProperties": {
      "taskProperties": [{
         "containers": [
            {
               ...,
               "image": "my_ecr_image1",
               ...
            },
            {
               ...
               "image": "my_ecr_image2",
               ...
            }
         ]
      }
   }
}```
General changes to the AWS Batch APIs

The following further outlines some of the key differences when using the EcsProperties and the EcsProperties API data types:

- Many of the parameters that are used within ContainerProperties appear within TaskContainerProperties. Some examples include, command, image, privileged, secrets, and users. They can all be found within TaskContainerProperties.

- Some of the TaskContainerProperties parameters don’t have functional equivalents in the legacy structure. Some examples include, dependsOn, essential, name, ipcMode, and pidMode. For more information, see EcsTaskDetails and TaskContainerProperties.

As well, some ContainerProperties parameters don’t have equivalents, or application, in the EcsProperties structure. In taskProperties, container has been replaced with containers so that the new object can accept up to ten elements. For more information see RegisterJobDefinition:containerProperties and EcsTaskProperties:containers.

- taskRoleArn is functionally equivalent to jobRoleArn. For more information see EcsTaskProperties:taskRoleArn and ContainerProperties:jobRoleArn.

- You can include from one (1) to ten (10) containers in the EcsProperties structure. For more information see EcsTaskProperties:containers.

- The taskProperties and instanceTypes objects are arrays, but currently accept only one element. For example, EcsProperties:taskProperties and NodeRangeProperty:instanceTypes.

Multi-container job definitions for Amazon ECS

To accommodate the multi-container structure for Amazon ECS, some of the API data types are different. For example,

- ecsProperties is the same level as containerProperties in the single-container definition. For more information, see EcsProperties in the AWS Batch API Reference Guide.

- taskProperties contains the properties defined for the Amazon ECS task. For more information, see EcsProperties in the AWS Batch API Reference Guide.

- containers includes similar information to containerProperties in the single-container definition. The main difference is that containers allows you to define up to ten containers. For more information, see ECSTaskProperties:containers in the AWS Batch API Reference Guide.
• **essential** parameter indicates how the container affects the job. All essential containers must complete successfully (exit as 0) in order for the job to progress. If a container that is marked as essential fails (exits as non-0), then the job fails.

The default value is `true` and at least one container must be marked as `essential`. For more information, see **essential** in the *AWS Batch API Reference Guide*.

• With the **dependsOn** parameter, you can define a list of container dependencies. For more information, see **dependsOn** in the *AWS Batch API Reference Guide*.

---

**Note**

The complexity of the `dependsOn` list and the associated container runtime can affect the start time for your job. If the dependencies take a long time to run, the job will remain in a STARTING state until they complete.

---

For more information about the `ecsProperties` and structure, see [RegisterJobDefinition](https://docs.aws.amazon.com/batch/latest/userguide/register-jobs.html) request syntax for `ecsProperties`.

### Multi-container job definitions for Amazon EKS

To accommodate the multi-container structure for Amazon EKS, some of the API data types are different. For example,

• **name** is a unique identifier for the container. This object isn't required for a single container, but is required when defining multiple containers in a pod. When `name` isn't defined for single containers, the default name, `default`, is applied.

• **initContainers** are defined within the `eksPodProperties` data type. They run before application containers, always runs to completion, and must complete successfully before the next container starts.

These containers are registered with the Amazon EKS Connector agent and persists the registration information in the Amazon Elastic Kubernetes Service backend data store. The `initContainers` object can accept up to ten (10) elements. For more information, see [Init Containers](https://kubernetes.io/docs/concepts/workloads/pods/init-containers/) in the *Kubernetes documentation*. 
---

**Note**

The `initContainers` object can affect the start time for your job. If the `initContainers` take a long time to run, the job will remain in a STARTING state until they complete.

- `shareProcessNamespace` indicates if the containers in the pod can share the same process namespace. The default value is false. Setting this to true enables containers to see and signal processes in other containers that are located in the same pod.

- Every container has importance. All containers must complete successfully (exit as 0) for the job to succeed. If one container fails (exits as other than 0), then the job fails.

For more information about the `eksProperties` and structure, see [RegisterJobDefinition](https://docs.aws.amazon.com/batch/latest/userguide/eks-properties.html) request syntax for `eksProperties`.

**AWS Batch job scenarios using EcsProperties**

To illustrate how AWS Batch job definitions that use EcsProperties can be structured based on your needs, this topic presents the following `RegisterJobDefinition` payloads. You can copy these examples into a file, customize them to your needs, and then use the AWS Command Line Interface (AWS CLI) to call `RegisterJobDefinition`.

**AWS Batch job for Amazon Elastic Container Service on Amazon Elastic Compute Cloud**

```json
{
    "jobDefinitionName": "multicontainer-ecs-ec2",
    "type": "container",
    "ecsProperties": {
        "taskProperties": {
            "containers": [
                { "name": "c1", "essential": false, "command": ["echo", "hello world"]
            ]
        }
    }
}
```
AWS Batch job for Amazon ECS on AWS Fargate

{  
  "jobDefinitionName": "multicontainer-ecs-fargate",
  "type": "container",
  
}
"platformCapabilities": [ "FARGATE"
],
"ecsProperties": {
  "taskProperties": [
    {
      "containers": [
        {
          "name": "c1",
          "command": [
            "echo",
            "hello world"
          ],
          "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
          "resourceRequirements": [
            {
              "type": "VCPU",
              "value": "2"
            },
            {
              "type": "MEMORY",
              "value": "4096"
            }
          ]
        },
        {
          "name": "c2",
          "essential": true,
          "command": [
            "echo",
            "hello world"
          ],
          "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
          "resourceRequirements": [
            {
              "type": "VCPU",
              "value": "6"
            },
            {
              "type": "MEMORY",
              "value": "12288"
            }
          ]
        }
      ]
    }
  ]
}
AWS Batch job for Amazon Elastic Kubernetes Service

{
  "jobDefinitionName": "multicontainer-eks",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "shareProcessNamespace": true,
      "initContainers": [
        {
          "name": "init-container",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": ["echo"],
          "args": ["hello world"],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        },
        {
          "name": "init-container-2",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": ["echo", "my second init container"],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        }
      ]
    }
  }
}
Multi-node parallel (MNP) AWS Batch job with multiple containers per node

```json
{
    "jobDefinitionName": "multicontainer-mnp",
    "type": "multinode",
    "nodeProperties": {
        "containers": [
            {
                "name": "c1",
                "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                "command": [
                    "echo world"
                ],
                "resources": {
                    "requests": {
                        "cpu": "1",
                        "memory": "512Mi"
                    }
                }
            },
            {
                "name": "sleep-container",
                "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                "command": [
                    "sleep",
                    "20"
                ],
                "resources": {
                    "requests": {
                        "cpu": "1",
                        "memory": "512Mi"
                    }
                }
            }
        ]
    }
}
```
"numNodes": 6,
"mainNode": 0,
"nodeRangeProperties": [
{
  "targetNodes": "0:5",
  "ecsProperties": {
    "taskProperties": [
      {
        "containers": [
          {
            "name": "range05-c1",
            "command": [
              "echo",
              "hello world"
            ],
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "resourceRequirements": [
              {
                "type": "VCPU",
                "value": "2"
              },
              {
                "type": "MEMORY",
                "value": "4096"
              }
            ]
          },
          {
            "name": "range05-c2",
            "command": [
              "echo",
              "hello world"
            ],
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "resourceRequirements": [
              {
                "type": "VCPU",
                "value": "2"
              },
              {
                "type": "MEMORY",
                "value": "4096"
              }
            ]
          }
        ]
      }
    ]
  }
}
Using the `awslogs` log driver

By default, AWS Batch enables the `awslogs` log driver to send log information to CloudWatch Logs. You can use this feature to view different logs from your containers in one convenient location and prevent your container logs from taking up disk space on your container instances. This topic helps you configure the `awslogs` log driver in your job definitions.

**Note**

In the AWS Batch console, you can configure the `awslogs` log driver in the Logging **configuration** section when you create a job definition.

**Note**

The type of information that's logged by the containers in your job depends mostly on their ENTRYPOINT command. By default, the logs that are captured show the command output that you normally see in an interactive terminal if you ran the container locally, which are the STDOUT and STDERR I/O streams. The `awslogs` log driver simply passes these logs from Docker to CloudWatch 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 container instances to CloudWatch Logs, see Using CloudWatch Logs with AWS Batch. For more information about CloudWatch Logs, see Monitoring Log Files and CloudWatch Logs quotas in the Amazon CloudWatch Logs User Guide.
### Available awslogs log driver options

The awslogs log driver supports the following options in AWS Batch job definitions. For more information, see [CloudWatch Logs logging driver](#) in the Docker documentation.

#### awslogs-region

**Required:** No

Specify the Region where the awslogs log driver should send your Docker logs. By default, the Region that's used is the same one as the one for the job. You can choose to send all of your logs from jobs in different Regions to a single Region in CloudWatch Logs. Doing this allows them to be visible all from one location. Alternatively, you can separate them by Region for more granular approach. However, when you choose this option, make sure that the specified log groups exists in the Region that you specified.

#### awslogs-group

**Required:** Optional

With the `awslogs-group` option, you can specify the log group that the awslogs log driver sends its log streams to. If this isn't specified, `aws/batch/job` is used.

#### awslogs-stream-prefix

**Required:** Optional

With the `awslogs-stream-prefix` option, you can associate a log stream with the specified prefix, and the Amazon ECS task ID of the AWS Batch job that the container belongs to. If you specify a prefix with this option, then the log stream takes the following format:

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

#### awslogs-datetime-format

**Required:** No

This option defines a multiline start pattern in Python `datetime` format. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. Thus the matched line is the delimiter between log messages.
One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

For more information, see `awslogs-datetime-format`.

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

---

**Note**

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

---

`awslogs-multiline-pattern`

Required: No

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

For more information, see `awslogs-multiline-pattern` in the Docker documentation.

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

---

**Note**

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

---

`awslogs-create-group`

Required: No

Specify whether you want the log group automatically created. If this option isn't specified, it defaults to false.
### Warning

This option isn't recommended. We recommend that you create the log group in advance using the CloudWatch Logs [CreateLogGroup](#) API action as each job tries to create the log group, increasing the chance that the job fails.

### Note

The IAM policy for your execution role must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

### Specifying a log configuration in your job definition

By default, AWS Batch enables the `awslogs` log driver. This section describes how to customize the `awslogs` log configuration for a job. For more information, see [Creating a single-node job definition](#).

The following log configuration JSON snippets have a `logConfiguration` object specified for each job. One is for a WordPress job that sends logs to a log group called `awslogs-wordpress`, and another is for a MySQL container that sends logs to a log group called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```json
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group": "awslogs-wordpress",
      "awslogs-stream-prefix": "awslogs-example"
   }
}

"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group": "awslogs-mysql",
      "awslogs-stream-prefix": "awslogs-example"
   }
}
```
In the AWS Batch console, the log configuration for the wordpress job definition is specified as shown in the following image.

![Log configuration]

After you have registered a task definition with the awslogs log driver in a job definition log configuration, you can submit a job with that job definition to start sending logs to CloudWatch Logs. For more information, see Submitting a job.

Specifying sensitive data

With AWS Batch, you can inject sensitive data into your jobs by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters, and then reference them in your job definition.

Secrets can be exposed to a job in the following ways:

- To inject sensitive data into your containers as environment variables, use the secrets job definition parameter.
- To reference sensitive information in the log configuration of a job, use the secretOptions job definition parameter.

Topics

Specifying sensitive data
Specifying sensitive data using Secrets Manager

With AWS Batch, you can inject sensitive data into your jobs by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your job definition. Sensitive data stored in Secrets Manager secrets can be exposed to a job as environment variables or as part of the log configuration.

When you inject a secret as an environment variable, you can specify a JSON key or version of a secret to inject. This process helps you control the sensitive data exposed to your job. For more information about secret versioning, see Key Terms and Concepts for AWS Secrets Manager in the AWS Secrets Manager User Guide.

Considerations for specifying sensitive data using Secrets Manager

The following should be considered when using Secrets Manager to specify sensitive data for jobs.

- To inject a secret using a specific JSON key or version of a secret, the container instance in your compute environment must have version 1.37.0 or later of the Amazon ECS container agent installed. 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 in the Amazon Elastic Container Service Developer Guide.

- 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 1.23.0 or later of the container agent.

- 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 aren't supported.

- When using a job definition that references Secrets Manager secrets to retrieve sensitive data for your jobs, if you're also using interface VPC endpoints, 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.

- Sensitive data is injected into your job when the job is initially started. If the secret is subsequently updated or rotated, the job doesn't receive the updated value automatically. You must launch a new job to force the service to launch a fresh job with the updated secret value.
Required IAM permissions for AWS Batch secrets

To use this feature, you must have the execution role and reference it in your job definition. This allows the container agent to pull the necessary Secrets Manager resources. For more information, see [AWS Batch execution IAM role](#).

To provide access to the Secrets Manager secrets that you create, manually add the following permissions as an inline policy to the execution role. For more information, see [Adding and Removing IAM Policies](#) in the [IAM User Guide](#).

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

The following example inline policy adds the required permissions.

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

Injecting sensitive data as an environment variable

Within your job definition, you can specify the following items:

- The `secrets` object containing the name of the environment variable to set in the job
- The Amazon Resource Name (ARN) of the Secrets Manager secret
- Additional parameters that contain the sensitive data to present to the job
The following example shows the full syntax that must be specified for the Secrets Manager secret.

```
```

The following section describes the additional parameters. These parameters are optional. However, if you don't use them, you must include the colon `:` 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 don't 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 can't 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](https://docs.aws.amazon.com/secretsmanager/latest/userguide/key-terms-and-concepts.html) 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 can't 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](https://docs.aws.amazon.com/secretsmanager/latest/userguide/key-terms-and-concepts.html) in the *AWS Secrets Manager User Guide*.

**Example container definitions**

The following examples show ways that 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.

```json
{
    "containerProperties": {{
        "secrets": [[
            "name": "environment_variable_name",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
        ]]
    }}
}
```

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.

```json
{
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981dd39ab30c",
    "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.

```json
{
    "containerProperties": {{
        "secrets": [[
            "name": "environment_variable_name",
        ]]
}
```
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.

```
{
    "Name": "apauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981dd39ab30c": [ "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead36b967e8": [ "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.

```
{
    "containerProperties": [{
        "secrets": [{
            "name": "environment_variable_name",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:apauthexample-AbCdEf::AWSPREVIOUS"
        }]
    }]
}
```

Reference a specific version ID from the previous output in a container definition by specifying the key name at the end of the ARN.
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.

```
{
    "containerProperties": [{
        "secrets": [{
            "name": "environment_variable_name",
        }]
    }]
}
```

To specify a specific key and version ID, use the following syntax.

```
{
    "containerProperties": [{
        "secrets": [{
            "name": "environment_variable_name",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-ceed36b967e8"
        }]
    }]
}
```
Injecting sensitive data in a log configuration

Within your job 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 job definition showing the format when referencing an Secrets Manager secret.

```
{
   "containerProperties": [{
      "logConfiguration": [{
         "logDriver": "splunk",
         "options": {
            "splunk-url": "https://cloud.splunk.com:8080"
         },
         "secretOptions": [{
            "name": "splunk-token",
            "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
         }]
      }]
   }
}
```

Creating an AWS Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For more information, see [Creating a Basic Secret](https://docs.aws.amazon.com/secretsmanager/latest/userguide/creating-a-basic-secret.html) in the *AWS Secrets Manager User Guide*.

**To create a basic secret**

Use Secrets Manager to create a secret for your sensitive data.

2. Choose Store a new secret.
3. For Select secret type, choose Other type of secrets.
4. Specify the details of your custom secret as Key and Value pairs. For example, you can specify a key of UserName, and then supply the appropriate user name as its value. Add a second key with the name of Password and the password text as its value. You could also add entries for
a database name, server address, or TCP port. You can add as many pairs as you need to store the information you require.

Alternatively, you can choose the Plaintext tab and enter the secret value in any way you like.

5. Choose the AWS KMS encryption key that you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks to see if there's a default key for the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose Add new key to create a custom KMS key specifically for this secret. To create your own KMS key, you must have permissions to create KMS keys in your account.

6. Choose Next.

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

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

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

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

Specifying sensitive data using Systems Manager Parameter Store

With AWS Batch, you can 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.

Topics

- Considerations for specifying sensitive data using Systems Manager Parameter Store
- Required IAM permissions for AWS Batch secrets
- Injecting sensitive data as an environment variable
- Injecting sensitive data in a log configuration
- Creating an AWS Systems Manager Parameter Store parameter
Considerations for specifying sensitive data using Systems Manager Parameter Store

The following should be considered when specifying sensitive data for containers using Systems Manager Parameter Store parameters.

- This feature requires that your container instance have version 1.23.0 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](https://docs.aws.amazon.com/elasticcontainerservice/latest/dg/ecs-update-agent.html) in the Amazon Elastic Container Service Developer Guide.

- Sensitive data is injected into the container for your job when the container is initially started. If the secret or Parameter Store parameter is subsequently updated or rotated, the container doesn't receive the updated value automatically. You must launch a new job to force the launch of a fresh job with updated secrets.

Required IAM permissions for AWS Batch secrets

To use this feature, you must have the execution role and reference it in your job definition. This allows the Amazon ECS container agent to pull the necessary AWS Systems Manager resources. For more information, see [AWS Batch execution IAM role](https://docs.aws.amazon.com/batch/latest/userguide/batch-execution-iam-role.html).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions as an inline policy to the execution role. For more information, see [Adding and Removing IAM Policies](https://docs.aws.amazon.com/IAM/latest/UserGuide/IAM-Policies.html) in the IAM User Guide.

- `ssm:GetParameters`—Required if you're referencing a Systems Manager Parameter Store parameter in a task definition.
- `secretsmanager:GetSecretValue`—Required if you're 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.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

```json
{
  "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:kms:<region>:<aws_account_id>:key/<key_id>"
        ]
    }
]
}

**Injecting sensitive data as an environment variable**

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.

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same Region as the task that you're launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

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

**Injecting 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're launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter.

```json
{
    "containerProperties": [{
        "logConfiguration": [{
            "logDriver": "fluentd",
            "options": {
                "tag": "fluentd demo"
            },
            "secretOptions": [{
                "name": "fluentd-address",
                "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
            }]
        }]
    }
}
```

Creating an AWS Systems Manager Parameter Store parameter

You can use the AWS Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see Walkthrough: Create and Use a Parameter in a Command (Console) in the AWS Systems Manager User Guide.

To create a Parameter Store parameter

2. In the navigation pane, choose Parameter Store, Create parameter.
3. For **Name**, type a hierarchy and a parameter name. For example, type test/database_password.

4. For **Description**, type an optional description.

5. For **Type**, choose **String**, **StringList**, or **SecureString**.

---

### Note

- If you choose **SecureString**, the **KMS Key ID** field appears. If you don't provide a KMS key ID, a KMS key ARN, an alias name, or an alias ARN, then the system uses alias/aws/ssm. This is the default KMS key for Systems Manager. To avoid using this key, choose a custom key. For more information, see [Use Secure String Parameters](https://docs.aws.amazon.com/systems-manager/latest/userguide/use-secure-string-parameters.html) in the *AWS Systems Manager User Guide*.

- When you create a secure string parameter in the console by using the `key-id` parameter with either a custom KMS key alias name or an alias ARN, you must specify the prefix `alias/` before the alias. The following is an ARN example:

```plaintext
arn:aws:kms:us-east-2:123456789012:alias/MyAliasName
```

The following is an alias name example:

```plaintext
alias/MyAliasName
```

6. For **Value**, type a value. For example, MyFirstParameter. If you chose **SecureString**, the value is masked exactly as you entered it.

7. Choose **Create parameter**.

---

### Private registry authentication for jobs

Private registry authentication for jobs using AWS Secrets Manager enables you to store your credentials securely and then reference them in your job definition. This provides a way to reference container images that exist in private registries outside of AWS that require authentication in your job definitions. This feature is supported by jobs hosted on Amazon EC2 instances and Fargate.
Important

If your job 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 jobs 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 in the Amazon Elastic Container Service Developer Guide.

For jobs hosted on Fargate, this feature requires platform version 1.2.0 or later. For information, see AWS Fargate Linux platform versions in the Amazon Elastic Container Service Developer Guide.

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 job using it.

Note

When using the AWS Batch API, AWS CLI, or AWS SDK, if the secret exists in the same AWS Region as the job 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 job definition that shows the required parameters:

```
"containerProperties": [
  {
    "image": "private-repo/private-image",
    "repositoryCredentials": {
      "credentialsParameter":
        "arn:aws:secretsmanager:region:123456789012:secret:secret_name"
    }
  }
]
```
Required IAM permissions for private registry authentication

The execution role is required to use this feature. This allows the container agent to pull the container image. For more information, see AWS Batch execution IAM role.

To provide access to the secrets that you create, add the following permissions as an inline policy to the 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:kms:region:123456789012: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:

```json
{
    "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 job 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. For more information, see Required IAM permissions for private registry authentication.

### Amazon EFS volumes

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your AWS Batch jobs. With Amazon EFS, storage capacity is elastic. It scales 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 AWS Batch to export file system data across your fleet of container instances. That way, your jobs have access to the same persistent storage. However, you must configure your container instance AMI to mount the Amazon EFS file system before the Docker daemon starts. Also, your job definitions must reference volume mounts on the container instance to use the file system. The following sections help you get started using Amazon EFS with AWS Batch.
Amazon EFS volume considerations

The following should be considered when using Amazon EFS volumes:

- For jobs using EC2 resources, 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 take advantage of these features. For more information, see Amazon ECS optimized AMI versions in the Amazon Elastic Container Service Developer Guide.

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. This is all 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 jobs using Fargate resources, Amazon EFS file system support was added when using platform version 1.4.0 or later. For more information, see AWS Fargate platform versions in the Amazon Elastic Container Service Developer Guide.

- When specifying Amazon EFS volumes in jobs using Fargate resources, Fargate creates a supervisor container that is responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the job's memory. The supervisor container is visible when querying the task metadata version 4 endpoint. For more information, see Task metadata endpoint version 4 in the Amazon Elastic Container Service User Guide for AWS Fargate.
Using Amazon EFS access points

Amazon EFS access points are application-specific entry points into an EFS file system that help you to manage 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 so that clients can only access data in the specified directory or its subdirectories.

Note

When creating an EFS access point, you specify a path on the file system to serve as the root directory. When you reference the EFS file system with an access point ID in your AWS Batch job definition, the root directory must either be omitted or set to / This enforces the path that's set on the EFS access point.

You can use an AWS Batch job IAM role to enforce that specific applications use a specific access point. By combining IAM policies with access points, you can easily provide secure access to specific datasets for your applications. This feature uses Amazon ECS IAM roles for task functionality. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

Specifying an Amazon EFS file system in your job definition

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

```json
{
    "containerProperties": [
        {
            "image": "amazonlinux:2",
            "command": [
                "ls",
                "-la",
                "/mount/efs"
            ]
        }
    ]
}
```
"mountPoints": [
    {
        "sourceVolume": "myEfsVolume",
        "containerPath": "/mount/efs",
        "readOnly": true
    }
],
"volumes": [
    {
        "name": "myEfsVolume",
        "efsVolumeConfiguration": {
            "fileSystemId": "fs-12345678",
            "rootDirectory": "/path/to/my/data",
            "transitEncryption": "ENABLED",
            "transitEncryptionPort": integer,
            "authorizationConfig": {
                "accessPointId": "fsap-1234567890abcdef1",
                "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
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. It can be up to 4,096 characters in length.

**Important**

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

`transitEncryption`

Type: String

Valid values: ENABLED | DISABLED

Required: No

Determines whether to enable encryption for Amazon EFS data that's in transit between the AWS Batch 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](https://docs.aws.amazon.com/efs/latest/ug/encrypt-data-in-transit.html) in the Amazon Elastic File System User Guide.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the AWS Batch 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. The value must be between 0 and 65,535. For more information, see [EFS Mount Helper](https://docs.aws.amazon.com/efs/latest/ug/efs-mount-helper.html) 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 /. This enforces the path that's 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

Determines whether to use the AWS Batch job IAM role that's defined in a job 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 about execution IAM roles, see AWS Batch execution IAM role.

Example job definitions

The following example job definitions illustrate how to use common patterns such as environment variables, parameter substitution, and volume mounts.

Use environment variables

The following example job definition uses environment variables to specify a file type and Amazon S3 URL. This particular example is from the Creating a Simple "Fetch & Run" AWS Batch Job compute blog post. The fetch_and_run.sh script that's described in the blog post uses these environment variables to download the myjob.sh script from S3 and declare its file type.
Even though the command and environment variables are hardcoded into the job definition in this example, you can specify command and environment variable overrides to make the job definition more versatile.

```json
{
    "jobDefinitionName": "fetch_and_run",
    "type": "container",
    "containerProperties": {
        "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/fetch_and_run",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "2000"
            },
            {
                "type": "VCPU",
                "value": "2"
            }
        ],
        "command": [
            "myjob.sh",
            "60"
        ],
        "jobRoleArn": "arn:aws:iam::123456789012:role/AWSBatchS3ReadOnly",
        "environment": [
            {
                "name": "BATCH_FILE_S3_URL",
                "value": "s3://my-batch-scripts/myjob.sh"
            },
            {
                "name": "BATCH_FILE_TYPE",
                "value": "script"
            }
        ],
        "user": "nobody"
    }
}
```

**Using parameter substitution**

The following example job definition illustrates how to allow for parameter substitution and to set default values.
The `Ref::` declarations in the command section are used to set placeholders for parameter substitution. When you submit a job with this job definition, you specify the parameter overrides to fill in those values, such as the `inputfile` and `outputfile`. The parameters section that follows sets a default for `codec`, but you can override that parameter as needed.

For more information, see [Parameters](#).

```json
{
    "jobDefinitionName": "ffmpeg_parameters",
    "type": "container",
    "parameters": {
        "codec": "mp4"
    },
    "containerProperties": {
        "image": "my_repo/ffmpeg",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "2000"
            },
            {
                "type": "VCPU",
                "value": "2"
            }
        ],
        "command": [
            "ffmpeg",
            "-i",
            "Ref::inputfile",
            "-c",
            "Ref::codec",
            "-o",
            "Ref::outputfile"
        ],
        "jobRoleArn": "arn:aws:iam::123456789012:role/ECSTask-S3FullAccess",
        "user": "nobody"
    }
}
```

**Test GPU functionality**

The following example job definition tests if the GPU workload AMI described in [Using a GPU workload AMI](#) is configured properly. This example job definition runs the TensorFlow deep MNIST classifier example [example](#) from GitHub.
You can create a file with the preceding JSON text called `tensorflow_mnist_deep.json` and then register an AWS Batch job definition with the following command:

```
aws batch register-job-definition --cli-input-json file://tensorflow_mnist_deep.json
```

### Multi-node parallel job

The following example job definition illustrates a multi-node parallel job. For more information, see Building a tightly coupled molecular dynamics workflow with multi-node parallel jobs in AWS Batch in the AWS Compute blog.

```
{
    "jobDefinitionName": "gromacs-jobdef",
    "revision": 6,
    "status": "ACTIVE",
    "type": "multinode",
```
"parameters": {},
"nodeProperties": {
  "numNodes": 2,
  "mainNode": 0,
  "nodeRangeProperties": [
    {
      "targetNodes": "0:1",
      "container": {
        "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/gromacs_mpi:latest",
        "resourceRequirements": [
          {
            "type": "MEMORY",
            "value": "24000"
          },
          {
            "type": "VCPU",
            "value": "8"
          }
        ],
        "command": [],
        "jobRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
        "ulimits": [],
        "instanceType": "p3.2xlarge"
      }
    }
  ]
}
Job queues

Jobs are submitted to a job queue where they reside until they can be scheduled to run in a compute environment. An AWS account can have multiple job queues. For example, you can create a queue that uses Amazon EC2 On-Demand instances for high priority jobs and another queue that uses Amazon EC2 Spot Instances for low-priority jobs. Job queues have a priority that's used by the scheduler to determine which jobs in which queue should be evaluated for execution first.

Topics
- Creating a job queue
- Job queue parameters

Creating a job queue

Before you can submit jobs in AWS Batch, you must create a job queue. When you create a job queue, you associate one or more compute environments to the queue and assign an order of preference.

You also set priority to the job queue that determines the order that the AWS Batch scheduler places jobs. This means that, if a compute environment is associated with more than one job queue, the job queue with a higher priority is given preference.

Creating a Fargate job queue

To create a Fargate job queue

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Job queues.
4. Choose Create.
5. For Orchestration type, choose Fargate.
6. For Name, enter a unique name for your job queue. The name can be up to 128 characters long, and can contain uppercase and lowercase letters, numbers, and underscores (_).
7. For Priority, enter an whole number value for the job queue's priority. Job queues with a higher priority are run before lower priority job queues that are associated with the same
compute environment. Priority is determined in descending order. For example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.

8. (Optional) For Scheduling policy Amazon Resource Name (ARN), choose an existing scheduling policy.

9. For Connected compute environments, select one or more compute environments from the list to associate with the job queue. Select compute environments in the order that you want the queue to attempt job queue placement. The job scheduler uses the order that you select compute environments in to determine which compute environment starts a given job. Before you can associate them with a job queue, compute environments must be in the VALID state. You can associate up to three compute environments with a job queue.

10. For Compute environment order, choose the up and down arrows to configure order that you want.

11. Choose Create job queue to finish and create your job queue.

Creating an Amazon EC2 job queue

To create an Amazon EC2 job queue

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Job queues.
4. Choose Create.
5. For Orchestration type, choose Amazon Elastic Compute Cloud (Amazon EC2).
6. For Name, enter a unique name for your job queue. The name can be up to 128 characters long, and can contain uppercase and lowercase letters, numbers, and underscores (_).
7. For Priority, enter an whole number value for the job queue's priority. Job queues with a higher priority are run before lower priority job queues that are associated with the same
compute environment. Priority is determined in descending order. For example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.

8. (Optional) For **Scheduling policy Amazon Resource Name (ARN)**, choose an existing scheduling policy.

9. For **Connected compute environments**, select one or more compute environments from the list to associate with the job queue. Select compute environments in the order that you want the queue to attempt job queue placement. The job scheduler uses the order that you select compute environments in to determine which compute environment starts a given job. Before you can associate them with a job queue, compute environments must be in the VALID state. You can associate up to three compute environments with a job queue. If you don't have an existing compute environment, choose **Create compute environment**

   **Note**

   All compute environments that are associated with a job queue must share the same provisioning model. AWS Batch doesn't support mixing provisioning models in a single job queue.

10. For **Compute environment order**, choose the up and down arrows to configure order that you want.

11. Choose **Create job queue** to finish and create your job queue.

### Creating an Amazon EKS job queue

**To create an Amazon EKS job queue**

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose **Job queues**.
4. Choose **Create**.
5. For **Orchestration type**, choose **Amazon Elastic Kubernetes Service (Amazon EKS)**.
6. For **Name**, enter a unique name for your job queue. The name can be up to 128 characters long, and can contain uppercase and lowercase letters, numbers, and underscores (_).
7. For **Priority**, enter an integer value for the job queue's priority. Job queues with a higher priority are run before lower priority job queues that are associated with the same compute environment. Priority is determined in descending order. For example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.

8. (Optional) For **Scheduling policy Amazon Resource Name (ARN)**, choose an existing scheduling policy.

9. For **Connected compute environments**, select one or more compute environments from the list to associate with the job queue. Select compute environments in the order that you want the queue to attempt job queue placement. The job scheduler uses the order that you select compute environments in to determine which compute environment starts a given job. Before you can associate them with a job queue, compute environments must be in the VALID state. You can associate up to three compute environments with a job queue.

   **Note**
   All compute environments that are associated with a job queue must share the same provisioning model. AWS Batch doesn't support mixing provisioning models in a single job queue.

   **Note**
   All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn't support mixing compute environment architecture types in a single job queue.

10. For **Compute environment order**, choose the up and down arrows to configure order that you want.

11. Choose **Create job queue** to finish and create your job queue.

**Job queue template**

The following is an empty job queue template. You can use this template to create your job queue. You can then save this job queue to a file and use it with the AWS CLI `--cli-input-json` option. For more information about these parameters, see [CreateJobQueue](https://docs.aws.amazon.com/batch/latest/APIReference/API_CreateJobQueue.html) in the *AWS Batch API Reference*. 

---

*Job queue template*
Job queue parameters

Job queues are split into four basic components: name, state, priority, and compute environment order. This section discusses these components associated components.

Topics

- Job queue name
- Job queue state time limit actions
• **Priority**

• **Scheduling policy**

• **State**

• **Compute environment order**

• **Tags**

## Job queue name

**jobQueueName**

The name for your job queue. Up to 128 letters (uppercase and lowercase), numbers, and underscores are allowed.

Type: String

Required: Yes

## Job queue state time limit actions

**jobStateTimeLimitActions**

The set of actions that AWS Batch performs on jobs that remain at the head of the job queue in the specified state longer than specified times. AWS Batch will perform each action after `maxTimeSeconds` has passed. *(Note: The minimum value for `maxTimeSeconds` is 600 (10 minutes) and its maximum value is 86,400 (24 hours).)*

Type: Array of JobStateTimeLimitActions objects

Required: No

## Priority

**priority**

The priority of the job queue. Job queues with a higher priority (or a higher integer value for the `priority` parameter) are evaluated first when associated with same compute environment.
Priority is determined in descending order, for example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1. All of the compute environments must be either Amazon EC2 (EC2 or SPOT) or Fargate (FARGATE or FARGATE_SPOT). Amazon EC2 and Fargate compute environments can't be mixed.

Type: Integer

Required: Yes

**Scheduling policy**

`schedulingPolicyArn`

The Amazon Resource Name (ARN) of the scheduling policy for the job queue. Job queues that don't have a scheduling policy are scheduled in a first-in, first-out (FIFO) model. After a job queue has a scheduling policy, it can be replaced but can't be removed. A job queue without a scheduling policy is scheduled as a FIFO job queue and can't have a scheduling policy added. Jobs queues with a scheduling policy can have a maximum of 500 active fair share identifiers. When the limit has been reached, submissions of any jobs that add a new fair share identifier fail.

Type: String

Required: No

**State**

`state`

The state of the job queue. If the job queue state is ENABLED (the default value), it can accept jobs. If the job queue state is DISABLED, new jobs can't be added to the queue, but jobs already in the queue can finish.

Type: String

Valid values: ENABLED | DISABLED

Required: No
**Compute environment order**

**computeEnvironmentOrder**

The set of compute environments mapped to a job queue and their order relative to each other. The job scheduler uses this parameter to determine which compute environment should run a specific job. Compute environments must be in the **VALID** state before you can associate them with a job queue. You can associate up to three compute environments with a job queue. All of the compute environments must be either Amazon EC2 (EC2 or SPOT) or Fargate (FARGATE or FARGATE_SPOT). Amazon EC2 and Fargate compute environments can't be mixed.

**Note**

All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn't support mixing compute environment architecture types in a single job queue.

Type: Array of **ComputeEnvironmentOrder** objects

Required: Yes

**computeEnvironment**

The Amazon Resource Name (ARN) of the compute environment.

Type: String

Required: Yes

**order**

The order of the compute environment. Compute environments are tried in ascending order. For example, if two compute environments are associated with a job queue, the compute environment with a lower `order` integer value is tried for job placement first.

**Tags**

**tags**

Key-value pair tags to associate with the job queue. For more information, see [Tagging your AWS Batch resources](#).
Type: String to string map

Required: No
Job scheduling

The AWS Batch scheduler evaluates when, where, and how to run jobs that are submitted to a job queue. If you don’t specify a scheduling policy when you create a job queue, the AWS Batch job scheduler defaults to a first-in, first-out (FIFO) strategy. A FIFO strategy might cause important jobs to get “stuck” behind jobs that were submitted earlier. By specifying a different scheduling policy, you can allocate compute resources according to your specific needs.

Note

If you want to schedule the specific order that jobs are run in, use the `dependsOn` parameter in `SubmitJob` to specify the dependencies for each job.

If you create a scheduling policy and attach it to a job queue, fair share scheduling is turned on. If the job queue has a scheduling policy, the scheduling policy determines the order that jobs are run in. For more information, see Scheduling policies.

Share identifiers

You can use share identifiers to tag jobs and differentiate between users and workloads. The AWS Batch scheduler tracks usage for each fair share identifier by using the \( T \times weightFactor \) formula, where \( T \) is the vCPU usage over time. The scheduler picks jobs with the lowest usage from the share identifier. You can use a fair share identifier without overriding it.

Note

Share identifiers are unique within a job queue and are not aggregated across job queues.

You can set scheduling priority to configure the order that jobs are run in on a share identifier. Jobs with a higher scheduling priority are scheduled first. If you don't specify a scheduling policy, all jobs that are submitted to the job queue are scheduled in FIFO order. When you submit a job, you can't specify a share identifier or scheduling priority.
Attached compute resources are allocated equally among all share identifiers unless explicitly overridden.

### Fair share scheduling

Fair share scheduling provides a set of controls to help schedule jobs.

- **Share decay seconds** – The period of time (in seconds) that the AWS Batch scheduler uses to calculate a fair share percentage for each fair share identifier. A value of zero indicates that only current usage is measured. A longer decay time gives more weight to time.

  
  ![Note]
  
  The period of time for decay is calculated as: \( \text{shareDecaySeconds} + \text{OrderMinutes} \) where \( \text{OrderMinutes} \) is the time in the order in minutes.

- **Compute reservation** – Prevents jobs in a single share identifier from using up all the resources that are attached to the job queue. The reserved ratio is \( \text{computeReservation}/100)^\text{ActiveFairShares} \) where \( \text{ActiveFairShares} \) is the number of active fair share identifiers.

  
  ![Note]
  
  If a share identifier has jobs in a SUBMITTED, PENDING, Runnable, STARTING, or RUNNING state, it's considered an active share identifier. After the period of time for decay expires, a share identifier is considered inactive.

- **Weight factor** – The weight factor for the share identifier. The default value is 1. A lower value lets jobs from the share identifier run or gives additional runtime to the share identifier.
example, jobs that use a share identifier with a weight factor of 0.125 (1/8) are assigned eight times the compute resources of jobs that use a share identifier with a weight factor of 1.

Note
You only need to define this attribute when you need to update the default weight factor of 1.
Compute environment

Job queues are mapped to one or more compute environments. Compute environments contain the Amazon ECS container instances that are used to run containerized batch jobs. A specific compute environment can also be mapped to one or more than one job queue. Within a job queue, the associated compute environments each have an order that's used by the scheduler to determine where jobs that are ready to be run will run. If the first compute environment has a status of VALID and has available resources, the job is scheduled to a container instance within that compute environment. If the first compute environment has a status of INVALID or can't provide a suitable compute resource, the scheduler attempts to run the job on the next compute environment.

Topics
- Managed compute environments
- Unmanaged compute environments
- Compute resource AMIs
- Launch template support
- Creating a compute environment
- Compute environment template
- Compute environment parameters
- EC2 Configurations
- Allocation strategies
- Updating compute environments
- Amazon EKS compute environments
- Compute Resource Memory Management

Managed compute environments

You can use a managed compute environment to have AWS Batch manage the capacity and instance types of the compute resources within the environment. This is based on the compute resource specifications that you define when you create the compute environment. You can choose either to use Amazon EC2 On-Demand Instances and Amazon EC2 Spot Instances. Or, you can
alternatively use Fargate and Fargate Spot capacity in your managed compute environment. When using Spot Instances, you can optionally set a maximum price. This way, Spot Instances only launch when the Spot Instance price is under a specified percentage of the On-Demand price.

⚠️ Important

Fargate Spot instances are not supported on Windows containers on AWS Fargate. A job queue will be blocked if a FargateWindows job is submitted to a job queue that only uses Fargate Spot compute environments.

Managed compute environments launch Amazon EC2 instances into the VPC and subnets that you specify and then registers them with an Amazon ECS cluster. The Amazon EC2 instances need external network access to communicate with the Amazon ECS service endpoint. Some subnets don't provide Amazon EC2 instances with public IP addresses. If your Amazon EC2 instances don't have a public IP address, they must use network address translation (NAT) to gain this access. For more information, see [NAT gateways](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/) in the *Amazon VPC User Guide*. For more information about how to create a VPC, see [Creating a virtual private cloud](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/). By default, AWS Batch managed compute environments use a recent, approved version of the Amazon ECS optimized AMI for compute resources. However, you might want to create your own AMI to use for your managed compute environments for various reasons. For more information, see [Compute resource AMIs](https://docs.aws.amazon.com/batch/latest/userguide/).
In April 2022, AWS Batch added enhanced support for updating compute environments. For more information, see [Updating compute environments](#). To use the enhanced updating of compute environments to update AMIs, follow these rules:

- Either don't set the service role (`serviceRole`) parameter or set it to the `AWSServiceRoleForBatch` service-linked role.
- Set the allocation strategy (`allocationStrategy`) parameter to `BEST_FIT_PROGRESSIVE`, `SPOT_CAPACITY_OPTIMIZED` or `SPOT_PRICE_CAPACITY_OPTIMIZED`.
- Set the update to latest image version (`updateToLatestImageVersion`) parameter to `true`.
- Don't specify an AMI ID in `imageId`, `imageIdOverride` (in `ec2Configuration`), or in the launch template (`LaunchTemplate`). In that case, AWS Batch selects the latest Amazon ECS optimized AMI that's supported by AWS Batch at the time the infrastructure update is initiated. Alternatively, you can specify the AMI ID in the `imageId` or `imageIdOverride` parameters, or the launch template identified by the `LaunchTemplate` properties. Changing any of these properties starts an infrastructure update. If the AMI ID is specified in the launch template, it can't be replaced by specifying an AMI ID in either the `imageId` or `imageIdOverride` parameters. It can only be replaced by specifying a different launch template. Or, if the launch template version is set to `$Default` or `$Latest`, by setting either a new default version for the launch template (if it's `$Default`) or by adding a new version to the launch template (if it's `$Latest`).

If these rules are followed, any update that starts an infrastructure update will cause the AMI ID to be re-selected. If the `version` setting in the launch template (`LaunchTemplate`) is set to `$Latest` or `$Default`, the latest or default version of the launch template are evaluated up at the time of the infrastructure update, even if the `LaunchTemplate` was not updated.

**Consideration when creating multi-node parallel jobs**

AWS Batch recommends creating dedicated compute environments for running multi-node parallel (MNP) jobs and non-MNP jobs. This is due to the way compute capacity is created in your managed
compute environment. When creating a new managed compute environment, if you specify a 
minvCpu value greater than zero then AWS Batch creates an instance pool for use with non-MNP 
jobs only. If a multi-node parallel job is submitted, AWS Batch creates new instance capacity to run 
the multi-node parallel jobs. In cases where there are both single-node and multi-node parallel 
jobs running in the same compute environment where either a minvCpus or maxvCpus value is 
set, if the required compute resources are unavailable AWS Batch will wait for the current jobs to 
finish before creating the compute resources necessary to run the new jobs.

**Unmanaged compute environments**

In an unmanaged compute environment, you manage your own compute resources. You must 
verify that the AMI you use for your compute resources meets the Amazon ECS container instance 
AMI specification. For more information, see [Compute resource AMI specification](#) and [Creating a 
compute resource AMI](#).

> **Note**
> AWS Fargate resources aren't supported in unmanaged compute environments.

After you created your unmanaged compute environment, use the `DescribeComputeEnvironments` API operation to view the compute environment details. Find the Amazon ECS cluster that's 
associated with the environment and then manually launch your container instances into that 
Amazon ECS cluster.

The following AWS CLI command also provides the Amazon ECS cluster ARN.

```
$ aws batch describe-compute-environments \
   --compute-environments unmanagedCE \
   --query "computeEnvironments[].ecsClusterArn"
```

For more information, see [Launching an Amazon ECS container instance](#) in the *Amazon Elastic 
Container Service Developer Guide*. When you launch your compute resources, specify the Amazon 
ECS cluster ARN that the resources register with the following Amazon EC2 user data. Replace 
`ecsClusterArn` with the cluster ARN that you obtained with the previous command.

```
#!/bin/bash
echo "ECS_CLUSTER=ecsClusterArn" >> /etc/ecs/ecs.config
```
Compute resource AMIs

By default, AWS Batch managed compute environments use a recent, approved version of the Amazon ECS optimized AMI for compute resources. However, you might want to create your own AMI to use for your managed and unmanaged compute environments. If you require any of the following, we recommend you create your own AMI:

- Increasing the storage size of your AMI root or data volumes
- Adding instance storage volumes for supported Amazon EC2 instance types
- Customizing the Amazon ECS container agent
- Customizing Docker
- Configuring a GPU workload AMI to allow containers to access GPU hardware on supported Amazon EC2 instance types

⚠️ **Note**

After a compute environment is created, AWS Batch doesn't upgrade the AMIs in the compute environment. AWS Batch also doesn't update the AMIs in your compute environment when a newer version of the Amazon ECS optimized AMI is available. You're responsible for the management of the guest operating system. This includes any updates and security patches. You're also responsible for any additional application software or utilities that you install on the compute resources. To use a new AMI for your AWS Batch jobs, do the following:

1. Create a new compute environment with the new AMI.
2. Add the compute environment to an existing job queue.
3. Remove the earlier compute environment from your job queue.
4. Delete the earlier compute environment.

In April 2022, AWS Batch added enhanced support for updating compute environments. For more information, see [Updating compute environments](#). To use the enhanced updating of compute environments to update AMIs, follow these rules:

- Either don't set the service role (`serviceRole`) parameter or set it to the `AWSServiceRoleForBatch` service-linked role.
• Set the allocation strategy (allocationStrategy) parameter to BEST_FIT_PROGRESSIVE, SPOT_CAPACITY_OPTIMIZED, or SPOT_PRICE_CAPACITY_OPTIMIZED.

• Set the update to latest image version (updateToLatestImageVersion) parameter to true.

• Don’t specify an AMI ID in imageId, imageIdOverride (in ec2Configuration), or in the launch template (launchTemplate). When you don’t specify an AMI ID, AWS Batch selects the latest Amazon ECS optimized AMI that AWS Batch supports at the time the infrastructure update is initiated. Alternatively, you can specify the AMI ID in the imageId or imageIdOverride parameters. Or, you can specify the launch template that’s identified by the LaunchTemplate properties. Changing any of these properties starts an infrastructure update. If the AMI ID is specified in the launch template, the AMI ID can’t be replaced by specifying an AMI ID in either the imageId or imageIdOverride parameters. The AMI ID can only be replaced by specifying a different launch template. If the launch template version is set to $Default or $Latest, the AMI ID can be replaced by setting either a new default version for the launch template (if $Default) or by adding a new version to the launch template (if $Latest).

If these rules are followed, any update that starts an infrastructure update causes the AMI ID to be re-selected. If the version setting in the launch template (launchTemplate) is set to $Latest or $Default, the latest or default version of the launch template is evaluated up at the time of the infrastructure update, even if the LaunchTemplate wasn’t updated.

Topics

• Compute resource AMI specification

• Creating a compute resource AMI

• Using a GPU workload AMI

• Amazon Linux deprecation

Compute resource AMI specification

The basic AWS Batch compute resource AMI specification consists of the following:
Required

- A modern Linux distribution that's running at least version 3.10 of the Linux kernel on an HVM virtualization type AMI. Windows containers aren't supported.

⚠️ Important

Multi-node parallel jobs can only run on compute resources that were launched on an Amazon Linux instance with the ecs-init package installed. We recommend that you use the default Amazon ECS optimized AMI when you create your compute environment. You can do this by not specifying a custom AMI. For more information, see Multi-node parallel jobs.

- The Amazon ECS container agent. We recommend that you use the latest version. For more information, see Installing the Amazon ECS Container Agent in the Amazon Elastic Container Service Developer Guide.

- The awslogs log driver must be specified as an available log driver with the ECS_AVAILABLE_LOGGING_DRIVERS environment variable when the Amazon ECS container agent is started. For more information, see Amazon ECS Container Agent Configuration in the Amazon Elastic Container Service Developer Guide.

- A Docker daemon that's running at least version 1.9, and any Docker runtime dependencies. For more information, see Check runtime dependencies in the Docker documentation.

⚠️ Note

We recommend the Docker version that ships with and is tested with the corresponding Amazon ECS agent version that you're using. Amazon ECS provides a changelog for the Linux variant of the Amazon ECS-optimized AMI on GitHub. For more information, see Changelog.

Recommended

- An initialization and nanny process to run and monitor the Amazon ECS agent. The Amazon ECS optimized AMI uses the ecs-init upstart process, and other operating systems might use systemd. For more information and examples, see Example container instance User.
Data Configuration Scripts in the Amazon Elastic Container Service Developer Guide. For more information about ecs-init, see the ecs-init project on GitHub. At a minimum, managed compute environments require the Amazon ECS agent to start at boot. If the Amazon ECS agent isn't running on your compute resource, then it can't accept jobs from AWS Batch.

The Amazon ECS optimized AMI is preconfigured with these requirements and recommendations. We recommend that you use the Amazon ECS optimized AMI or an Amazon Linux AMI with the ecs-init package that's installed for your compute resources. Choose another AMI if your application requires a specific operating system or a Docker version that's not yet available in those AMIs. For more information, see Amazon ECS-Optimized AMI in the Amazon Elastic Container Service Developer Guide.

Creating a compute resource AMI

You can create your own custom compute resource AMI to use for your managed and unmanaged compute environments. For instructions, see the Compute resource AMI specification. Then, after you created a custom AMI, you can create a compute environment that uses that AMI that you can associate a job queue with. Last, start submitting jobs to that queue.

To create a custom compute resource AMI

1. Choose a base AMI to start from. The base AMI must use HVM virtualization. The base AMI can't be a Windows AMI.

Note

The AMI that you choose for a compute environment must match the architecture of the instance types that you want to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support Arm instances. Amazon ECS vend both x86 and Arm versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

The Amazon ECS optimized Amazon Linux 2 AMI is the default AMI for compute resources in managed compute environments. The Amazon ECS optimized Amazon Linux 2 AMI is preconfigured and tested on AWS Batch by AWS engineers. It's a minimal AMI that you can get
started with and to get your compute resources that are running on AWS quickly. For more information, see [Amazon ECS Optimized AMI](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/amazon-ecs-optimized-ami.html) in the *Amazon Elastic Container Service Developer Guide*.

Alternatively, you can choose another Amazon Linux 2 variant and install the *ecs-init* package with the following commands. For more information, see [Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-init.html) in the *Amazon Elastic Container Service Developer Guide*:

```
$ sudo amazon-linux-extras disable docker
$ sudo amazon-linux-extras install ecs-init
```

For example, if you want to run GPU workloads on your AWS Batch compute resources, you can start with the [Amazon Linux Deep Learning AMI](https://aws.amazon.com/amazon-linux-deep-learning-ami/). Then, configure the AMI to run AWS Batch jobs. For more information, see [Using a GPU workload AMI](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/iam-instance-profiles.html).

⚠️ **Important**

You can choose a base AMI that doesn't support the *ecs-init* package. However, if you do, you must configure a way to start the Amazon ECS agent at boot and keep it running. You can also view several example user data configuration scripts that use systemd to start and monitor the Amazon ECS container agent. For more information, see [Example container instance user data configuration scripts](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-instance-userscripts.html) in the *Amazon Elastic Container Service Developer Guide*.

2. Launch an instance from your selected base AMI with the appropriate storage options for your AMI. You can configure the size and number of attached Amazon EBS volumes, or instance storage volumes if the instance type you selected supports them. For more information, see [Launching an Instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-launch.html) and [Amazon EC2 Instance Store](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-store-intro.html) in the *Amazon EC2 User Guide for Linux Instances*.

3. Connect to your instance with SSH and perform any necessary configuration tasks. This might include any or all of the following steps:

   - Installing the Amazon ECS container agent. For more information, see [Installing the Amazon ECS Container Agent](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-agent-install.html) in the *Amazon Elastic Container Service Developer Guide*.
   - Configuring a script to format instance store volumes.
• Adding instance store volume or Amazon EFS file systems to the `/etc/fstab` file so that they're mounted at boot.

• Configuring Docker options, such as enabling debugging or adjusting base image size.

• Installing packages or copying files.

For more information, see [Connecting to Your Linux Instance Using SSH](#) in the *Amazon EC2 User Guide for Linux Instances*.

4. If you started the Amazon ECS container agent on your instance, you must stop it and remove any persistent data checkpoint files before creating your AMI. Otherwise, if you don't do this, the agent doesn't start on instances that are launched from your AMI.

   a. Stop the Amazon ECS container agent.

      • Amazon ECS-optimized Amazon Linux 2 AMI:

         ```bash
         sudo systemctl stop ecs
         ```

      • Amazon ECS-optimized Amazon Linux AMI:

         ```bash
         sudo stop ecs
         ```

   b. Remove the persistent data checkpoint files. By default, these files are located in the `/var/lib/ecs/data/` directory. Use the following command to remove these files, if there are any.

      ```bash
      sudo rm -rf /var/lib/ecs/data/*
      ```

5. Create a new AMI from your running instance. For more information, see [Creating an Amazon EBS Backed Linux AMI](#) in the *Amazon EC2 User Guide for Linux Instances* guide.

### To use your new AMI with AWS Batch

1. After the new AMI is created, create a compute environment with the new AMI. To do this, choose the image type and enter the custom AMI ID in the Image ID override box when you create the AWS Batch compute environment. For more information, see the section called “To create a managed compute environment using EC2 resources”.

The AMI that you choose for a compute environment must match the architecture of the instance types that you want to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support Arm instances. Amazon ECS vends both x86 and Arm versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

2. Create a job queue and associate your new compute environment. For more information, see Creating a job queue.

Note
All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn’t support mixing compute environment architecture types in a single job queue.

3. (Optional) Submit a sample job to your new job queue. For more information, see Example job definitions, Creating a single-node job definition, and Submitting a job.

Using a GPU workload AMI

To run GPU workloads on your AWS Batch compute resources, you must use an AMI with GPU support. For more information, see Working with GPUs on Amazon ECS and Amazon ECS-optimized AMIs in Amazon Elastic Container Service Developer Guide.

In managed compute environments, if the compute environment specifies any p2, p3, p4, p5, g3, g3s, g4, or g5 instance types or instance families, then AWS Batch uses an Amazon ECS GPU optimized AMI.

In unmanaged compute environments, an Amazon ECS GPU-optimized AMI is recommended. You can use the AWS Command Line Interface or AWS Systems Manager Parameter Store GetParameter, GetParameters, and GetParametersByPath operations to retrieve the metadata for the recommended Amazon ECS GPU-optimized AMIs.
Note

The p5 instance family is only supported on versions equal or later than 20230912 of the Amazon ECS GPU-optimized AMI and they are incompatible with p2 and g2 instance types. If you need to use p5 instances, ensure that your compute environment doesn’t contain p2 or g2 instances and uses the latest default Batch AMI. Creating a new compute environment will use the latest AMI but if you are updating your compute environment to include p5, you can ensure you are using the latest AMI by setting `updateToLatestImageVersion` to true in `ComputeResource` properties. For more information on AMI compatibility with GPU instances, see [Working with GPUs on Amazon ECS](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/gpu-workload-ami.html) in *Amazon Elastic Container Service Developer Guide*.

The following examples show how to use the [GetParameter](https://docs.aws.amazon.com/cli/latest/reference/ssm/get-parameter.html) command.

AWS CLI

```bash
$ aws ssm get-parameter --name /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended --region us-east-2 --output json
```

The output includes the AMI information in the `Value` parameter.

```json
{
   "Parameter": {
      "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended",
      "LastModifiedDate": 1555434128.664,
      "Value": "{\"schema_version\":1,\"image_name\":\"amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs\",\"image_id\":\"ami-083c800fe4211192f\",\"os\":\"Amazon Linux 2\",\"ecs_runtime_version\":\"Docker version 18.06.1-ce\",\"ecs_agent_version\":\"1.27.0\"},
      "Type": "String",
      "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended"
   }
}
```
```python
from __future__ import print_function
import json
import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameter(Name='/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended')
jsonVal = json.loads(response['Parameter']['Value'])
print("image_id   = " + jsonVal['image_id'])
print("image_name = " + jsonVal['image_name'])
```

The output only includes the AMI ID and AMI name:

```
image_id   = ami-083c800fe4211192f
image_name = amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
```

The following examples demonstrate the use of `GetParameters`.

**AWS CLI**

```
$ aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name \
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id \
--region us-east-2 --output json
```

The output includes the full metadata for each of the parameters:

```
{
   "InvalidParameters": [],
   "Parameters": [
   {
      "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id",
      "LastModifiedDate": 1555434128.749,
      "Value": "ami-083c800fe4211192f",
```

Using a GPU workload AMI
from __future__ import print_function

import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameters(
    Names=['/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name',
           '/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id'])

for parameter in response['Parameters']:
    print(parameter['Name'] + " = " + parameter['Value'])

The output includes the AMI ID and AMI name, using the full path for the names.

/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id = ami-083c800fe4211192f
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name = amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs

The following examples show how to use the GetParametersByPath command.
AWS CLI

```
$ aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended \
   --region us-east-2 --output json
```

The output includes the full metadata for all of the parameters under the specified path.

```
{
   "Parameters": [
      {
         "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_agent_version",
         "LastModifiedDate": 1555434128.801,
         "Value": "1.27.0",
         "Version": 8,
         "Type": "String",
         "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_agent_version"
      },
      {
         "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_runtime_version",
         "LastModifiedDate": 1548368308.213,
         "Value": "Docker version 18.06.1-ce",
         "Version": 1,
         "Type": "String",
         "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_runtime_version"
      },
      {
         "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id",
         "LastModifiedDate": 1555434128.749,
         "Value": "ami-083c800fe4211192f",
         "Version": 9,
         "Type": "String",
         "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id"
      },
      {
         "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name",
```
Using a GPU workload AMI

```
from __future__ import print_function

import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameters_by_path(Path='/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended')
for parameter in response['Parameters']:
    print(parameter['Name'] + ' = ' + parameter['Value'])
```
The output includes the values of all the parameter names at the specified path, using the full path for the names.

/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_agent_version = 1.27.0
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_runtime_version = Docker version 18.06.1-ce
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id = ami-083c800fe4211192f
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name = amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/os = Amazon Linux 2
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/schema_version = 1

For more information, see [Retrieving Amazon ECS-Optimized AMI Metadata](#) in the *Amazon Elastic Container Service Developer Guide*.

### Amazon Linux deprecation

The Amazon Linux AMI (also called Amazon Linux 1) reached its end of life on December 31, 2023. AWS Batch has ended support for Amazon Linux AMI as it will not receive any security updates or bug fixes starting January 1, 2024. For more information about the Amazon Linux end-of-life, see [AL FAQ](#).

We recommend that you update existing Amazon Linux based compute environments to Amazon Linux 2023 to prevent unforeseen workload interruptions, and continue to receive security and other updates.

Your compute environments using the Amazon Linux AMI may continue functioning beyond the December 31, 2023 end-of-life date. However, these compute environments will no longer receive any new software updates, security patches, or bug fixes from AWS. It is your responsibility to maintain these compute environments on the Amazon Linux AMI after end-of-life. We recommend migrating AWS Batch compute environments to Amazon Linux 2023 or Amazon Linux 2 to maintain optimal performance and security.

For help migrating AWS Batch from the Amazon Linux AMI to Amazon Linux 2023 or Amazon Linux 2, see [Updating compute environments - AWS Batch](#).
Launch template support

AWS Batch supports using Amazon EC2 launch templates with your EC2 compute environments. With launch templates, you can modify the default configuration of your AWS Batch compute resources without needing to create customized AMIs.

Note
Launch templates aren't supported on AWS Fargate resources.

You must create a launch template before you can associate it with a compute environment. You can create a launch template in the Amazon EC2 console. Or, you can use the AWS CLI or an AWS SDK. For example, the following JSON file represents a launch template that resizes the Docker data volume for the default AWS Batch compute resource AMI and also sets it to be encrypted.

```json
{
   "LaunchTemplateName": "increase-container-volume-encrypt",
   "LaunchTemplateData": {
      "BlockDeviceMappings": [
         {
            "DeviceName": "/dev/xvda",
            "Ebs": {
               "Encrypted": true,
               "VolumeSize": 100,
               "VolumeType": "gp2"
            }
         }
      ]
   }
}
```

You can create the previous launch template by saving the JSON to a file that's called `lt-data.json` and running the following AWS CLI command.

```
aws ec2 --region <region> create-launch-template --cli-input-json file://lt-data.json
```

For more information about launch templates, see Launching an Instance from a Launch Template in the Amazon EC2 User Guide for Linux Instances.
If you use a launch template to create your compute environment, you can move the following existing compute environment parameters to your launch template:

- Amazon EC2 key pair
- Amazon EC2 AMI ID
- Security group IDs
- Amazon EC2 tags

The following launch template parameters are ignored by AWS Batch:

- Instance type (specify your desired instance types when you create your compute environment)
- Instance role (specify your desired instance role when you create your compute environment)
- Network interface subnets (specify your desired subnets when you create your compute environment)
- Instance market options (AWS Batch must control Spot Instance configuration)
- Disable API termination (AWS Batch must control instance lifecycle)

AWS Batch only updates the launch template with a new launch template version during infrastructure updates. For more information, see Updating compute environments.

Amazon EC2 user data in launch templates

You can supply Amazon EC2 user data in your launch template that’s run by cloud-init when your instances launch. Your user data can perform common configuration scenarios, including but not limited to the following:

- Including users or groups
• Installing packages

• Creating partitions and file systems

Amazon EC2 user data in launch templates must be in the **MIME multi-part archive** format. This is because your user data is merged with other AWS Batch user data that's required to configure your compute resources. You can combine multiple user data blocks together into a single MIME multi-part file. For example, you might want to combine a cloud boothook that configures the Docker daemon with a user data shell script that writes configuration information for the Amazon ECS container agent.

If you're using AWS CloudFormation, the **AWS::CloudFormation::Init** type can be used with the **cfn-init** helper script to perform common configuration scenarios.

A MIME multi-part file consists of the following components:

• The content type and part boundary declaration: Content-Type: multipart/mixed; boundary="==BOUNDARY=="

• The MIME version declaration: MIME-Version: 1.0

• One or more user data blocks that contain the following components:
  • The opening boundary that signals the beginning of a user data block: --==BOUNDARY==. You must keep the line before this boundary blank.
  • The content type declaration for the block: Content-Type: text/cloud-config; charset="us-ascii". For more information about content types, see the **Cloud-Init documentation**. You must keep the line after the content type declaration blank.
  • The content of the user data, such as a list of shell commands or cloud-init directives.
  • The closing boundary that signals the end of the MIME multi-part file: --==BOUNDARY==--. You must keep the line before the closing boundary blank.

The following are example MIME multi-part files that you can use to create your own.

⚠️ **Note**

If you add user data to a launch template in the Amazon EC2 console, you can paste it in as plaintext. Or, you can upload it from a file. If you use the AWS CLI or an AWS SDK, you must first base64 encode the user data and submit that string as the value of the **UserData** parameter when you call **CreateLaunchTemplate**, as shown in this JSON file.
Amazon EC2 user data in launch templates

Examples

- **Example: Mount an existing Amazon EFS file system**
- **Example: Override default Amazon ECS container agent configuration**
- **Example: Mount an existing Amazon FSx for Lustre file system**

**Example: Mount an existing Amazon EFS file system**

**Example**

This example MIME multi-part file configures the compute resource to install the `amazon-efs-utils` package and mount an existing Amazon EFS file system at /mnt/efs.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- amazon-efs-utils

runcmd:
  - file_system_id_01=fs-abcdef123
  - efs_directory=/mnt/efs
  - mkdir -p ${efs_directory}
  - echo "${file_system_id_01}:/ ${efs_directory} efs tls,_netdev" >> /etc/fstab
  - mount -a -t efs defaults

--==MYBOUNDARY==--
Example: Override default Amazon ECS container agent configuration

Example

This example MIME multi-part file overrides the default Docker image cleanup settings for a compute resource.

```bash
# !/bin/bash
echo ECS_IMAGE_CLEANUP_INTERVAL=60m >> /etc/ecs/ecs.config
echo ECS_IMAGE_MINIMUM_CLEANUP_AGE=60m >> /etc/ecs/ecs.config
```

Example: Mount an existing Amazon FSx for Lustre file system

Example

This example MIME multi-part file configures the compute resource to install the `lustre2.10` package from the Extras Library and mount an existing FSx for Lustre file system at `/scratch` and a mount name of `fsx`. This example is for Amazon Linux 2. For installation instructions for other Linux distributions, see Installing the Lustre Client in the Amazon FSx for Lustre User Guide. For more information, see Mounting your Amazon FSx file system automatically in the Amazon FSx for Lustre User Guide.

```bash
runcmd:
- file_system_id_01=fs-0abcdef1234567890
- region=us-east-2
- fsx_directory=/scratch
- amazon-linux-extras install -y lustre2.10
```
In the `volumes` and `mountPoints` members of the container properties the mount points must be mapped into the container.

```json
{
    "volumes": [
        {
            "host": {
                "sourcePath": "/scratch"
            },
            "name": "Scratch"
        }
    ],
    "mountPoints": [
        {
            "containerPath": "/scratch",
            "sourceVolume": "Scratch"
        }
    ]
}
```

**Creating a compute environment**

Before you can run jobs in AWS Batch, you need to create a compute environment. You can create a managed compute environment where AWS Batch manages the Amazon EC2 instances or AWS Fargate resources within the environment based on your specifications. Or, alternatively, you can create an unmanaged compute environment where you handle the Amazon EC2 instance configuration within the environment.

⚠️ **Important**

Fargate Spot instances are not supported in the following scenarios:

- On Amazon Linux containers with ARM64 architecture.
- Windows containers on AWS Fargate
A job queue will be blocked in these scenarios if a job is submitted to a job queue that only uses Fargate Spot compute environments.

Contents

- To create a managed compute environment using AWS Fargate resources
- To create a managed compute environment using EC2 resources
- To create an unmanaged compute environment using EC2 resources
- To create a managed compute environment using Amazon EKS resources

To create a managed compute environment using AWS Fargate resources

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Compute environments.
4. Choose Create.
5. Configure the compute environment.

Note

Compute environments for Windows containers on AWS Fargate jobs must at least one vCPU.

a. For Compute environment configuration, choose Fargate.

b. For Name, specify a unique name for your compute environment. The name can contain up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

c. For Service role, choose service-linked role that lets the AWS Batch service to make calls to the required AWS API operations on your behalf. For example, choose AWSServiceRoleForBatch. For more information, see Service-linked role permissions for AWS Batch.
d. (Optional) Expand **Tags**. To add a tag, choose **Add tag**. Then, enter a **Key** name and optional **Value**. Choose **Add tag**.

e. Choose **Next page**.

6. **In the Instance configuration section:**

   a. (Optional) For **Use Fargate Spot capacity**, turn on Fargate Spot. For information about Fargate Spot, see *Using Amazon EC2 Spot and Fargate_SPOT*.

   b. For **Maximum vCPUs**, choose the maximum number of vCPUs that your compute environment can scale out to, regardless of job queue demand.

   c. Choose **Next page**.

7. **Configure networking.**

   ![](Important.png)

   **Important**

   Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses.

   For more information about interface VPC endpoints, see *Amazon ECS Interface VPC Endpoints (AWS PrivateLink)* in the *Amazon Elastic Container Service Developer Guide*. If you do not have an interface VPC endpoint configured and your compute resources 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*. For more information, see the section called “Create a VPC”.

   a. For **Virtual Private Cloud (VPC) ID**, choose a VPC where you want to launch your instances.

   b. For **Subnets**, choose the subnets to use. By default, all subnets within the selected VPC are available.

   ![](Note.png)

   **Note**

   AWS Batch on Fargate doesn’t currently support Local Zones. For more information, see *Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts* in the *Amazon Elastic Container Service Developer Guide*.
c. For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is chosen.

d. Choose **Next page**.

8. For **Review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create compute environment**.

**To create a managed compute environment using EC2 resources**


2. From the navigation bar, select the AWS Region to use.

3. In the navigation pane, choose **Compute environments**.

4. Choose **Create**.

5. Configure the environment.

   a. For **Compute environment configuration**, choose **Amazon Elastic Compute Cloud (Amazon EC2)**.

   b. For **Orchestration type**, choose **Managed**.

   c. For **Name**, specify a unique name for your compute environment. The name can contain up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

   d. (Optional) For **Service role**, choose service-linked role that lets the AWS Batch service make calls to the required AWS API operations on your behalf. For example, choose **AWSServiceRoleForBatch**. For more information, see [Service-linked role permissions for AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/servicesrolepermissions.html).

   e. For **Instance role**, choose to create a new instance profile or use an existing instance profile that has the required IAM permissions attached. This instance profile allows the Amazon ECS container instances that are created for your compute environment to make calls to the required AWS API operations on your behalf. For more information, see [Amazon ECS instance role](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/instance-roles.html). If you choose to create a new instance profile, the required role (**ecsInstanceRole**) is created for you.

   f. (Optional) Expand **Tags**.

   g. (Optional) For **EC2 tags**, choose **Add tag** to add a tag to resources that are launched in the compute environment. Then, enter a **Key** name and optional **Value**. Choose **Add tag**.
h. (Optional) For Tags, choose Add tag. Then, enter a Key name and optional Value. Choose Add tag.

For more information, see Tagging your AWS Batch resources.

i. Choose Next page.

6. In the Instance configuration section:

a. (Optional) For Enable using Spot instances, turn on Spot. For more information, see Spot Instances.

b. (Spot only) For Maximum % on-demand price, choose the maximum percentage that a Spot Instance price can be when compared with the On-Demand price for that instance type before instances are launched. For example, if your maximum price is 20%, then the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more than your maximum percentage. If you leave this field empty, the default value is 100% of the On-Demand price.

c. (Spot only) For Spot fleet role, choose an existing Amazon EC2 Spot Fleet IAM role to apply to your Spot compute environment. If you don't already have an existing Amazon EC2 Spot Fleet IAM role, you must create one first. For more information, see Amazon EC2 spot fleet role.

! Important

To tag your Spot Instances on creation, your Amazon EC2 Spot Fleet IAM role must use the newer AmazonEC2SpotFleetTaggingRole managed policy. The AmazonEC2SpotFleetRole managed policy doesn't have the required permissions to tag Spot Instances. For more information, see Spot Instances not tagged on creation and the section called “Tagging your resources”.

d. For Minimum vCPUs, choose the minimum number of vCPUs that your compute environment maintains, regardless of job queue demand.

e. For Desired vCPUs, choose the number of vCPUs that your compute environment launches with. As your job queue demand increases, AWS Batch can increase the desired number of vCPUs in your compute environment and add EC2 instances, up to the maximum vCPUs. As demand decreases, AWS Batch can decrease the desired number of vCPUs in your compute environment and remove instances, down to the minimum vCPUs.
f. For **Maximum vCPUs**, choose the maximum number of vCPUs that your compute environment can scale out to, regardless of job queue demand.

g. For **Allowed instance types**, choose the Amazon EC2 instance types that can be launched. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3). Or, you can specify specific sizes within a family (such as c5.8xlarge). Metal instance types aren’t in the instance families. For example, c5 doesn’t include c5.metal. You can also choose `optimal` to select instance types (from the C4, M4, and R4 instance families) that match the demand of your job queues.

**Note**

When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can’t mix x86 and ARM instances in the same compute environment.

**Note**

AWS Batch will scale GPUs based on the required amount in your job queues. To use GPU scheduling, the compute environment must include instance types from the p2, p3, p4, p5, g3, g3s, g4, or g5 families.

**Note**

Currently, `optimal` uses instance types from the C4, M4, and R4 instance families. In AWS Regions that don’t have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

h. Expand **Additional configuration**.

i. (Optional) For **Placement group**, enter a placement group name to group resources in the compute environment.

j. (Optional) For **EC2 key pair**, choose a public and private key pair as security credentials when you connect to the instance. For more information about Amazon EC2 key pairs, see [Amazon EC2 key pairs and Linux instances](#).
k. For **Allocation strategy**, choose the allocation strategy to use when selecting instance types from the list of allowed instance types. **BEST_FIT_PROGRESSIVE** is usually the better choice for EC2 On-Demand compute environments, **SPOT_CAPACITY_OPTIMIZED**, and **SPOT_PRICE_CAPACITY_OPTIMIZED** for EC2 Spot compute environments. For more information, see the section called “Allocation strategies”.

l. (Optional) For **EC2 configuration** choose **Image type** and **Image ID override** values to provide information for AWS Batch to select Amazon Machine Images (AMIs) for instances in the compute environment. If the **Image ID override** isn't specified for each **Image type**, AWS Batch selects a recent **Amazon ECS optimized AMI**. If no **Image type** is specified, the default is a **Amazon Linux 2** for non-GPU, non AWS Graviton instance.

⚠️ **Important**

To use a custom AMI, choose the image type and then enter the custom AMI ID in the **Image ID override** box.

**Amazon Linux 2**

Default for all AWS Graviton-based instance families (for example, C6g, M6g, R6g, and T4g) and can be used for all non-GPU instance types.

**Amazon Linux 2 (GPU)**

Default for all GPU instance families (for example P4 and G4) and can be used for all non AWS Graviton-based instance types.

**Amazon Linux**

Can be used for non-GPU, non AWS Graviton instance families. The standard support for Amazon Linux AMI has ended. For more information, see **Amazon Linux AMI**.

⚠️ **Note**

The AMI that you choose for a compute environment must match the architecture of the instance types that you want to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support Arm instances. Amazon ECS vends both x86 and Arm versions of the Amazon ECS optimized Amazon Linux 2 AMI. For
m. (Optional) For **Launch template**, select an existing Amazon EC2 launch template to configure your compute resources. The default version of the template is automatically populated. For more information, see **Launch template support**.

   ![Note]
   
   In a launch template, you can specify a custom AMI that you created.

n. (Optional) For **Launch template version**, enter $Default, $Latest, or a specific version number to use.

   ![Important]
   
   If the version parameter of the launch template is $Default or $Latest, the default or latest version of the specified launch template is evaluated during an infrastructure update. If a different AMI ID is selected by the default or the latest version of the launch template is selected, that AMI ID is used in the update. For more information, see the section called “Updating the AMI ID”.

do. Choose **Next page**.

7. In the **Network configuration** section:

   ![Important]
   
   Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses.

   For more information about interface VPC endpoints, see Amazon ECS Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide. If you do not have an interface VPC endpoint configured and your compute resources 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. For more information, see the section called “Create a VPC”.
a. For **Virtual Private Cloud (VPC) ID**, choose a VPC where to launch your instances.
b. For **Subnets**, choose the subnets to use. By default, all subnets within the selected VPC are available.

c. (Optional) For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is chosen.

8. Choose **Next page**.
9. For **Review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create compute environment**.

**To create an unmanaged compute environment using EC2 resources**

2. From the navigation bar, select the AWS Region to use.
3. On the **Compute Environments** page, choose **Create**.
4. Configure the environment.
   a. For **Compute environment configuration**, choose **Amazon Elastic Compute Cloud (Amazon EC2)**.
   b. For **Orchestration type**, choose **Unmanaged**.
5. For **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
6. (Optional) For **Service role**, choose a role that lets the AWS Batch service to make calls to the required AWS API operations on your behalf. For example, choose **AWSBatchServiceRole**. For more information, see the section called “Using Service-Linked Roles”..
7. For **Maximum vCPUs**, choose the maximum number of vCPUs that your compute environment can scale out to, regardless of job queue demand.

8. (Optional) Expand **Tags**. To add a tag, choose **Add tag**. Then, enter a **Key** name and optional **Value**. Choose **Add tag**. For more information, see [Tagging your AWS Batch resources](#).

9. Choose **Next page**.

10. For **Review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create compute environment**.

### To create a managed compute environment using Amazon EKS resources


2. From the navigation bar, select the AWS Region to use.

3. In the navigation pane, choose **Compute environments**.

4. Choose **Create**.

5. For **Compute environment configuration**, choose **Amazon Elastic Kubernetes Service (Amazon EKS)**.

6. For **Name**, specify a unique name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

7. For **Instance role**, choose an existing instance profile that has the required IAM permissions attached.

   **Note**

   To create a compute environment in the AWS Batch console, choose an instance profile that has the `eks:ListClusters` and `eks:DescribeCluster` permissions.

8. For **EKS cluster**, choose an existing Amazon EKS cluster.

9. For **Namespace**, enter a Kubernetes namespace to group your AWS Batch processes in the cluster.

10. (Optional) Expand **Tags**. Choose **Add tag** and then enter a key-value pair.

11. Choose **Next page**.
12. (Optional) For **Use EC2 Spot Instances**, turn on **Enable using Spot instances** to use Amazon EC2 Spot Instances.

13. (Spot only) For **Maximum % on-demand price**, choose the maximum percentage that a Spot Instance price can be when compared with the On-Demand price for that instance type before instances are launched. For example, if your maximum price is 20%, then the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more than your maximum percentage. If you leave this field empty, the default value is 100% of the On-Demand price.

14. (Spot only) For **Spot fleet role**, choose the Amazon EC2 Spot fleet IAM role for the SPOT compute environment.

⚠️ **Important**

This role is required if the allocation strategy is set to **BEST_FIT** or not specified.

15. (Optional) For **Minimum vCPUs**, choose the minimum number of vCPUs that your compute environment maintains, regardless of job queue demand.

16. (Optional) For **Maximum vCPUs**, choose the maximum number of vCPUs that your compute environment can scale out to, regardless of job queue demand.

17. For **Allowed instance types**, choose the Amazon EC2 instance types that can be launched. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3). Or, you can specify specific sizes within a family (for example, c5.8xlarge). Metal instance types aren't in the instance families. For example, c5 doesn't include c5.metal. You can also choose **optimal** to select instance types (from the C4, M4, and R4 instance families) because you need that match the demand of your job queues.

⚠️ **Note**

When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.
AWS Batch scales GPUs based on the required amount in your job queues. To use GPU scheduling, the compute environment must include instance types from the p2, p3, p4, p5, g3, g3s, g4, or g5 families.

Note

Currently, optimal uses instance types from the C4, M4, and R4 instance families. In AWS Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

18. (Optional) Expand **Additional configuration**.

   a. (Optional) For **Placement group**, enter a placement group name to group resources in the compute environment.

   b. For **Allocation strategy**, choose **BEST_FIT_PROGRESSIVE**.

   c. (Optional) For **Amazon Machine Images (AMIs) Configuration**, choose **Add amazon machine images (amis) configuration**. Then, choose an **Image Type**, enter an **Image ID override**, and **Kubernetes version**.

   **Important**

   To use a custom AMI, choose the image type and then enter the custom AMI ID in the **Image ID override** box.

   **Note**

   If the **Image ID override** isn't specified for each **Image type**, AWS Batch selects a recent **Amazon ECS optimized AMI**. If no **Image type** is specified, the default is a **Amazon Linux 2** for non-GPU, non AWS Graviton instance.
d. (Optional) For **Launch template**, choose an existing launch template.

e. (Optional) For **Launch template version**, enter $\text{Default}$, $\text{Latest}$, or a version number.

19. Choose **Next page**.

20. For **Virtual Private Cloud (VPC) ID**, choose a VPC where to launch the instances.

21. For **Subnets**, choose the subnets to use. By default, all subnets within the selected VPC are available.

{} **Note**

AWS Batch on Amazon EKS supports Local Zones. For more information, see [Amazon EKS and AWS Local Zones](#) in the *Amazon EKS User Guide*.

22. (Optional) For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is selected.

23. Choose **Next page**.

24. For **Review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create compute environment**.

### Compute environment template

The following example shows an empty compute environment template. You can use this template to create your compute environment that can then be saved to a file and used with the AWS CLI `--cli-input-json` option. For more information about these parameters, see [CreateComputeEnvironment](#) in the *AWS Batch API Reference*.

```
{
    "computeEnvironmentName": "",
```
"type": "UNMANAGED",
"state": "DISABLED",
"unmanagedvCpus": 0,
"computeResources": {
  "type": "EC2",
  "allocationStrategy": "BEST_FIT_PROGRESSIVE",
  "minvCpus": 0,
  "maxvCpus": 0,
  "desiredvCpus": 0,
  "instanceTypes": [ ""
  ],
  "imageId": "",
  "subnets": [
  ""
  ],
  "securityGroupIds": [ ""
  ],
  "ec2KeyPair": "",
  "instanceRole": "",
  "tags": {
    "KeyName": ""
  },
  "placementGroup": "",
  "bidPercentage": 0,
  "spotIamFleetRole": "",
  "launchTemplate": {
    "launchTemplateId": "",
    "launchTemplateName": "",
    "version": ""
  },
  "ec2Configuration": [
    {
      "imageType": "",
      "imageIdOverride": "",
      "imageKubernetesVersion": ""
    }
  ],
  "serviceRole": "",
  "tags": {
    "KeyName": ""
  }
},
"eksConfiguration": {
  "eksClusterArn": "",
  "kubernetesNamespace": ""
}
}

### Note

You can generate the preceding compute environment template with the following AWS CLI command.

```bash
$ aws batch create-compute-environment --generate-cli-skeleton
```

---

## Compute environment parameters

Compute environments are split into several basic components: the name, type, and state of the compute environment, the compute resource definition (if it's a managed compute environment), the Amazon EKS configuration (if it uses Amazon EKS resources), the service role to use to provide IAM permissions to AWS Batch, and the tags for the compute environment.

### Topics

- Compute environment name
- Type
- State
- Compute resources
- Amazon EKS configuration
- Service role
- Tags

### Compute environment name

**computeEnvironmentName**

The name for your compute environment. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
**Type**

type

The type of the compute environment. Choose MANAGED to have AWS Batch manage the EC2 or Fargate compute resources that you define. For more information, see [Compute resources](#). Choose UNMANAGED to manage your own EC2 compute resources.

Type: String

Valid values: MANAGED | UNMANAGED

Required: Yes

**State**

state

The state of the compute environment.

If the state is ENABLED, the AWS Batch scheduler attempts to place jobs within the environment. These jobs are from an associated job queue on the compute resources. If the compute environment is managed, the instances scale out or in automatically based on job queue demand.

If the state is DISABLED, the AWS Batch scheduler doesn't attempt to place jobs within the environment. Jobs that are in a STARTING or RUNNING state continue to progress normally. Managed compute environments that are in the DISABLED state don't scale out.

---

**Note**

Compute environments in a DISABLED state might continue to incur billing charges. To prevent additional charges, turn off and then delete the compute environment. For more information, see [DeleteComputeEnvironment](#) in the AWS Batch API Reference and [Avoiding unexpected charges](#) in the [AWS Billing User Guide](#).
When an instance is idle, the instance scales down to the `minvCpus` value. However, the instance size doesn't change. For example, consider a `c5.8xlarge` instance with a `minvCpus` value of 4 and a `desiredvCpus` value of 36. This instance doesn't scale down to a `c5.large` instance.

Type: String

Valid values: ENABLED | DISABLED

Required: No

**Compute resources**

computeResources

Details of the compute resources managed by the compute environment. For more information, see [Compute environment](#).

Type: `ComputeResource` object

Required: This parameter is required for managed compute environments

type

The type of compute environment. You can choose either to use EC2 On-Demand Instances (EC2) and EC2 Spot Instances (SPOT), or to use Fargate capacity (FARGATE) and Fargate Spot capacity (FARGATE_SPOT) in your managed compute environment. If you choose SPOT, you must also specify an Amazon EC2 Spot Fleet role with the `spotIamFleetRole` parameter. For more information, see [Amazon EC2 spot fleet role](#).

Valid values: EC2 | SPOT | FARGATE | FARGATE_SPOT

Required: Yes

allocationStrategy

The allocation strategy to use for the compute resource if not enough instances of the best fitting EC2 instance type can be allocated. This might be because of availability of the instance type in the AWS Region or [Amazon EC2 service limits](#). For more information, see [Allocation strategies](#).
Note
This parameter isn't applicable to jobs that run on Fargate resources.

BEST_FIT (default)

AWS Batch selects an instance type that best fits the needs of the jobs with a preference for the lowest cost instance type. If additional instances of the selected instance type aren't available, AWS Batch waits for the additional instances to be available. If there aren't enough instances available, or if you're hitting Amazon EC2 service limits, additional jobs don't run until after currently running jobs are complete. This allocation strategy keeps costs lower but can limit scaling. If you're using Spot Fleets with BEST_FIT, the Spot Fleet IAM Role must be specified. Compute resources that use a BEST_FIT allocation strategy don't support infrastructure updates and can't update some parameters. For more information, see Updating compute environments.

Note
BEST_FIT isn't supported for compute environments using Amazon EKS resources.

BEST_FIT_PROGRESSIVE

Use additional instance types that are large enough to meet the requirements of the jobs in the queue. Prefer instance types with a lower cost for each unit vCPU. If additional instances of the previously selected instance types aren't available, AWS Batch selects new instance types.

SPOT_CAPACITY_OPTIMIZED

(Only available for Spot Instance compute resources) Use additional instance types that are large enough to meet the requirements of the jobs in the queue. Prefer instance types that are less likely to be interrupted.

SPOT_PRICE_CAPACITY_OPTIMIZED

(Only available for Spot Instance compute resources) The price and capacity optimized allocation strategy looks at both price and capacity to select the Spot Instance pools that are the least likely to be interrupted and have the lowest possible price.
With BEST_FIT_PROGRESSIVE, SPOT_CAPACITY_OPTIMIZED, and SPOT_PRICE_CAPACITY_OPTIMIZED allocation strategies using On-Demand or Spot Instances and the BEST_FIT strategy using Spot Instances, AWS Batch might need to exceed $maxvCpus$ to meet your capacity requirements. In this event, AWS Batch never exceeds $maxvCpus$ by more than a single instance.

Valid values: BEST_FIT | BEST_FIT_PROGRESSIVE | SPOT_CAPACITY_OPTIMIZED | SPOT_PRICE_CAPACITY_OPTIMIZED

Required: No

$minvCpus$

The minimum number of vCPUs that an environment maintains even if a compute environment is DISABLED.

Note

This parameter isn't applicable to jobs that run on Fargate resources.

Type: Integer

Required: No

$maxvCpus$

The maximum number of vCPUs that the AWS Batch compute environment can support.

Note

With BEST_FIT_PROGRESSIVE, SPOT_CAPACITY_OPTIMIZED, and SPOT_PRICE_CAPACITY_OPTIMIZED allocation strategies using On-Demand or Spot Instances and the BEST_FIT strategy using Spot Instances, AWS Batch might...
need to exceed `maxvCpus` to meet your capacity requirements. In this event, AWS Batch never exceeds `maxvCpus` by more than a single instance. For example, AWS Batch uses no more than a single instance from among those specified in your compute environment.

**Type:** Integer  
**Required:** No

### desiredvCpus

The desired number of vCPUS in the compute environment. AWS Batch modifies this value between the minimum and maximum values based on job queue demand.

**Note**  
This parameter isn't applicable to jobs that run on Fargate resources.

**Type:** Integer  
**Required:** No

### instanceTypes

The instance types that can be launched. This parameter isn't applicable to jobs that run on Fargate resources. Do not specify it. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3). Or, you can specify specific sizes within a family (such as `c5.8xlarge`). Note that metal instance types aren't in the instance families (for example `c5` does not include `c5.metal`). You can also choose `optimal` to select instance types (from the C4, M4, and R4 instance families) that match the demand of your job queues.

**Note**  
When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.
Currently, optimal uses instance types from the C4, M4, and R4 instance families. In AWS Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

Type: Array of strings

Required: yes

imageId

*This parameter is deprecated.*

The Amazon Machine Image (AMI) ID used for instances launched in the compute environment. This parameter is overridden by the `imageIdOverride` member of the `Ec2Configuration` structure.

This parameter isn't applicable to jobs that run on Fargate resources.

The AMI that you choose for a compute environment must match the architecture of the instance types that you want to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support Arm instances. Amazon ECS vends both x86 and Arm versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see [Amazon ECS optimized Amazon Linux 2 AMI](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/amazon-ecs-optimized-amazon-linux-2-ami.html) in the *Amazon Elastic Container Service Developer Guide*.

Type: String

Required: No
subnets

The VPC subnets into which the compute resources are launched. These subnets must be within the same VPC. Fargate compute resources can contain a maximum of 16 subnets. For more information, see VPCs and Subnets in the Amazon VPC User Guide.

Note

AWS Batch on Amazon EC2 and AWS Batch on Amazon EKS support Local Zones. For more information, see Local Zones in the Amazon EC2 User Guide for Linux Instances, Amazon EKS and AWS Local Zones in the Amazon EKS User Guide and Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts in the Amazon Elastic Container Service Developer Guide.

AWS Batch on Fargate doesn't currently support Local Zones.

When updating compute environments, if you provide an empty list of VPC subnets, the resulting behavior differs between Fargate and EC2 compute resources. For Fargate compute resources, providing an empty list is handled as if this parameter wasn't specified and no change is made. For EC2 compute resources, providing an empty list removes the VPC subnets from the compute resource. If you change the VPC subnets, an infrastructure update of the compute environment is required. This is the case for both Fargate and EC2 compute resources. For more information, see Updating compute environments.

Type: Array of strings

Required: Yes

securityGroupIds

The Amazon EC2 security groups associated with instances launched in the compute environment. One or more security groups must be specified, either in securityGroupIds or using a launch template referenced in launchTemplate. This parameter is required for jobs that run on Fargate resources and must contain at least one security group. (Fargate doesn't support launch templates.) If security groups are specified using both securityGroupIds and launchTemplate, the values in securityGroupIds will be used.

When updating compute environments, if you provide an empty list of security groups, the resulting behavior differs between Fargate and EC2 compute resources. For Fargate compute
resources, providing an empty list is handled as if this parameter wasn't specified and no change is made. For EC2 compute resources, providing an empty list removes the security groups from the compute resource. If you change the security groups, an infrastructure update of the compute environment is required. This is the case for both Fargate and EC2 compute resources. For more information, see Updating compute environments.

Type: Array of strings

Required: Yes

**ec2KeyPair**

The EC2 key pair that's used for instances launched in the compute environment. You can use this key pair to log in to your instances with SSH. When updating a compute environment, if you change the EC2 key pair, an infrastructure update of the compute environment is required. For more information, see Updating compute environments.

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.

Type: String

Required: No

**instanceRole**

The Amazon ECS instance profile to attach to Amazon EC2 instances in a compute environment. This parameter isn't applicable to jobs that run on Fargate resources. Do not specify it. You can specify the short name or full Amazon Resource Name (ARN) of an instance profile. For example, ecsInstanceRole or arn:aws:iam::aws_account_id:instance-profile/ecsInstanceRole. For more information, see Amazon ECS instance role.

When updating a compute environment, if you change this setting, an infrastructure update of the compute environment is required. For more information, see Updating compute environments.

Type: String

Required: No
tags

Key-value pair tags to be applied to EC2 instances that are launched in the compute environment. For example, you can specify "Name": "AWS Batch Instance - C4OnDemand" as a tag so that each instance in your compute environment has that name. This is helpful for recognizing your AWS Batch instances in the Amazon EC2 console. These tags aren't seen when using the AWS Batch ListTagsForResource API operation.

When updating a compute environment, if you change the EC2 tags, an infrastructure update of the compute environment is required. For more information, see Updating compute environments.

Note

This parameter isn't applicable to jobs that run on Fargate resources.

Type: String to string map

Required: No

placementGroup

The Amazon EC2 placement group to associate with your compute resources. This parameter isn't applicable to jobs that run on Fargate resources. Do not specify it. If you intend to submit multi-node parallel jobs to your compute environment, consider creating a cluster placement group and associate it with your compute resources. This keeps your multi-node parallel job on a logical grouping of instances within a single Availability Zone with high network flow potential. For more information, see Placement Groups in the Amazon EC2 User Guide for Linux Instances.

Note

This parameter isn't applicable to jobs that run on Fargate resources.

Type: String

Required: No
**bidPercentage**

The maximum percentage that an EC2 Spot Instance price can be when compared with the On-Demand price for that instance type before instances are launched. For example, if your maximum percentage is 20%, the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more than your maximum percentage. If you leave this field empty, the default value is 100% of the On-Demand price. For most use cases, we recommend leaving this field empty.

When updating a compute environment, if you change the bid percentage, an infrastructure update of the compute environment is required. For more information, see [Updating compute environments](#).

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.

**Required:** No

**spotIamFleetRole**

The Amazon Resource Name (ARN) of the Amazon EC2 Spot Fleet IAM role applied to a SPOT compute environment. This role is required if the allocation strategy set to BEST_FIT or if the allocation strategy isn't specified. For more information, see [Amazon EC2 spot fleet role](#).

**Note**

This parameter isn't applicable to jobs that run on Fargate resources.

**Important**

To tag your Spot Instances on creation, the Spot Fleet IAM role specified here must use the newer `AmazonEC2SpotFleetTaggingRole` managed policy. The previously recommended `AmazonEC2SpotFleetRole` managed policy doesn't have the required permissions to tag Spot Instances. For more information, see [Spot Instances not tagged on creation](#).
launchTemplate

An optional launch template to associate with your compute resources. This parameter isn't applicable to jobs that run on Fargate resources. Do not specify it. Any other compute resource parameters that you specify in a CreateComputeEnvironment or UpdateComputeEnvironment API operation override the same parameters in the launch template. To use a launch template, you must specify either the launch template ID or launch template name in the request, but not both. For more information, see Launch template support.

When updating a compute environment, to remove the custom launch template and use the default launch template, set the launchTemplateId or launchTemplateName member of the launch template specification to an empty string. Removing the launch template from a compute environment doesn't remove the AMI specified in the launch template, if it was the one that was used. To update the AMI selected from a launch template, the updateToLatestImageVersion parameter must be set to true. When updating a compute environment, if you change the launch template, an infrastructure update of the compute environment is required. For more information, see Updating compute environments.

Type: LaunchTemplateSpecification

object

Required: No

launchTemplateId

The ID of the launch template.

Type: String

Required: No

launchTemplateName

The name of the launch template.

Type: String
version

The version number of the launch template, $Latest$, or $Default$.

If the value is $Latest$, the latest version of the launch template is used. If the value is $Default$, the default version of the launch template is used. During an infrastructure update, if either $Latest$ or $Default$ was specified for the compute environment, AWS Batch re-evaluates the launch template version and might use a different version of the launch template. This is even if the launch template wasn't specified in the update.

Default: $Default$.

Type: String

Required: No

e2Configuration

Provides information used to select Amazon Machine Images (AMIs) for instances in the EC2 compute environment. If E2Configuration isn't specified, the default is Amazon Linux 2 (ECS_AL2). Before March 31, 2021, this default was Amazon Linux (ECS_AL1) for non-GPU, non AWS Graviton instances.

When updating a compute environment, if you change this parameter, an infrastructure update of the compute environment is required. For more information, see Updating compute environments.

Note

This parameter isn't applicable to jobs that run on Fargate resources.

Type: Array of E2Configuration objects

Required: No

imageIdOverride

The AMI ID that's used for instances launched in the compute environment that matches the image type. This setting overrides the imageId set in the computeResource object.
Type: String

Required: No

imageKubernetesVersion

The Kubernetes version for the compute environment. If you don't specify a value, the latest version that AWS Batch supports is used.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

imageType

The image type to match with the instance type to select an AMI. The supported values are different for ECS and EKS resources.

ECS

If the imageIdOverride parameter isn't specified, then a recent Amazon ECS-optimized Amazon Linux 2 AMI (ECS_AL2) is used. If a new image type is specified in an update, but neither an imageId nor a imageIdOverride parameter is specified, then the latest Amazon ECS optimized AMI for that image type that's supported by AWS Batch is used.

ECS_AL2

Amazon Linux 2: Default for all non-GPU instance families.

ECS_AL2_NVIDIA

Amazon Linux 2 (GPU): Default for all GPU instance families (for example P4 and G4) and can be used for all non AWS Graviton-based instance types.

ECS_AL1

Amazon Linux. Amazon Linux has reached the end-of-life of standard support. For more information, see Amazon Linux AMI.

EKS

If the imageIdOverride parameter isn't specified, then a recent Amazon EKS-optimized Amazon Linux AMI (EKS_AL2) is used. If a new image type is specified in an update, but neither an imageId nor a imageIdOverride parameter is specified,
then the latest Amazon EKS optimized AMI for that image type that AWS Batch supports is used.

**EKS_AL2**

*Amazon Linux 2:* Default for all non-GPU instance families.

**EKS_AL2_NVIDIA**

*Amazon Linux 2 (accelerated):* Default for all GPU instance families (for example, P4 and G4) and can be used for all non AWS Graviton-based instance types.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: Yes

---

**Amazon EKS configuration**

Configuration for the Amazon EKS cluster that supports the AWS Batch compute environment. The cluster must exist before the compute environment can be created.

**eksClusterArn**

The Amazon Resource Name (ARN) of the Amazon EKS cluster. An example is `arn:aws:eks:us-east-1:123456789012:cluster/ClusterForBatch`.

Type: String

Required: Yes

**kubernetesNamespace**

The namespace of the Amazon EKS cluster. AWS Batch manages pods in this namespace. The value can't be empty or null. It must be fewer than 64 characters long, cannot be set to `default`, cannot start with "kube-", and must match this regular expression: `^[a-z0-9]([-a-z0-9][a-z0-9]{0,62})?\$`. For more information, see **Namespaces** in the Kubernetes documentation.

Type: String

Required: Yes
Type: **EksConfiguration** Object

Required: No

**Service role**

**serviceRole**

The full Amazon Resource Name (ARN) of the IAM role that allows AWS Batch to make calls to other AWS services on your behalf. For more information, see [Using service-linked roles for AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/service-linked-roles.html). We recommend that you don't specify the service role. This way, AWS Batch uses the **AWSServiceRoleForBatch** service-linked role.

⚠️ **Important**

If your account has already created the AWS Batch service-linked role (**AWSServiceRoleForBatch**), that role is used by default for your compute environment unless you specify a role here. If the AWS Batch service-linked role doesn't exist in your account, and no role is specified here, the service tries to create the AWS Batch service-linked role in your account. For more information about the **AWSServiceRoleForBatch** service-linked role, see [Service-linked role permissions for AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/service-linked-roles.html).

If the compute environment is created using the **AWSServiceRoleForBatch** service-linked role, it can't be changed to use a regular IAM role. Likewise, if the compute environment is created with a regular IAM role, it can't be changed to use the **AWSServiceRoleForBatch** service-linked role. To update the parameters for the compute environment that require an infrastructure update to change, the **AWSServiceRoleForBatch** service-linked role must be used. For more information, see [Updating compute environments](https://docs.aws.amazon.com/batch/latest/userguide/creating-youropen.html).

If your specified role has a path other than `/`, make sure either to specify the full role ARN (recommended) or prefix the role name with the path.

⚠️ **Note**

Depending on how you created your AWS Batch service role, its Amazon Resource Name (ARN) might contain the `service-role` path prefix. When you only specify the name of the service role, AWS Batch assumes that your ARN doesn't use the `service-role`
path prefix. Because of this, we recommend that you specify the full ARN of your service role when you create compute environments.

**Tags**

tags

Key-value pair tags to associate with the compute environment. For more information, see [Tagging your AWS Batch resources](#).

Type: String to string map

Required: No

**EC2 Configurations**

AWS Batch uses Amazon ECS optimized AMIs for EC2 and EC2 Spot compute environments. The default is [Amazon Linux 2](#) (ECS_AL2). Before March 31, 2021, this default was [Amazon Linux](#) (ECS_AL1) for non-GPU, non AWS Graviton instances.

**Note**

AWS Batch also supports Amazon Linux 2023.

The Amazon Linux AMI (also called Amazon Linux 1) reached its end of life on December 31, 2023. AWS Batch has ended support for Amazon Linux AMI as it will not receive any security updates or bug fixes starting January 1, 2024. For more information about the Amazon Linux end-of-life, see [AL FAQ](#).

We recommend that you update existing Amazon Linux based compute environments to Amazon Linux 2023 to prevent unforeseen workload interruptions, and continue to receive security and other updates.
Your compute environments using the Amazon Linux AMI may continue functioning beyond the December 31, 2023 end-of-life date. However, these compute environments will no longer receive any new software updates, security patches, or bug fixes from AWS. It is your responsibility to maintain these compute environments on the Amazon Linux AMI after end-of-life. We recommend migrating AWS Batch compute environments to Amazon Linux 2023 or Amazon Linux 2 to maintain optimal performance and security.

For help migrating AWS Batch from the Amazon Linux AMI to Amazon Linux 2023 or Amazon Linux 2, see Updating compute environments - AWS Batch

**Allocation strategies**

When a managed compute environment is created, AWS Batch selects instance types from the `instanceTypes` specified that best fit the needs of the jobs. The allocation strategy defines behavior when AWS Batch needs additional capacity. This parameter isn't applicable to jobs that run on Fargate resources. Don't specify this parameter.

**BEST_FIT (default)**

AWS Batch selects an instance type that best fits the needs of the jobs with a preference for the lowest-cost instance type. If additional instances of the selected instance type aren't available, AWS Batch waits for the additional instances to be available. If there aren't enough instances available, or if the user is reaching the [Amazon EC2 service quotas](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/Costquotas.html), then additional jobs don't run until currently running jobs are complete. This allocation strategy keeps costs lower but can limit scaling. If you're using Spot Fleets with BEST_FIT, the Spot Fleet IAM Role must be specified. BEST_FIT isn't supported when updating compute environments. For more information, see [Updating compute environments](https://docs.aws.amazon.com/batch/latest/userguide/compute-environment-update.html).

**Note**

AWS Batch manages AWS resources in your account. Compute environments with the BEST_FIT allocation strategy originally utilized launch configurations by default. However, the use of launch configurations with new AWS accounts will be restricted over time. Therefore, beginning in late April 2024, newly-created BEST_FIT compute environments will default to launch templates. If your service role lacks permissions to manage launch templates, AWS Batch may continue to utilize launch configurations. Existing compute environments will continue to use launch configurations.
BEST_FIT_PROGRESSIVE

AWS Batch selects additional instance types that are large enough to meet the requirements of the jobs in the queue. Instance types with a lower cost for each unit vCPU are preferred. If additional instances of the previously selected instance types aren't available, AWS Batch selects new instance types.

SPOT_CAPACITY_OPTIMIZED

AWS Batch selects one or more instance types that are large enough to meet the requirements of the jobs in the queue. Instance types that are less likely to be interrupted are preferred. This allocation strategy is only available for Spot Instance compute resources.

SPOT_PRICE_CAPACITY_OPTIMIZED

The price and capacity optimized allocation strategy looks at both price and capacity to select the Spot Instance pools that are the least likely to be interrupted and have the lowest possible price. This allocation strategy is only available for Spot Instance compute resources.

Note

We recommend that you use SPOT_PRICE_CAPACITY_OPTIMIZED rather than SPOT_CAPACITY_OPTIMIZED in most instances.

The BEST_FIT_PROGRESSIVE and BEST_FIT strategies use On-Demand or Spot Instances, and the SPOT_CAPACITY_OPTIMIZED and SPOT_PRICE_CAPACITY_OPTIMIZED strategies use Spot Instances. However, AWS Batch might need to exceed maxvCpus to meet your capacity requirements. In this event, AWS Batch never exceeds maxvCpus by more than a single instance.

Updating compute environments

After you create a compute environment that uses EC2 resources, you can update many of the settings of the compute environment directly. However, changing some of the settings requires that AWS Batch replace the instances in the compute environment.

For compute environments that use Fargate resources, you can update the following.

- securityGroupIds
- subnets
AWS Batch has two update mechanisms. The first is a scaling update where instances are added or removed from the compute environment. The second is an infrastructure update where the instances in the compute environment are replaced. An infrastructure update takes much longer than a scaling update.

If you update compute environments with AWS Batch, changing only these settings causes a scaling update: desired vCPUs (desiredvCpus), maximum vCPUs (maxvCpus), minimum vCPUs (minvCpus), service role (serviceRole), and state (state).

**Note**

When you update the desiredvCpus setting, the value must be between the minvCpus and maxvCpus values. Additionally, the updated desiredvCpus value must be greater than or equal to the current desiredvCpus value. For more information, see the section called “Error message when you update the desiredvCpus setting”.

If any of the following settings are changed in an `UpdateComputeEnvironment` API action, AWS Batch initiates an infrastructure update. An infrastructure update requires that the service role is set to `AWSServiceRoleForBatch` (the default) and that the allocation strategy is `BEST_FIT_PROGRESSION`, `SPOT_CAPACITY_OPTIMIZED`, or `SPOT_PRICE_CAPACITY_OPTIMIZED`. `BEST_FIT` isn't supported. Except for service role, all of the settings that can be changed for a scaling update can also be changed for an infrastructure update.

**Note**

We recommend that you use `SPOT_PRICE_CAPACITY_OPTIMIZED` rather than `SPOT_CAPACITY_OPTIMIZED` in most instances.

During an infrastructure update, the status of the compute environment changes to UPDATING. New instances are launched using the updated settings. New jobs are scheduled on the new
instances. Jobs that are currently running are dispatched according to the infrastructure update policy. For more information, see UpdateComputeEnvironment and UpdatePolicy in the AWS Batch API Reference.

In the UpdatePolicy data type, consider the following scenarios:

**Note**
In these scenarios, the following is true. When an instance is terminated, running jobs are stopped. By default, these jobs aren’t retried. To retry one of these jobs after an instance is terminated, configure a job retry strategy. For more information, see the section called “Automated job retries” in the AWS Batch User Guide.

- If the terminateJobsOnUpdate setting is set to true, running jobs are terminated during an infrastructure update. The jobExecutionTimeoutMinutes setting is ignored.
- If the terminateJobsOnUpdate setting is set to false, jobs can run for additional time after the infrastructure update occurs. This additional time is configured in the jobExecutionTimeoutMinutes setting. By default, the jobExecutionTimeoutMinutes setting is 30 minutes.

As capacity becomes available in the compute environment, new instances are launched with the updated settings and jobs are started on the new instances. As all of the jobs complete on instances with the old settings, the old instances are terminated. What capacity becoming available means is that desired number of vCPUs is below the maximum number of vCPUs by at least as many vCPUs as required by the smallest instance type.

**Infrastructure updates**

An infrastructure update is required to change some settings for a compute environment. If any of the following settings are changed, an infrastructure update is started:

**Important**
The compute environment must use the AWSServiceRoleForBatch service-linked role to make changes that require an infrastructure update.

If the compute environment uses a service-linked role, it can't be changed to use a regular IAM role. Likewise, if the compute environment has a regular IAM role, it can't be changed
to use a service-linked role. Therefore, you can only perform infrastructure updates on compute environments that were created by using a service-linked role.

- Allocation strategy (allocationStrategy, must be either BEST_FIT_PROGRESSION, SPOT_CAPACITY_OPTIMIZED, or SPOT_PRICE_CAPACITY_OPTIMIZED. If the original allocation strategy is BEST_FIT, infrastructure updates aren't supported.)

```
Note

We recommend that you use SPOT_PRICE_CAPACITY_OPTIMIZED rather than SPOT_CAPACITY_OPTIMIZED in most instances.
```

- Bid percentage (bidPercentage)
- EC2 configuration (ec2Configuration)
- Key pair (ec2KeyPair)
- Image ID (imageId)
- Instance role (instanceRole)
- Instance types (instanceTypes)
- Launch template (launchTemplate)
- Placement group (placementGroup)
- Security groups (securityGroupIds)
- VPC subnets (subnets)
- EC2 tags (tags)
- Compute environment type (type, can be one of EC2 or SPOT)
- Whether to update to the latest AMI that's supported by AWS Batch during an infrastructure update updateToLatestImageVersion

### Updating the AMI ID

During an infrastructure update, the compute environment's AMI ID might change, depending on whether AMIs are specified in any of these three settings. AMIs are specified in the imageId (in computeResources), imageIdOverride (in ec2Configuration), or the launch template.
specified in launchTemplate. Suppose that no AMI IDs are specified in any of those settings and the updateToLatestImageVersion setting is true. Then, the latest Amazon ECS optimized AMI supported by AWS Batch is used for any infrastructure update.

If an AMI ID is specified in at least one of these settings, the update depends on which setting provided the AMI ID used before the update. When you create a compute environment, the priority for selecting an AMI ID is first the launch template, then the imageId setting, and finally the imageIdOverride setting. However, if the AMI ID that's used came from the launch template, updating either the imageId or imageIdOverride settings doesn't update the AMI ID. The only way to update an AMI ID selected from the launch template is to update the launch template. If the version parameter of the launch template is $Default or $Latest, the default or latest version of the specified launch template is evaluated. If a different AMI ID is selected by the default or the latest version of the launch template is selected, that AMI ID is used in the update.

If the launch template was not used to select the AMI ID, the AMI ID that's specified in the imageId or imageIdOverride parameters is used. If both are specified, the AMI ID specified in the imageIdOverride parameter is used.

Suppose that the compute environment uses an AMI ID specified by the imageId, imageIdOverride, or launchTemplate parameters, and you want to use the latest Amazon ECS optimized AMI supported by AWS Batch. Then, the update must remove the settings that provided AMI IDs. For imageId, this requires specifying an empty string for that parameter. For imageIdOverride, this requires specifying an empty string for the ec2Configuration parameter.

If the AMI ID came from the launch template, you can change to the latest Amazon ECS optimized AMI that's supported by AWS Batch by either one of the following ways:

- Remove the launch template by specifying an empty string for the launchTemplateId or launchTemplateName parameter. This removes the entire launch template, rather than the AMI ID alone.
- If the updated version of the launch template doesn't specify an AMI ID, the updateToLatestImageVersion parameter must be set to true.

**Amazon EKS compute environments**

[Getting started with AWS Batch on Amazon EKS](#) provides a short guide to creating EKS compute environments. This section provides more details on Amazon EKS compute environments.
Default AMI selection

When you create an Amazon EKS compute environment, you don’t need to specify an Amazon Machine Image (AMI). AWS Batch selects an Amazon EKS optimized AMI based on the Kubernetes version and instance types that are specified in your `CreateComputeEnvironment` request. In general, we recommend that you use the default AMI selection. For more information about Amazon EKS optimized AMIs, see Amazon EKS optimized Amazon Linux AMIs in the Amazon EKS User Guide.

Run the following command to see which AMI type AWS Batch is selected for your Amazon EKS compute environment. This following example is a non-GPU instance type.

```
# compute CE example: indicates Batch has chosen the AL2 x86 or ARM EKS 1.29 AMI, depending on instance types
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1 \
    | jq '.computeEnvironments[].computeResources.ec2Configuration'

[  
  {  
    "imageType": "EKS_AL2",  
    "imageKubernetesVersion": "1.29"  
  }  
]
```

This following example is a GPU instance type.

```
# GPU CE example: indicates Batch has chosen the AL2 x86 EKS Accelerated 1.29 AMI
$ aws batch describe-compute-environments --compute-environments My-Eks-GPU-CE \
    | jq '.computeEnvironments[].computeResources.ec2Configuration'

[  
]
```
Supported Kubernetes versions

AWS Batch on Amazon EKS currently supports the following Kubernetes versions:

- 1.29
- 1.28
- 1.27
- 1.26
- 1.25
- 1.24
- 1.23

You might see an error message that resembles the following when you use the CreateComputeEnvironment API operation or UpdateComputeEnvironment API operation to create or update a compute environment. This issue occurs if you specify an unsupported Kubernetes version in EC2Configuration.

At least one imageKubernetesVersion in EC2Configuration is not supported.

To resolve this issue, delete the compute environment and then re-create it with a supported Kubernetes version.

You can perform a minor version upgrade on your Amazon EKS cluster. For example, you can upgrade the cluster from 1.xx to 1.yy even if the minor version isn't supported.

However, the compute environment status might change to INVALID after a major version update. For example, if you perform a major version upgrade from 1.xx to 2.yy. If the major version isn't supported by AWS Batch, you see an error message that resembles the following.

reason=CLIENT_ERROR - ... EKS Cluster version [2.yy] is unsupported
Updating Kubernetes version of compute environment

With AWS Batch, you can update the Kubernetes version of a compute environment to support Amazon EKS cluster upgrades. The Kubernetes version of a compute environment is the Amazon EKS AMI version for the Kubernetes nodes that AWS Batch launches to run jobs. You can perform a Kubernetes version upgrade on their Amazon EKS nodes before or after you update the version of Amazon EKS cluster's control-plane. We recommend that you update the nodes after upgrading the control plane. For more information, see Updating an Amazon EKS cluster Kubernetes version in the Amazon EKS User Guide.

To upgrade the Kubernetes version of a compute environment, use the UpdateComputeEnvironment API operation.

```shell
$ aws batch update-compute-environment \
  --compute-environment <compute-environment-name> \
  --compute-resources \
    'ec2Configuration=[[imageType=EKS_AL2,imageKubernetesVersion=1.23]]'
```

Shared responsibility of the Kubernetes nodes

Maintenance of the compute environments is a shared responsibility.

- Don't change or remove AWS Batch nodes, labels, taints, namespaces, launch templates, or auto scaling groups. Don't add taints to AWS Batch managed nodes. If you make any of these changes, your compute environment cannot be supported and failures including idle instances occur.
- Don't target your pods to AWS Batch managed nodes. If you target your pods to the managed nodes, broken scaling and stuck job queues occur. Run workloads that don't use AWS Batch on self-managed nodes or managed node groups. For more information, see Managed node groups in the Amazon EKS User Guide.
- You can target a DaemonSet to run on AWS Batch managed nodes. For more information, see Running a DaemonSet on AWS Batch managed nodes.

AWS Batch doesn't automatically update compute environment AMIs. It's your responsibility to update them. Run the following command to update your AMIs to the latest AMI version.

```shell
$ aws batch update-compute-environment \
  --compute-environment <compute-environment-name> \
```
AWS Batch doesn't automatically upgrade the Kubernetes version. Run the following command to update the Kubernetes version of your computer environment to 1.23.

```bash
$ aws batch update-compute-environment \
  --compute-environment <compute-environment-name> \
  --compute-resources \ 
  'ec2Configuration=[{imageType=EKS_AL2,imageKubernetesVersion=1.23}]'
```

When updating to a more recent AMI or the Kubernetes version, you can specify whether to terminate jobs when they're updated (`terminateJobsOnUpdate`) and how long to wait for before an instance is replaced if running jobs don't finish (`jobExecutionTimeoutMinutes`). For more information, see [Updating compute environments](#) and the infrastructure update policy ([UpdatePolicy](#)) set in the [UpdateComputeEnvironment](#) API operation.

### Running a DaemonSet on AWS Batch managed nodes

AWS Batch sets taints on AWS Batch managed Kubernetes nodes. You can target a DaemonSet to run on AWS Batch managed nodes with the following tolerations.

```yaml
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
```

Another way to do this is with the following tolerations.

```yaml
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
  effect: "NoSchedule"
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
  effect: "NoExecute"
```

### Customizing with launch templates

AWS Batch on Amazon EKS supports launch templates. There are constraints on what your launch template can do.
Important

AWS Batch runs /etc/eks/bootstrap.sh. Don't run /etc/eks/bootstrap.sh in your launch template or cloud-init user-data scripts. You can add additional parameters besides the --kubelet-extra-args parameter to bootstrap.sh. To do this, set the AWS_BATCH_KUBELET_EXTRA_ARGS variable in the /etc/aws-batch/batch.config file. See the following example for details.

Note

If the launch template is changed after CreateComputeEnvironment is called, UpdateComputeEnvironment must be called to evaluate the version of the launch template for replacement.

Topics

- Adding kubelet extra arguments
- Configuring the container runtime
- Mounting an Amazon EFS volume
- IPv6 support

Adding kubelet extra arguments

AWS Batch supports adding extra arguments to the kubelet command. For the list of supported parameters, see kubelet in the Kubernetes documentation. In the following example, --node-labels mylabel=helloworld is added to the kubelet command line.

```bash
#!/bin/bash
mkdir -p /etc/aws-batch
```
Configuring the container runtime

You can use the AWS Batch CONTAINER_RUNTIME environment variable to configure the container runtime on a managed node. The following example sets the container runtime to containerd when bootstrap.sh runs. For more information, see containerd in the Kubernetes documentation.

```
>Note
The CONTAINER_RUNTIME environment variable is equivalent to the --container-runtime option of bootstrap.sh. For more information, see Options in the Kubernetes documentation.
```

Mounting an Amazon EFS volume

You can use launch templates to mount volumes to the node. In the following example, the cloud-config packages and runcmd settings are used. For more information, see Cloud config examples in the cloud-init documentation.

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

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
mkdir -p /etc/aws-batch
mkdir -p /etc/aws-batch
echo CONTAINER_RUNTIME=containerd >> /etc/aws-batch/batch.config
```

Customizing with launch templates
To use this volume in the job, it must be added in the `eksProperties` parameter to `RegisterJobDefinition`. The following example is a large portion of the job definition.

```json
{
    "jobDefinitionName": "MyJobOnEks_EFS",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["ls", "-la", "/efs"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    },
                    "volumeMounts": [
                        {
                            "name": "efs-volume",
                            "mountPath": "/efs"
                        }
                    ]
                }
            ]
        }
    }
}
```
In the node, the Amazon EFS volume is mounted in the `/mnt/efs` directory. In the container for the Amazon EKS job, the volume is mounted in the `/efs` directory.

### IPv6 support

AWS Batch supports Amazon EKS clusters that have IPv6 addresses. No customizations are required for AWS Batch support. However, before you begin, we recommend that you review the considerations and conditions that are outlined in Assigning IPv6 addresses to pods and services in the Amazon EKS User Guide.

### Compute Resource Memory Management

When the Amazon ECS container agent registers a compute resource into a compute environment, the agent must determine how much memory the compute resource has available to reserve for your jobs. Because of platform memory overhead and memory occupied by the system kernel, this number is different than the installed memory amount for Amazon EC2 instances. For example, an `m4.large` instance has 8 GiB of installed memory. However, this doesn't always translate to exactly 8192 MiB of memory available for jobs when the compute resource registers.

Suppose that you specify 8192 MiB for the job, and none of your compute resources have 8192 MiB or greater of memory available to satisfy this requirement. Then, the job can't be placed in your compute environment. If you're using a managed compute environment, AWS Batch must launch a larger instance type to accommodate the request.

The default AWS Batch compute resource AMI also reserves 32 MiB of memory for the Amazon ECS container agent and other critical system processes. This memory isn't available for job allocation. For more information, see Reserving System Memory.
The Amazon ECS container agent uses the Docker ReadMemInfo( ) function to query the total memory available to the operating system. Linux provides command line utilities to determine the total memory.

**Example - Determine Linux total memory**

The `free` command returns the total memory that's recognized by the operating system.

```
$ free -b
```

The following is example output for an `m4.large` instance that's 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. This means that there's 7985 MiB available for tasks.

**Reserving System Memory**

If you occupy all of the memory on a compute resource with your jobs, it's possible that your jobs contend with critical system processes for memory and possibly cause a system failure. The Amazon ECS container agent provides a configuration variable that's called `ECS_RESERVED_MEMORY`. You can use this configuration variable to remove a specified number of MiB of memory from the pool that's allocated to your jobs. This effectively reserves that memory for critical system processes.

The default AWS Batch compute resource AMI reserves 32 MiB of memory for the Amazon ECS container agent and other critical system processes.

**Viewing Compute Resource Memory**

You can view how much memory a compute resource registers with in the Amazon ECS console or with the [DescribeContainerInstances](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_DescribeContainerInstances.html) API operation. If you're trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, you can observe the memory available for that compute resource and then assign your jobs that much memory.
To view compute resource memory

2. Choose Clusters, and then choose the cluster that hosts your compute resources to view.
   
   The cluster name for your compute environment begins with your compute environment name.
3. Choose Infrastructure.
4. Under Container instances, choose the container instance.
5. The Resources and networking section shows the registered and available memory for the compute resource.
   
   The Registered memory value is what the compute resource registered with Amazon ECS when it was first launched, and the Available memory value is what hasn't already been allocated to jobs.

Memory and vCPU considerations for AWS Batch on Amazon EKS

In AWS Batch on Amazon EKS, you can specify the resources that are made available to a container. For example, you can specify requests or limits values for vCPU and memory resources.

The following are constraints for specifying vCPU resources:

• At least one vCPU requests or limits value must be specified.
• One vCPU unit is equivalent to one physical or virtual core.
• The vCPU value must be entered in whole numbers or in increments of 0.25.
• The smallest valid vCPU value is 0.25.
• If both are specified, the requests value must be less than or equal to the limits value. This way, you can configure both soft and hard vCPU configurations.
• vCPU values can't be specified in milliCPU form. For example, 100m isn't a valid value.
• AWS Batch uses the requests value for scaling decisions. If a requests value isn't specified, the limits value is copied to the requests value.

The following are constraints for specifying memory resources:

• At least one memory requests or limits value must be specified.
Memory values must be in mebibytes (MiBs).

If both are specified, the requests value must be equal to the limits value.

AWS Batch uses the requests value for scaling decisions. If a requests value is not specified, the limits value is copied to the requests value.

The following are constraints for specifying GPU resources:

- If both are specified, the requests value must be equal to the limits value.
- AWS Batch uses the requests value for scaling decisions. If a requests value isn't specified, the limits value is copied to the requests value.

**Example job definitions**

The following AWS Batch on Amazon EKS job definition configures soft vCPU shares. This lets AWS Batch on Amazon EKS use all of the vCPU capacity for the instance type. However, if there are other jobs running, the job is allocated a maximum of 2 vCPUs. Memory is limited to 2 GB.

```json
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "requests": {
                            "cpu": "2",
                            "memory": "2048Mi"
                        }
                    }
                }
            ]
        }
    }
}
```
The following AWS Batch on Amazon EKS job definition has a request value of 1 and allocates a maximum of 4 vCPUs to the job.

```json
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "requests": {
                            "cpu": "1"
                        },
                        "limits": {
                            "cpu": "4",
                            "memory": "2048Mi"
                        }
                    }
                }
            ]
        }
    }
}
```

The following AWS Batch on Amazon EKS job definition sets a vCPU limits value of 1 and a memory limits value of 1 GB.

```json
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    }
                }
            ]
        }
    }
}
```
When AWS Batch translates an AWS Batch on Amazon EKS job into an Amazon EKS pod, AWS Batch copies the limits value to the requests value. This is if a requests value isn’t specified. When you submit the preceding example job definition, the pod spec is as follows.

```yaml
apiVersion: v1
kind: Pod
...
spec:
  ...  
  containers:
    - command:
      - sleep
      - 60
      image: public.ecr.aws/amazonlinux/amazonlinux:2
  resources:
    limits:
      cpu: 1
      memory: 1024Mi
    requests:
      cpu: 1
      memory: 1024Mi
...
```

### Node CPU and memory reservations

AWS Batch relies on the default logic of the bootstrap.sh file for vCPU and memory reservations. For more information about the bootstrap.sh file, see [bootstrap.sh](#). When you size your vCPU and memory resources, consider the examples that follow.
Note

If no instances are running, vCPU and memory reservations can initially affect AWS Batch scaling logic and decision making. After the instances are running, AWS Batch adjusts the initial allocations.

Node CPU reservation example

The CPU reservation value is calculated in millicores using the total number of vCPUs that are available to the instance.

<table>
<thead>
<tr>
<th>vCPU number</th>
<th>Percentage reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6%</td>
</tr>
<tr>
<td>2</td>
<td>1%</td>
</tr>
<tr>
<td>3-4</td>
<td>0.5%</td>
</tr>
<tr>
<td>4 and above</td>
<td>0.25%</td>
</tr>
</tbody>
</table>

Using the preceding values, the following is true:

- The CPU reservation value for a c5.large instance with 2 vCPUs is 70 m. This is calculated in the following way: (1*60) + (1*10) = 70 m.
- The CPU reservation value for a c5.24xlarge instance with 96 vCPUs is 310 m. This is calculated in the following way: (1*60) + (1*10) + (2*5) + (92*2.5) = 310 m.

In this example, there are 1930 (calculated 2000-70) millicore vCPU units available to run jobs on a c5.large instance. Suppose your job requires 2 (2*1000 m) vCPU units, the job doesn’t fit on a single c5.large instance. However, a job that requires 1.75 vCPU units fits.

Node memory reservation example

The memory reservation value is calculated in mebibytes using the following:

- The instance capacity in mebibytes. For example, an 8 GB instance is 7,748 MiB.
• The kubeReserved value. The kubeReserved value is the amount of memory to reserve for system daemons. The kubeReserved value is calculated in the following way: ($(11 \times \text{maximum number of pods that is supported by the instance type}) + 255$). For information about the maximum number of pods that's supported by an instance type, see eni-max-pods.txt

• The HardEvictionLimit value. When available memory falls below the HardEvictionLimit value, the instance attempts to evict pods.

The formula to calculate the allocatable memory is as follows: $(\text{instance_capacity_in_MiB} - (11 \times \text{maximum_number_of_pods}) - 255 - \text{HardEvictionLimit value}).$

A c5.large instance supports up to 29 pods. For an 8 GB c5.large instance with a HardEvictionLimit value of 100 MiB, the allocatable memory is 7074 MiB. This is calculated in the following way: $(7748 - (11 \times 29) - 255 - 100) = 7074 \text{ MiB}$. In this example, an 8,192 MiB job doesn't fit on this instance even though it's an 8 gibibyte (GiB) instance.

**DaemonSets**

When you use DaemonSets, consider the following:

• If no AWS Batch on Amazon EKS instances are running, DaemonSets can initially affect AWS Batch scaling logic and decision making. AWS Batch initially allocates 0.5 vCPU units and 500 MiB for expected DaemonSets. After the instances are running, AWS Batch adjusts the initial allocations.

• If a DaemonSet defines vCPU or memory limits, AWS Batch on Amazon EKS jobs have fewer resources. We recommend that you keep the number of DaemonSets that are assigned to AWS Batch jobs as low as possible.
Scheduling policies

You can use scheduling policies to configure how compute resources in a job queue are allocated between users or workloads. Using scheduling policies, you can assign different fair share identifiers to workloads or users. AWS Batch assigns each fair share identifier a percentage of the total resources that are available during a period of time.

The fair share percentage is calculated using the shareDecaySeconds and shareDistribution values. You can add time to the fair share analysis by assigning a share decay time to the policy. Adding time gives more weight to time and less to the defined weight. You can hold compute resources in reserve for fair share identifiers that aren't active by specifying a compute reservation. For more information, see Scheduling policy parameters.

Topics
• Creating a scheduling policy
• Scheduling policy parameters

Creating a scheduling policy

Before you can create a job queue with a scheduling policy, you must create a scheduling policy. When you create a scheduling policy, you associate one or more fair share identifiers or fair share identifier prefixes with weights for the queue and optionally assign a decay period and compute reservation to the policy.

To create a scheduling policy

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Scheduling policies, Create.
4. For Name, enter a unique name for your scheduling policy. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
5. (Optional) For Share decay seconds, enter an integer value for the scheduling policy's share decay time. A longer share decay time will use consider compute resource usage over a longer time when scheduling jobs. This can allow jobs using a fair share identifier to temporarily use more compute resources than the weight for that fair share identifier would allow if that fair share identifier had not recently been using compute resources.
6. **(Optional) For Compute reservation,** enter an integer value for the scheduling policy's compute reservation. The compute reservation will hold some vCPUs in reserve to be used for fair share identifiers that are not currently active.

The reserved ratio is \((computeReservation/100)^{ActiveFairShares}\) where \(ActiveFairShares\) is the number of active fair share identifiers.

For example, a computeReservation value of 50 indicates that AWS Batch should reserve 50% of the maximum available VCPU if there is only one fair share identifier, 25% if there are two fair share identifiers, and 12.5% if there are three fair share identifiers. A computeReservation value of 25 indicates that AWS Batch should reserve 25% of the maximum available VCPU if there is only one fair share identifier, 6.25% if there are two fair share identifiers, and 1.56% if there are three fair share identifiers.

7. In the **Share attributes** section, you can specify the fair share identifier and weight for each fair share identifier to associate with the scheduling policy.
   a. Choose **Add share identifier**.
   b. For **Share identifier**, specify the fair share identifier. If the string ends with '*', this becomes a fair share identifier prefix used to match fair share identifiers for jobs. All of the fair share identifiers and fair share identifier prefixes in a scheduling policy must be unique and cannot overlap. For example, you can't have fair share identifiers prefix 'UserA*' and fair share identifier 'UserA1' in the same scheduling policy.
   c. For **Weight factor**, specify the relative weight for the fair share identifier. The default value is 1.0. A lower value has a higher priority for compute resources. If a fair share identifier prefix is used, jobs with fair share identifiers that start with the prefix will share the weight factor. This effectively increases the weight factor for those jobs, lowering their individual priority but maintaining the same weight factor for the fair share identifier prefix.

8. **(Optional) In the Tags section,** you can specify the key and value for each tag to associate with the scheduling policy. For more information, see [Tagging your AWS Batch resources](#).

9. Choose **Submit** to finish and create your scheduling policy.

**Scheduling policy template**

An empty scheduling policy template is shown below. You can use this template to create your scheduling policy which can then be saved to a file and used with the AWS CLI --cli-input-
json option. For more information about these parameters, see CreateSchedulingPolicy in the AWS Batch API Reference.

```
{
   "name": "",
   "fairsharePolicy": {
      "shareDecaySeconds": 0,
      "computeReservation": 0,
      "shareDistribution": [
         {
            "shareIdentifier": "",
            "weightFactor": 0.0
         }
      ],
   },
   "tags": {
      "KeyName": ""
   }
}
```

Note
You can generate the preceding job queue template with the following AWS CLI command.

```
$ aws batch create-scheduling-policy --generate-cli-skeleton
```

Scheduling policy parameters

Scheduling policies are split into three basic components: the name, fair share policy, and tags of the scheduling policy.

Topics

- Scheduling policy name
- Fair share policy
- Tags
Scheduling policy name

name

The name for your scheduling policy. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Type: String

Required: Yes

Fair share policy

fairsharePolicy

The fair share policy of the scheduling policy.

```
"fairsharePolicy": {
    "computeReservation": number,
    "shareDecaySeconds": number,
    "shareDistribution": [
        {
            "shareIdentifier": "string",
            "weightFactor": number
        }
    ]
}
```

Type: Object

Required: No

computeReservation

A value used to reserve some of the available maximum VCPU for fair share identifiers that have not yet been used.

The reserved ratio is \((computeReservation/100)^{ActiveFairShares}\) where \(ActiveFairShares\) is the number of active fair share identifiers.

For example, a computeReservation value of 50 indicates that AWS Batch should reserve 50% of the maximum available VCPU if there is only one active fair share identifier, 25%
if there are two active fair share identifiers, and 12.5% if there are three active fair share identifiers. A `computeReservation` value of 25 indicates that AWS Batch should reserve 25% of the maximum available VCPU if there is only one active fair share identifier, 6.25% if there are two active fair share identifiers, and 1.56% if there are three active fair share identifiers.

Type: Integer

Valid range: Minimum value of 0. Maximum value of 99.

Required: No

`shareDecaySeconds`

The time period to use to calculate a fair share percentage for each fair share identifier in use. A value of zero (0) indicates that only current usage should be measured. The decay allows for more recently run jobs to have more weight than jobs that ran earlier.

Type: Integer

Valid range: Minimum value of 0. Maximum value of 604800 (1 week).

Required: No

`shareDistribution`

Array of objects that contain the weights for the fair share identifiers for the fair share policy. Fair share identifiers that are not included have a default weight of 1.0.

```json
"shareDistribution": [ 
  {
    "shareIdentifier": "string",
    "weightFactor": number
  }
]
```

Type: Array

Required: No

`shareIdentifier`

A fair share identifier or fair share identifier prefix. If the string ends with "*" then this string specifies a fair share identifier prefix for fair share identifiers that begin with that
prefix. For example if the value is `UserA*` and the `weightFactor` is 1 and there are two fair share identifiers that begin with `UserA`, then each of those fair share identifiers will have a weight of 2; if there are five such fair share identifiers, then each would have a weight of 5.

The list of fair share identifiers and fair share identifier prefixes in a fair share policy cannot overlap. For example you cannot have a fair share identifier prefix of `UserA*` and a fair share identifier of `UserA-1` in the same fair share policy.

Type: String  
Required: Yes

`weightFactor`

The weight factor for the fair share identifier. The default value is 1.0. A lower value has a higher priority for compute resources. For example, jobs that use a share identifier with a weight factor of 0.125 (1/8) get 8 times the compute resources of jobs that use a share identifier with a weight factor of 1.

The smallest supported value is 0.0001 and the largest supported value is 999.9999.

Type: Float  
Required: No

**Tags**

tags

Key-value pair tags to associate with the scheduling policy. For more information, see [Tagging your AWS Batch resources](https://docs.aws.amazon.com/batch/latest/userguide/tagging.html).

Type: String to string map  
Required: No
Orchestrate AWS Batch jobs with Step Functions state machines in the AWS Batch console

You can use the AWS Batch console to view details about your Step Functions state machines and the functions that they use.

Sections

- Viewing state machine details
- Editing a state machine
- Running a state machine

Viewing state machine details

The AWS Batch console displays a list of your state machines in the current AWS Region that contain at least one workflow step that submits a AWS Batch job.

Choose a state machine to view a graphical representation of the workflow. Steps highlighted in blue represent AWS Batch jobs. Use the graph controls to zoom in, zoom out, and center the graph.

Note

When a AWS Batch job is dynamically referenced with JsonPath in the state machine definition, the function details cannot be shown in the AWS Batch console. Instead, the job name is listed as a Dynamic reference, and the corresponding steps in the graph are grayed out.

To view state machine details

1. Open the AWS Batch console Workflow orchestration powered by Step Functions page.
2. Choose a state machine.

   The AWS Batch console opens the Details page.
Editing a state machine

When you want to edit a state machine, AWS Batch opens the Edit definition page of the Step Functions console.

To edit a state machine

1. Open the AWS Batch console Workflow orchestration powered by Step Functions page.
2. Choose a state machine.
3. Choose Edit.

   The Step Functions console opens the Edit definition page.
4. Edit the state machine and choose Save.

For more information about editing state machines, see Step Functions state machine language in the AWS Step Functions Developer Guide.

Running a state machine

When you want to run a state machine, AWS Batch opens the New execution page of the Step Functions console.

To run a state machine

1. Open the AWS Batch console Workflow orchestration powered by Step Functions page.
2. Choose a state machine.
3. Choose Execute.

   The Step Functions console opens the New execution page.
4. (Optional) Edit the state machine and choose Start execution.

For more information about running state machines, see Step Functions state machine execution concepts in the AWS Step Functions Developer Guide.
AWS Batch on AWS Fargate

AWS Fargate is a technology that you can use with AWS Batch 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 jobs with Fargate resources, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate job has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another job.

Contents

- When to use Fargate
- Job definitions on Fargate
- Job queues on Fargate
- Compute environments on Fargate

When to use Fargate

We recommend using Fargate in most scenarios. Fargate launches and scales the compute to closely match the resource requirements that you specify for the container. With Fargate, you don't need to over-provision or pay for additional servers. You also don't need to worry about the specifics of infrastructure-related parameters such as instance type. When the compute environment needs to be scaled up, jobs that run on Fargate resources can get started more quickly. Typically, it takes a few minutes to spin up a new Amazon EC2 instance. However, jobs that run on Fargate can be provisioned in about 30 seconds. The exact time required depends on several factors, including container image size and number of jobs.

However, we recommend that you use Amazon EC2 if your jobs require any of the following:

- More than 16 vCPUs
- More than 120 gibibytes (GiB) of memory
- A GPU
- A custom Amazon Machine Image (AMI)
• Any of the linuxParameters parameters

If you have a large number of jobs, we recommend that you use Amazon EC2 infrastructure. For example, if the number of concurrently running jobs exceeds the Fargate throttling limits. This is because, with EC2, jobs can be dispatched at a higher rate to EC2 resources than to Fargate resources. Moreover, more jobs can run concurrently when you use EC2. For more information, see AWS Fargate service quotas in the Amazon Elastic Container Service Developer Guide.

Job definitions on Fargate

AWS Batch jobs on Fargate don’t support all of the job definition parameters that are available. Some parameters are not supported at all, and others behave differently for Fargate jobs.

The following list describes job definition parameters that are not valid or otherwise restricted in Fargate jobs.

platformCapabilities

Must be specified as FARGATE.

"platformCapabilities": [ "FARGATE" ]

type

Must be specified as container.

"type": "container"

Parameters in containerProperties

executionRoleArn

Must be specified for jobs running on Fargate resources. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
fargatePlatformConfiguration

(Optional, only for Fargate job definitions). Specifies the Fargate platform version, or LATEST for a recent platform version. Possible values for platformVersion are 1.3.0, 1.4.0, and LATEST (default).

```
"fargatePlatformConfiguration": { "platformVersion": "1.4.0" }
```

instanceType, ulimits

Not applicable for jobs running on Fargate resources.

memory, vcpus

These settings must be specified in resourceRequirements

privileged

Either don't specify this parameter, or specify false.

```
"privileged": false
```

resourceRequirements

Both memory and vCPU requirements must be specified using supported values. GPU resources aren't supported for jobs that run on Fargate resources.

If you use GuardDuty Runtime Monitoring, there is a slight memory overhead for the GuardDuty security agent. Therefore the memory limit must include the size of the GuardDuty security agent. For information about the GuardDuty security agent memory limits, see CPU and memory limits in the GuardDuty User Guide. For information about the best practices, see How do I remediate out of memory errors on my Fargate tasks after enabling Runtime Monitoring in the Amazon ECS Developer Guide.

```
"resourceRequirements": [   {"type": "MEMORY", "value": "512"},   {"type": "VCPU", "value": "0.25"}
]
```

Parameters in linuxParameters

- devices, maxSwap, sharedMemorySize, swappiness, tmpfs

Not applicable for jobs that run on Fargate resources.
Parameters in `logConfiguration`:

**logDriver**

Only `awslogs` and `splunk` are supported. For more information, see [Using the `awslogs` log driver](#).

Members in `networkConfiguration`:

**assignPublicIp**

If the private subnet doesn't have a NAT gateway attached to send traffic to the Internet, `assignPublicIp` must be "ENABLED". For more information, see [AWS Batch execution IAM role](#).

---

**Job queues on Fargate**

AWS Batch job queues on Fargate are essentially unchanged. The only restriction is that the compute environments that are listed in `computeEnvironmentOrder` must all be Fargate compute environments (FARGATE or FARGATE_SPOT). EC2 and Fargate compute environments can't be mixed.

**Compute environments on Fargate**

AWS Batch compute environments on Fargate don't support all of the compute environment parameters that are available. Some parameters are not supported at all. Others have specific requirements for Fargate.

The following list describes compute environment parameters that aren't valid or otherwise restricted in Fargate jobs.

**type**

This parameter must be `MANAGED`.

"type": "MANAGED"
Parameters in the `computeResources` object

- `allocationStrategy`, `bidPercentage`, `desiredvCpus`, `imageId`, `instanceTypes`, `ec2Configuration`, `ec2KeyPair`, `instanceRole`, `launchTemplate`, `minvCpus`, `placementGroup`, `spotIamFleetRole`

These aren't applicable for Fargate compute environments and can't be provided.

- `subnets`

  If the subnets listed in this parameter don't have NAT gateways attached, the `assignPublicIp` parameter in the job definition must be set to `ENABLED`.

- `tags`

  This isn't applicable for Fargate compute environments and can't be provided. To specify tags for Fargate compute environments, use the `tags` parameter that's not in the `computeResources` object.

- `type`

  This must be either `FARGATE` or `FARGATE_SPOT`.

```json
"type": "FARGATE_SPOT"
```
AWS Batch simplifies your batch workloads on Amazon EKS clusters by providing managed batch capabilities. This includes queuing, dependency tracking, managed job retries and priorities, pod management, and node scaling. AWS Batch can handle multiple Availability Zones and multiple Amazon EC2 instance types and sizes. AWS Batch integrates several of the Amazon EC2 Spot best practices to run your workloads in a fault-tolerant manner, allowing for fewer interruptions. You can use AWS Batch to run a handful of overnight jobs or millions of mission-critical jobs with confidence.
AWS Batch is a managed service that orchestrates batch workloads in your Kubernetes clusters that are managed by Amazon Elastic Kubernetes Service (Amazon EKS). AWS Batch conducts this orchestration external to your clusters using an “overlay” model. Since AWS Batch is a managed service, there are no Kubernetes components (for example, Operators or Custom Resources) to install or manage in your cluster. AWS Batch only needs your cluster to be configured with Role-Based Access Controls (RBAC) that allow AWS Batch to communicate with the Kubernetes API server. AWS Batch calls Kubernetes APIs to create, monitor, and delete Kubernetes pods and nodes.

AWS Batch has built-in scaling logic to scale Kubernetes nodes based on job queue load with optimizations in terms of job capacity allocations. When the job queue is empty, AWS Batch scales down the nodes to the minimum capacity that you set, which by default is zero. AWS Batch manages the full lifecycle of these nodes, and decorates the nodes with labels and taints. This way,
other Kubernetes workloads aren't placed on the nodes managed by AWS Batch. The exception to this are DaemonSets, which can target AWS Batch nodes to provide monitoring and other functionality required for proper execution of the jobs. Additionally, AWS Batch doesn't run jobs, specifically pods, on nodes in your cluster that it doesn't manage. This way, you can use separate scaling logic and services for other applications on the cluster.

To submit jobs to AWS Batch, you interact directly with the AWS Batch API. AWS Batch translates jobs into podspecs and then creates the requests to place pods on nodes managed by AWS Batch in your Amazon EKS cluster. You can use tools such as kubectl to view running pods and nodes. When a pod has completed its execution, AWS Batch deletes the pod it created to maintain a lower load on the Kubernetes system.

You can get started by connecting a valid Amazon EKS cluster with AWS Batch. Then attach an AWS Batch job queue to it, and register an Amazon EKS job definition using podspec equivalent attributes. Last, submit jobs using the SubmitJob API operation referencing to the job definition. For more information, see Getting started with AWS Batch on Amazon EKS.
Elastic Fabric Adapter

An Elastic Fabric Adapter (EFA) is a network device to accelerate High Performance Computing (HPC) applications. AWS Batch supports applications that use EFA if the following conditions are met.

• For a list of instance types that support EFAs, see Supported instance types in the Amazon EC2 User Guide for Linux Instances.

Tip
To see a list of instance types that support EFAs in an AWS Region, run the following command. Then, cross reference the list that's returned with the list of available instance types in the AWS Batch console.

```bash
$ aws ec2 describe-instance-types  --region us-east-1 --filters Name=network-info.efa-supported,Values=true --query "InstanceTypes[*].[InstanceType]" --output text | sort
```

• For a list of operating systems that support EFA, see Supported operating systems.

• The AMI has the EFA driver loaded.

• The security group for the EFA must allows all inbound and outbound traffic to and from the security group itself.

• All instances that use an EFA must be in the same cluster placement group.

• The job definition must include a devices member with hostPath set to /dev/infiniband/uverbs0 to allow the EFA device to be passed through to the container. If containerPath is specified, it must also be set to /dev/infiniband/uverbs0. If permissions is set it must be set to READ | WRITE | MKNOD.

The location of the LinuxParameters members are different for multi-node parallel jobs and single-node container jobs. The following examples show the differences, but are missing required values.

Example Example for multi-node parallel job

```json
{
   "jobDefinitionName": "EFA-MNP-JobDef",
```
"type": "multinode",
"nodeProperties": {
...
"nodeRangeProperties": [
{
...
"container": {
...
"linuxParameters": {
"devices": [
{
"hostPath": "/dev/infiniband/uverbs0",
"containerPath": "/dev/infiniband/uverbs0",
"permissions": [
"READ", "WRITE", "MKNOD"
]
},
],
],
]
},
},
],
},
},
}

Example Example for single-node container job

{
"jobDefinitionName": "EFA-Container-JobDef",
"type": "container",
...
"containerProperties": {
...
"linuxParameters": {
"devices": [
{
"hostPath": "/dev/infiniband/uverbs0",
},
],
},
],
}
}
For more information about EFA, see Elastic Fabric Adapter in Amazon EC2 User Guide for Linux Instances.
AWS Batch IAM policies, roles, and permissions

By default, users don't have permission to create or modify AWS Batch resources or to perform tasks using the AWS Batch API, AWS Batch console, or the AWS CLI. To allow users to perform these actions, create IAM policies that grant users permission for the specific resources and API operations. Then, attach the policies to the users or groups that require those permissions.

When you attach a policy to a user or group of users, the policy either allows or denies the permissions to perform specific tasks on specific resources. For more information, see Permissions and Policies in the IAM User Guide. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

AWS Batch makes calls to other AWS services on your behalf. As a result, AWS Batch must authenticate using your credentials. More specifically, AWS Batch authenticates by creating an IAM role and policy that provides these permissions. Then, it associates the role with your compute environments when you create them. For more information, see Amazon ECS instance role, IAM Roles, Using Service-Linked Roles, and Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

Getting Started

An IAM policy must grant or deny permissions to use one or more AWS Batch actions.

Topics

- Policy structure
- Supported resource-level permissions for AWS Batch API actions
- Example policies
- AWS Batch managed policy
- Creating AWS Batch IAM policies
- Amazon ECS instance role
- Amazon EC2 spot fleet role
- EventBridge IAM role

Policy structure

The following topics explain the structure of an IAM policy.
Topics

- Policy syntax
- Actions for AWS Batch
- Amazon Resource Names for AWS Batch
- Checking that users have the required permissions

Policy syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows.

```json
{
  "Statement": [ {
    "Effect": "effect",
    "Action": "action",
    "Resource": "arn",
    "Condition": {
      "condition": {
        "key": "value"
      }
    }
  }
  ]
}
```

There are various elements that make up a statement:

- **Effect**: The effect can be Allow or Deny. By default, users don't have permission to use resources and API actions. So, all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.

- **Action**: The action is the specific API action that you're granting or denying permission for. For instructions on how to specify the action, see Actions for AWS Batch.

- **Resource**: The resource that's affected by the action. With some AWS Batch API actions, you can include specific resources in your policy that can be created or modified by the action. To specify a resource in the statement, use its Amazon Resource Name (ARN). For more information, see Supported resource-level permissions for AWS Batch API actions and Amazon Resource Names for AWS Batch. If the AWS Batch API operation currently doesn't support resource-level permissions, include a wildcard (*) to specify that all resources can be affected by the action.
• **Condition**: Conditions are optional. They can be used to control when your policy is in effect.

For more information about example IAM policy statements for AWS Batch, see [Creating AWS Batch IAM policies](#).

**Actions for AWS Batch**

In an IAM policy statement, you can specify any API action from any service that supports IAM. For AWS Batch, use the following prefix with the name of the API action: `batch:` (for example, `batch:SubmitJob` and `batch:CreateComputeEnvironment`).

To specify multiple actions in a single statement, separate each action with a comma.

```
"Action": ["batch:action1", "batch:action2"]
```

You can also specify multiple actions by including a wildcard (*). For example, you can specify all actions with a name that begins with the word "Describe."

```
"Action": "batch:Describe*"
```

To specify all AWS Batch API actions, include a wildcard (*).

```
"Action": "batch:*
```

For a list of AWS Batch actions, see [Actions](#) in the *AWS Batch API Reference*.

**Amazon Resource Names for AWS Batch**

Each IAM policy statement applies to the resources that you specify using their Amazon Resource Names (ARNs).

An Amazon Resource Name (ARN) has the following general syntax:

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

*service*

The service (for example, batch).
region

The AWS Region for the resource (for example, us-east-2).

account

The AWS account ID, with no hyphens (for example, 123456789012).

resourceType

The type of resource (for example, compute-environment).

resourcePath

A path that identifies the resource. You can use a wildcard (*) in your paths.

AWS Batch API operations currently support resource-level permissions on several API operations. For more information, see Supported resource-level permissions for AWS Batch API actions. To specify all resources, or if a specific API action doesn't support ARNs, include a wildcard (*) in the Resource element.

"Resource": "*

Checking that users have the required permissions

Before you put an IAM policy into production, make sure that it grants users the permissions to use the specific API actions and resources that they need.

To do this, first create a user for testing purposes and attach the IAM policy to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

Note

You can also test your policies by using the IAM Policy Simulator. For more information about the policy simulator, see Working with the IAM Policy Simulator in the IAM User Guide.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed. Retest until you get the desired results.
Important

It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow at least five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the DecodeAuthorizationMessage action. For more information, see DecodeAuthorizationMessage in the AWS Security Token Service API Reference, and decode-authorization-message in the AWS CLI Command Reference.

Supported resource-level permissions for AWS Batch API actions

The term resource-level permissions refers to the ability to specify the resources that users are allowed to perform actions on. AWS Batch has partial support for resource-level permissions. For some AWS Batch actions, you can control when users are allowed to use those actions based on conditions that must be met. You can also control based on the specific resources that users are allowed to use. For example, you can grant users permissions to submit jobs, but only to a specific job queue and only with a specific job definition.

The following list describes the AWS Batch API actions that currently support resource-level permissions. The list also describes the supported resources, resource ARNs, and condition keys for each action.

Important

If an AWS Batch API action isn't listed in this list, then it doesn't support resource-level permissions. If an AWS Batch API action doesn't support resource-level permissions, you can grant users permission to use the action. However, you must include a wildcard (*) for the resource element of your policy statement.

Actions

CancelJob, CreateComputeEnvironment, CreateJobQueue, CreateSchedulingPolicy, DeleteComputeEnvironment, DeleteJobQueue, DeleteSchedulingPolicy,
CancelJob

Cancels a job in an AWS Batch queue.

Resource

Job

arn:aws:batch:<region>:<account>:job/<jobId>

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

CreateComputeEnvironment

Creates an AWS Batch compute environment.

Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Condition keys

aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.

aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

CreateJobQueue

Creates an AWS Batch job queue.

Supported resource-level permissions
Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Scheduling Policy


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Condition keys

aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.

aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

DeleteComputeEnvironment

Deletes an AWS Batch compute environment.

Resource

Compute Environment

**Condition keys**

`aws:ResourceTag/${TagKey}` (String)

Filters actions based on the tags that are associated with the resource.

**CreateSchedulingPolicy**

Creates an AWS Batch scheduling policy.

**Resource**

**Scheduling Policy**


**Condition keys**

`aws:ResourceTag/${TagKey}` (String)

Filters actions based on the tags that are associated with the resource.

**Condition keys**

`aws:RequestTag/${TagKey}` (String)

Filters actions based on the tags that are passed in the request.

`aws:TagKeys` (String)

Filters actions based on the tag keys that are passed in the request.

**DeleteJobQueue**

Deletes the specified job queue. Deleting the job queue eventually deletes all of the jobs in the queue. Jobs are deleted at a rate of about 16 jobs each second.

**Resource**

**Job Queue**

`arn:aws:batch:region:account:job-queue/queue-name`

**Condition keys**

`aws:ResourceTag/${TagKey}` (String)

Filters actions based on the tags that are associated with the resource.

**DeleteSchedulingPolicy**

Deletes the specified scheduling policy.
Resource

Scheduling Policy


Condition keys
aws:ResourceTag/${TagKey}(String)

Filters actions based on the tags that are associated with the resource.

DeregisterJobDefinition

Deregisters an AWS Batch job definition.

Resource

Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision

Condition keys
aws:ResourceTag/${TagKey}(String)

Filters actions based on the tags that are associated with the resource.

ListTagsForResource

Lists the tags for the specified resource.

Resource

Compute Environment


Condition keys
aws:ResourceTag/${TagKey}(String)

Filters actions based on the tags that are associated with the resource.

Job

arn:aws:batch:region:account:job/jobId

Condition keys
aws:ResourceTag/${TagKey}(String)

Filters actions based on the tags that are associated with the resource.
**Job Definition**

```yaml
arn:aws:batch:region:account:job-definition/definition-name:revision
```

**Condition keys**

- `aws:ResourceTag/${TagKey}` (String)

  Filters actions based on the tags that are associated with the resource.

**Job Queue**

```yaml
arn:aws:batch:region:account:job-queue/queue-name
```

**Condition keys**

- `aws:ResourceTag/${TagKey}` (String)

  Filters actions based on the tags that are associated with the resource.

**Scheduling Policy**

```yaml
```

**Condition keys**

- `aws:ResourceTag/${TagKey}` (String)

  Filters actions based on the tags that are associated with the resource.

**RegisterJobDefinition**

Registers an AWS Batch definition.

**Resource**

**Job Definition**

```yaml
arn:aws:batch:region:account:job-definition/definition-name
```

**Condition keys**

- `batch:AWSLogsCreateGroup` (Boolean)

  When this parameter is true, the `awslogs-group` is created for the logs.

- `batch:AWSLogsGroup` (String)

  The `awslogs` group where the logs are located.

- `batch:AWSLogsRegion` (String)

  The Region where the logs are sent to.
batch:AWSLogsStreamPrefix (String)

The awslogs log stream prefix.

batch:Image (String)

The Docker image used to start a job.

batch:LogDriver (String)

The log driver used for the job.

batch:Privileged (Boolean)

When this parameter is true, the container for the job is given elevated permissions on the host container instance.

batch:User (String)

The user name or numeric uid to use inside the container for the job.

aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.

aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

SubmitJob

Submits an AWS Batch job from a job definition.

Resource

Job

arn:aws:batch:region:account:job/jobId

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job Definition

arn:aws:batch:region:account:job-definition/definition-name[:revision]
Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Note
This key can only be used when the job definition Amazon Resource Name (ARN) is in the format

Job Queue
arn:aws:batch:region:account:job-queue/queue-name

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

TagResource
Tags the specified resource.

Resource

Compute Environment

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job

arn:aws:batch:region:account:job/jobId

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.
Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Scheduling Policy


Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Condition keys
aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.
aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

TerminateJob

Terminates a job in an AWS Batch job queue.

Resource

Job

arn:aws:batch:region:account:job/jobId
Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

UntagResource

Untags the resource that's specified.

Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job

arn:aws:batch:region:account:job/jobId

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.
Scheduling Policy


Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Condition keys
aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

UpdateComputeEnvironment

Updates an AWS Batch compute environment.

Resource
Compute Environment


Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

UpdateJobQueue

Updates a job queue.

Resource
Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys
aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

Scheduling Policy

**Condition keys**

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

**UpdateSchedulingPolicy**

Updates a scheduling policy.

**Resource**

**Scheduling Policy**


**Condition keys**

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

**Condition keys for AWS Batch API actions**

AWS Batch defines the following condition keys that are used in the Condition element of an IAM policy. You can use these keys to refine the conditions that the policy statement applies to. To view the global condition keys that are available to all services, see available global condition keys in the IAM User Guide.

batch:AWSLogsCreateGroup (Boolean)

When this parameter is true, the awslogs-group is created for the logs.

batch:AWSLogsGroup (String)

The awslogs group where the logs are located.

batch:AWSLogsRegion (String)

The AWS Region where the logs are sent to.

batch:AWSLogsStreamPrefix (String)

The awslogs log stream prefix.

batch:Image (String)

The Docker image used to start a job.
batch:LogDriver (String)

The log driver used for the job.

batch:Privileged (Boolean)

When this parameter is true, the container for the job is given elevated permissions on the host container instance (similar to the root user).

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags that are associated with the resource.

aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.

batch:ShareIdentifier (String)

Filters actions based on the shareIdentifier parameter sent to SubmitJob.

aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

batch:User (String)

The user name or numeric user ID (uid) to use inside the container for the job.

Example policies

The following examples show policy statements that you can use to control the permissions that users have for AWS Batch.

Examples

- Read-only access
- Restricting to POSIX user, Docker image, privilege level, and role on job submission
- Restrict to job definition prefix on job submission
- Restrict to job queue
- Deny action when condition all keys match strings
- Deny action when any condition keys match strings
- Use the batch:ShareIdentifier condition key
Read-only access

The following policy grants users permissions to use all AWS Batch API actions with a name that starts with Describe and List.

Unless another statement grants them permission to do so, users don't have permission to perform any actions on the resources. By default, they're denied permission to use API actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:Describe*",
        "batch:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

Restricting to POSIX user, Docker image, privilege level, and role on job submission

The following policy allows a POSIX user to manage their own set of restricted job definitions.

Use the first and second statements to register and deregister any job definition name whose name is prefixed with `JobDefA_`.

The first statement also uses conditional context keys to restrict the POSIX user, privileged status, and container image values within the `containerProperties` of a job definition. For more information, see `RegisterJobDefinition` in the AWS Batch API Reference. In this example, job definitions can only be registered when the POSIX user is set to `nobody`. The privileged flag is set to `false`. Last, the image is set to `myImage` in an Amazon ECR repository.

⚠️ Important

Docker resolves the `user` parameter to that user `uid` from within the container image. In most cases, this is found in the `/etc/passwd` file within the container image. This name
resolution can be avoided by using direct uid values in both the job definition and any associated IAM policies. Both the AWS Batch API operations and the batch:User IAM conditional keys support numeric values.

Use the third statement to restrict to only a specific role to a job definition.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:RegisterJobDefinition"
            ],
            "Resource": [
            ],
            "Condition": {
                "StringEquals": {
                    "batch:User": ["nobody"],
                    "batch:Image": ["<aws_account_id>.dkr.ecr.<aws_region>.amazonaws.com/myImage"
                ],
                "Bool": {
                    "batch:Privileged": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "batch:DeregisterJobDefinition"
            ],
            "Resource": [
            ]
        },
        {
            "Effect": "Allow",
            "Action": ["batch:SubmitJob"]
        }
    ]
}```
Restrict to job definition prefix on job submission

Use the following policy to submit jobs to any job queue with any job definition name that starts with `JobDefA`.

⚠️ Important
When scoping resource-level access for job submission, you must provide both job queue and job definition resource types.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["batch:SubmitJob"],
            "Resource": [
            ]
        }
    ]
}
```

Restrict to job queue

Use the following policy to submit jobs to a specific job queue that's named `queue1` with any job definition name.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["batch:SubmitJob"],
            "Resource": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/queue1"
            ]
        }
    ]
}
```
Important

When scoping resource-level access for job submission, you must provide both job queue and job definition resource types.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["batch:SubmitJob"],
      "Resource": [
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/*",
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/queue1"
      ]
    }
  ]
}
```

Deny action when condition all keys match strings

The following policy denies access to the RegisterJobDefinition API operation when both the batch:Image (container image ID) condition key is "string1" and the batch:LogDriver (container log driver) condition key is "string2." AWS Batch evaluates condition keys on each container. When a job spans multiple containers such as a multi-node parallel job, it's possible for the containers to have different configurations. If multiple condition keys are evaluated in one statement, they're combined using AND logic. So, if any of the multiple condition keys doesn't match for a container, the Deny effect isn't applied for that container. Rather, a different container in the same job might be denied.

For the list of condition keys for AWS Batch, see Condition keys for AWS Batch in the Service Authorization Reference. Except for batch:ShareIdentifier, all batch condition keys can be used in this way. The batch:ShareIdentifier condition key is defined for a job, not a job definition.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": ["batch:RegisterJobDefinition"],
      "Condition": {
        "StringEqualsPair": {
          "batch:Image": "string1",
          "batch:LogDriver": "string2"
        }
      },
      "ConditionKeyList": ["batch:Image", "batch:LogDriver"]
    }
  ]
}
```
"Statement": [  
  {    
    "Effect": "Allow",    
    "Action": [      
      "batch:RegisterJobDefinition"    
    ],    
    "Resource": [      
      "*"    
    ]  
  },  
  {    
    "Effect": "Deny",    
    "Action": "batch:RegisterJobDefinition",    
    "Resource": "*",    
    "Condition": {      
      "StringEquals": {        
        "batch:Image": "string1",        
        "batch:LogDriver": "string2"      
      }    
    }  
  }  
]

Deny action when any condition keys match strings

The following policy denies access to the RegisterJobDefinition API operation when either the batch:Image (container image ID) condition key is "string1" or the batch:LogDriver (container log driver) condition key is "string2." When a job spans multiple containers such as a multi-node parallel job, it's possible for the containers to have different configurations. If multiple condition keys are evaluated in one statement, they're combined using AND logic. So, if any of the multiple condition keys doesn't match for a container, the Deny effect isn't applied for that container. Rather, a different container in the same job might be denied.

For the list of condition keys for AWS Batch, see Condition keys for AWS Batch in the Service Authorization Reference. Except for batch:ShareIdentifier, all batch condition keys can can be used in this way. (The batch:ShareIdentifier condition key is defined for a job, not a job definition.)

{
  "Version": "2012-10-17",
}
"Statement": [  
    {  
        "Effect": "Allow",  
        "Action": [  
            "batch:RegisterJobDefinition"  
        ],  
        "Resource": [  
            "*"  
        ]  
    },  
    {  
        "Effect": "Deny",  
        "Action": [  
            "batch:RegisterJobDefinition"  
        ],  
        "Resource": [  
            "*"  
        ],  
        "Condition": {  
            "StringEquals": {  
                "batch:Image": [  
                    "string1"  
                ]  
            }  
        }  
    },  
    {  
        "Effect": "Deny",  
        "Action": [  
            "batch:RegisterJobDefinition"  
        ],  
        "Resource": [  
            "*"  
        ],  
        "Condition": {  
            "StringEquals": {  
                "batch:LogDriver": [  
                    "string2"  
                ]  
            }  
        }  
    }  
]
Use the `batch:ShareIdentifier` condition key

Use the following policy to submit jobs that use the `jobDefA` job definition to the `jobqueue1` job queue with the `lowCpu` share identifier.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": [
                "arn:aws::batch:<aws_region>:<aws_account_id>:job-definition/JobDefA",
                "arn:aws::batch:<aws_region>:<aws_account_id>:job-queue/jobqueue1"
            ],
            "Condition": {
                "StringEquals": {
                    "batch:ShareIdentifier": [
                        "lowCpu"
                    ]
                }
            }
        }
    ]
}
```

AWS Batch managed policy

AWS Batch provides a managed policy that you can attach to users that provides permission to use AWS Batch resources and API operations. You can apply this policy directly, or you can use it as a starting point for creating your own policies. For more information about each API operation mentioned in these policies, see [Actions](#) in the AWS Batch API Reference.

**AWSBatchFullAccess**

This policy allows full administrator access to AWS Batch.

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:*",
        "cloudwatch:GetMetricStatistics",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeImages",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ecs:DescribeClusters",
        "ecs:Describe*",
        "ecs:List*",
        "eks:DescribeCluster",
        "eks:ListClusters",
        "logs:Describe*",
        "logs:Get*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "iam:ListInstanceProfiles",
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/service-role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/ecsInstanceRole",
        "arn:aws:iam::*:instance-profile/ecsInstanceRole",
        "arn:aws:iam::*:role/iaws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/AWSBatchJobRole"
      ]
    }
  ]
}
Creating AWS Batch IAM policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to. Then, you can attach those policies to users.

When you attach a policy to a user or group of users, the policy allows or denies the users permission for specific tasks on specific resources. For more information, see Permissions and Policies in the IAM User Guide. For instructions on how to manage and create custom IAM policies, see Managing IAM Policies.

Amazon ECS instance role

AWS Batch compute environments are populated with Amazon ECS container instances. They run the Amazon ECS container agent locally. The Amazon ECS container agent makes calls to various AWS API operations on your behalf. Therefore, container instances that run the agent require an IAM policy and role for these services to recognize that the agent belongs to you. You must create an IAM role and an instance profile for the container instances to use when they're launched. Otherwise, you can't create a compute environment and launch container instances into it. This requirement applies to container instances launched with or without the Amazon ECS optimized AMI provided by Amazon. For more information, see Amazon ECS container instance IAM role in the Amazon Elastic Container Service Developer Guide.

The Amazon ECS instance role and instance profile are automatically created for you in the console first-run experience. However, you can follow these steps to check if your account already has the
Amazon ECS instance role and instance profile. The following steps also cover how to attach the managed IAM policy.

To check for the `ecsInstanceRole` in the IAM console

2. In the navigation pane, choose Roles.
3. Search the list of roles for `ecsInstanceRole`. If the role doesn't exist, use the following steps to create the role.
   a. Choose Create Role.
   b. For Trusted entity type, choose AWS service.
   c. For Common use cases, choose EC2.
   d. Choose Next.
   e. For Permissions policies, search for `AmazonEC2ContainerServiceforEC2Role`.
   f. Choose the check box next to `AmazonEC2ContainerServiceforEC2Role`, then choose Next.
   g. For Role Name, type `ecsInstanceRole` and choose Create Role.

   **Note**
   If you use the AWS Management Console to create a role for Amazon EC2, the console creates an instance profile with the same name as the role.

Alternatively, you can use the AWS CLI to create the `ecsInstanceRole` IAM role. The following example creates an IAM role with a trust policy and an AWS managed policy.

To create an IAM role and instance profile (AWS CLI)

1. Create the following trust policy and save it in a text file that's named `ecsInstanceRole-role-trust-policy.json`.

   ```json
   {
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   
   ```
2. Use the `create-role` command to create the `ecsInstanceRole` role. Specify the trust policy file location in the `assume-role-policy-document` parameter.

```
$ aws iam create-role \
   --role-name ecsInstanceRole \
   --assume-role-policy-document file://ecsInstanceRole-role-trust-policy.json
```

The following is an example response.

```
{
   "Role": {
      "Path": "/",
      "RoleName": "ecsInstanceRole",
      "RoleId": "AROAT46P5RDIY4EXAMPLE",
      "Arn": "arn:aws:iam::123456789012:role/ecsInstanceRole",
      "CreateDate": "2022-12-12T23:46:37.247Z",
      "AssumeRolePolicyDocument": {
         "Version": "2012-10-17",
         "Statement": [
            {
               "Effect": "Allow",
               "Principal": {
                  "Service": "ec2.amazonaws.com"
               },
               "Action": "sts:AssumeRole",
            }
         ]
      }
   }
}
```

3. Use the `create-instance-profile` command to create an instance profile that's named `ecsInstanceRole`.

Amazon ECS instance role
### Note
You need to create roles and instance profiles as separate actions in the AWS CLI and AWS API.

$ aws iam create-instance-profile --instance-profile-name ecsInstanceRole

The following is an example response.

```json
{
    "InstanceProfile": {
        "Path": "/",
        "InstanceProfileName": "ecsInstanceRole",
        "InstanceProfileId": "AIPAT46P5RDITREXAMPLE",
        "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole",
        "CreateDate": "2022-06-30T23:53:34.093Z",
        "Roles": [],
    }
}
```

4. Use the `add-role-to-instance-profile` command to add the `ecsInstanceRole` role to the `ecsInstanceRole` instance profile.

   ```bash
   aws iam add-role-to-instance-profile \
   --role-name ecsInstanceRole --instance-profile-name ecsInstanceRole
   ```

5. Use the `attach-role-policy` command to attach the `AmazonEC2ContainerServiceforEC2Role` AWS managed policy to the `ecsInstanceRole` role.

   ```bash
   $ aws iam attach-role-policy \
   --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role \
   --role-name ecsInstanceRole
   ```
Amazon EC2 spot fleet role

If you create a managed compute environment that uses Amazon EC2 Spot Fleet Instances, you must create the AmazonEC2SpotFleetTaggingRole policy. This policy grants Spot Fleet permission to launch, tag, and terminate instances on your behalf. Specify the role in your Spot Fleet request. You must also have the AWSServiceRoleForEC2Spot and AWSServiceRoleForEC2SpotFleet service-linked roles for Amazon EC2 Spot and Spot Fleet. Use the following instruction to create all of these roles. For more information, see Using Service-Linked Roles and Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

Topics

• Create Amazon EC2 spot fleet roles in the AWS Management Console
• Create Amazon EC2 Spot Fleet Roles with the AWS CLI

Create Amazon EC2 spot fleet roles in the AWS Management Console

To create the AmazonEC2SpotFleetTaggingRole IAM service-linked role for Amazon EC2 Spot Fleet

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. For Access Management, choose Roles,
3. For Roles, choose Create role.
4. From Select trusted entity for Trusted entity type, choose AWS service.
5. For Use cases for other AWS services, choose EC2 and then choose EC2 - Spot Fleet Tagging.
6. Choose Next.
7. From Permissions policies for Policy name, verify AmazonEC2SpotFleetTaggingRole.
8. Choose Next.
9. For Name, review, and create:
   a. For Role name, enter a name to identify the role.
   b. For Description, enter a short explanation for the policy.
   c. (Optional) For Step 1: Select trusted entities, choose Edit to modify the code.
   d. (Optional) For Step 2: Add permissions, choose Edit to modify the code.
   e. (Optional) For Add tags, choose Add tag to add tags to the resource.
f. Choose **Create role**.

---

**Note**

In the past, there were two managed policies for the Amazon EC2 Spot Fleet role.

- **AmazonEC2SpotFleetRole**: This is the original managed policy for the Spot Fleet role. However, we no longer recommend that you use it with AWS Batch. This policy doesn't support Spot Instance tagging in compute environments, which is required to use the AWSServiceRoleForBatch service-linked role. If you previously created a Spot Fleet role with this policy, apply the new recommended policy to that role. For more information, see [Spot Instances not tagged on creation](#).

- **AmazonEC2SpotFleetTaggingRole**: This role provides all of the necessary permissions to tag Amazon EC2 Spot Instances. Use this role to allow Spot Instance tagging on your AWS Batch compute environments.

---

### Create Amazon EC2 Spot Fleet Roles with the AWS CLI

To create the **AmazonEC2SpotFleetTaggingRole** IAM role for your Spot Fleet compute environments

1. Run the following command with the AWS CLI.

```bash
$ aws iam create-role --role-name AmazonEC2SpotFleetTaggingRole --assume-role-policy-document '{
  "Version":"2012-10-17",
  "Statement": [ 
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "spotfleet.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
''
```
2. To attach the `AmazonEC2SpotFleetTaggingRole` managed IAM policy to your `AmazonEC2SpotFleetTaggingRole` role, run the following command with the AWS CLI.

```
$ aws iam attach-role-policy \
  --policy-arn \n  arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \n  --role-name \n  AmazonEC2SpotFleetTaggingRole
```

To create the `AWSServiceRoleForEC2Spot` IAM service-linked role for Amazon EC2 Spot

**Note**

If the `AWSServiceRoleForEC2Spot` IAM service-linked role already exists, you see an error message that resembles the following.

```
An error occurred (InvalidInput) when calling the CreateServiceLinkedRole operation:
Service role name AWSServiceRoleForEC2Spot has been taken in this account, please try a different suffix.
```

- Run the following command with the AWS CLI.

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

To create the `AWSServiceRoleForEC2SpotFleet` IAM service-linked role for Amazon EC2 Spot Fleet

**Note**

If the `AWSServiceRoleForEC2SpotFleet` IAM service-linked role already exists, you see an error message that resembles the following.

```
An error occurred (InvalidInput) when calling the CreateServiceLinkedRole operation:
```
Service role name AWSServiceRoleForEC2SpotFleet has been taken in this account, please try a different suffix.

- Run the following command with the AWS CLI.

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

## EventBridge IAM role

Amazon EventBridge delivers a near-real time stream of system events that describe changes in AWS resources. AWS Batch jobs are available as EventBridge targets. Using simple rules that you can quickly set up, you can match events and submit AWS Batch jobs in response to them. Before you can submit AWS Batch jobs with EventBridge rules and targets, EventBridge must have permissions to run AWS Batch jobs on your behalf.

⚠️ **Note**

When you create a rule in the EventBridge console that specifies an AWS Batch queue as a target, you can create this role. For an example walkthrough, see [AWS Batch jobs as EventBridge targets](#). You can create the EventBridge role manually using the IAM console. For instructions, see [Creating a role using custom trust policies (console)](#) in the IAM User Guide.

The trust relationship for your EventBridge IAM role must provide the events.amazonaws.com service principal the ability to assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [ 
    { 
      "Sid": "",
      "Effect": "Allow",
      "Principal": { 
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```
Make sure that the policy that's attached to your EventBridge IAM role allows `batch:SubmitJob` permissions on your resources. In the following example, AWS Batch provides the `AWSBatchServiceEventTargetRole` managed policy to provide these permissions.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": "*"
        }
    ]
}
```
AWS Batch Event Stream for Amazon EventBridge

You can use the AWS Batch event stream for Amazon EventBridge to receive near real-time notifications regarding the current state of jobs in your job queues.

You can use EventBridge to gain further insights about your AWS Batch service. More specifically, you can use it to check the progress of jobs, build AWS Batch custom workflows, generate usage reports or metrics, or build your own dashboards. With AWS Batch and EventBridge, you don’t need scheduling and monitoring code that continuously polls AWS Batch for job status changes. Instead, you can handle AWS Batch job state changes asynchronously using a variety of Amazon EventBridge targets. These include AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

Events from the AWS Batch event stream are ensured to be delivered at least one time. In the event that duplicate events are sent, the event provides enough information to identify duplicates. That way, you can compare the time stamp of the event and the job status.

AWS Batch jobs are available as EventBridge targets. Using simple rules, you can match events and submit AWS Batch jobs in response to them. For more information, see What is EventBridge? in the Amazon EventBridge User Guide. You can also use EventBridge to schedule automated actions that self-trigger at certain times using cron or rate expressions. For more information, see Creating an Amazon EventBridge rule that runs on a schedule in the Amazon EventBridge User Guide. For an example walkthrough, see AWS Batch jobs as EventBridge targets. For information about using the EventBridge Scheduler, see Setting up Amazon EventBridge Scheduler in the Amazon EventBridge User Guide.

Topics

- AWS Batch Events
- Using AWS User Notifications with AWS Batch
- AWS Batch jobs as EventBridge targets
- Tutorial: Listening for AWS Batch EventBridge
- Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events
AWS Batch Events

AWS Batch sends job status change events to EventBridge. AWS Batch tracks the state of your jobs. If a previously submitted job's status changes, an event is invoked. For example, if a job in the RUNNING status moves to the FAILED status. These events are classified as job state change events.

**Note**

AWS Batch might add other event types, sources, and details in the future. If you're programmatical deserializing event JSON data, make sure that your application is prepared to handle unknown properties. This is to avoid issues if and when these additional properties are added.

Job State Change Events

Anytime that an existing (previously submitted) job changes states, an event is created. For more information about AWS Batch job states, see [Job states](#).

**Note**

Events aren't created for the initial job submission.

Example Job State Change Event

Job state change events are delivered in the following format. The `detail` section resembles the `JobDetail` object that's returned from a [DescribeJobs](#) API operation in the [AWS Batch API Reference](#).

For more information about EventBridge parameters, see [Events and Event Patterns](#) in the [Amazon EventBridge User Guide](#).

```json
{
    "version": "0",
    "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
    "detail-type": "Batch Job State Change",
    "source": "aws.batch",
    "account": "123456789012",
    "time": "2022-01-11T23:36:40Z",
    "region": "us-east-1",
}
```
"resources": [
  "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
],
"detail": {
  "jobArn": "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
  "jobName": "event-test",
  "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
  "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/PexjEHAPPYPATHCANARY2JOBQUEUE",
  "status": "RUNNABLE",
  "attempts": [],
  "createdAt": 1641944200058,
  "retryStrategy": {
    "attempts": 2,
    "evaluateOnExit": []
  },
  "dependsOn": [],
  "parameters": {},
  "container": {
    "image": "137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest",
    "command": [
      "sleep",
      "600"
    ],
    "volumes": [],
    "environment": [],
    "mountPoints": [],
    "ulimits": [],
    "networkInterfaces": [],
    "resourceRequirements": [
      {
        "value": "2",
        "type": "VCPU"
      },
      {
        "value": "256",
        "type": "MEMORY"
      }
    ],
    "secrets": []
  },
  "tags": {"
Job queue blocked events

Anytime that AWS Batch detects a job in the RUNNABLE state and thus blocking a queue, an event is created in Amazon CloudWatch Events. For more information about supported blocked queue causes, see example blocked job queue messages. The same reason is also available in the statusReason field in the DescribeJobs API action.

Example Job State Change Event

Job state change events are delivered in the following format. The detail section resembles the JobDetail object that's returned from a DescribeJobs API operation in the AWS Batch API Reference. For more information about EventBridge parameters, see Events and Event Patterns in the Amazon EventBridge User Guide.

```
{
    "version": "0",
    "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
    "detail-type": "Batch Job Queue Blocked",
    "source": "aws.batch",
    "account": "123456789012",
    "time": "2022-01-11T23:36:40Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
        "arn:aws:batch:us-east-1:123456789012:job-queue/PexjEHappyPathCanary2JobQueue"
    ],
    "detail": {
        "jobArn": "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
        "jobName": "event-test",
        "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
        "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/PexjEHappyPathCanary2JobQueue",
```

"status": "RUNNABLE",
"statusReason": "blocked-reason",
"attempts": [],
"createdAt": 1641944200058,
"retryStrategy": {
  "attempts": 2,
  "evaluateOnExit": []
},
"dependsOn": [],
"parameters": {},
"container": {
  "image": "137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest",
  "command": [
    "sleep",
    "600"
  ],
  "volumes": [],
  "environment": [],
  "mountPoints": [],
  "ulimits": [],
  "networkInterfaces": [],
  "resourceRequirements": [
    {
      "value": "2",
      "type": "VCPU"
    },
    {
      "value": "256",
      "type": "MEMORY"
    }
  ],
  "secrets": []
},
"tags": {
  "resourceArn": "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
},
"propagateTags": false,
"platformCapabilities": []
}
Using AWS User Notifications with AWS Batch

You can use AWS User Notifications to set up delivery channels to get notified about AWS Batch 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.

To configure User Notifications in AWS Batch:

1. Open the AWS Batch console.
2. Choose Dashboard.
3. Choose Configure Notifications.
4. In AWS User Notifications, choose Create notification configuration.

For more information about how to configure and view user notifications, see Getting started with AWS User Notifications.

AWS Batch jobs as EventBridge targets

Amazon EventBridge delivers a near real-time stream of system events that describe changes in Amazon Web Services resources. Typically, AWS Batch on Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, and AWS Fargate jobs are available as EventBridge targets. Using simple rules, you can match events and submit AWS Batch jobs in response to them. For more information, see What is EventBridge? in the Amazon EventBridge User Guide.

You can also use EventBridge to schedule automated actions that are invoked at certain times using cron or rate expressions. For more information, see Creating an Amazon EventBridge rule that runs on a schedule in the Amazon EventBridge User Guide.

For information about how to create a rule that runs when an event matches an event pattern, see Creating Amazon EventBridge rules that react to events in the Amazon EventBridge User Guide.

Common use cases for AWS Batch jobs as an EventBridge target include the following use cases:

- A scheduled job occurs at regular time intervals. For example, a cron job occurs only during low-usage hours when Amazon EC2 Spot Instances are less expensive.
An AWS Batch job runs in response to an API operation that's logged in CloudTrail. For example, a job is submitted whenever an object is uploaded to a specified Amazon S3 bucket. Each time this happens, the EventBridge input transformer passes the bucket and key name of the object to AWS Batch parameters.

**Note**

In this scenario, all of related AWS resources must be in the same Region. This includes resources such as the Amazon S3 bucket, EventBridge rule, and CloudTrail logs.

Before you can submit AWS Batch jobs with EventBridge rules and targets, the EventBridge service requires several permissions to run AWS Batch jobs. When you create a rule in the EventBridge console that specifies an AWS Batch job as a target, you can also create this role. For more information about the required service principal and IAM permissions for this role, see EventBridge IAM role.

**Creating a scheduled AWS Batch job**

The following procedure covers how to create a scheduled AWS Batch job and the required EventBridge IAM role.

**To create a scheduled AWS Batch job with EventBridge**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose **Rules**.
4. Choose **Create rule**.
5. For **Name**, specify a unique name for your compute environment. The name can contain up to 64 characters. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
6. (Optional) For **Description**, enter a description for the rule.

7. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **default**. When an AWS service in your account emits an event, it always goes to your account's default event bus.

8. (Optional) Turn off the rule on the selected bus if you don't want to run the rule immediately.

9. For **Rule type**, choose **Schedule**.

10. Choose **Continue to create rule** or **Next**.

11. For **Schedule pattern**, do one of the following:

   - Choose **A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month** and then enter a cron expression. For more information, see Cron Expressions in the Amazon EventBridge User Guide.
   - Choose **A schedule that runs at a regular rate, such as every 10 minutes.** and then enter a rate expression.

12. Choose **Next**.

13. For **Target types**, choose **AWS service**.

14. For **Select a target**, choose **Batch job queue**. Then, configure the following:

   - **Job queue**: Enter the Amazon Resource Name (ARN) of the job queue to schedule your job in.
   - **Job definition**: Enter the name and revision or full ARN of the job definition to use for your job.
   - **Job name**: Enter a name for your job.
   - **Array size**: (Optional) Enter an array size for your job to run more than one copy. For more information, see Array jobs.
   - **Job attempts**: (Optional) Enter the number of times to retry your job if it fails. For more information, see Automated job retries.

15. For **Batch job queue** target types, EventBridge needs permission to send events to the target. EventBridge can create the IAM role needed for your rule to run. Do one of the following:
• To create an IAM role automatically, choose Create a new role for this specific resource.
• To use an IAM role that you've already created, choose Use existing role.

16. (Optional) Expand Additional settings.

   a. For Configure target input, choose how the text from an event is processed before it's passed to the target.
   b. For Maximum age of event, specify the time interval for how long unprocessed events are kept.
   c. For Retry attempts, enter the number of times that an event is retried.
   d. For Dead-letter queue, choose an option for how unprocessed events are handled. If necessary, specify the Amazon SQS queue to use as the dead-letter queue.

17. (Optional) Choose Add another target to add another target for this rule.

18. Choose Next.

19. (Optional) For Tags, choose Add new tag to add a resource label for the rule. For more information, see Amazon EventBridge tags.

20. Choose Next.

21. For Review and create, review the configuration steps. If you need to make changes, choose Edit. When you're finished, choose Create rule.

For more information about creating rules, see Creating an Amazon EventBridge rule that runs on a schedule in the Amazon EventBridge User Guide.

Creating a rule with an event pattern

The following procedure covers how to create a rule with an event pattern.

To create a rule that sends the event to a target when the event matches a defined pattern

Note

This procedure works for all AWS Batch on Amazon ECS, Amazon EKS, and AWS Fargate jobs.

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. From the navigation bar, select the AWS Region to use.

3. In the navigation pane, choose Rules.

4. Choose Create rule.

5. For Name, specify a unique name for your compute environment. The name can contain up to 64 characters. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

   ![Note]
   
   A rule can't have the same name as another rule in the same Region and on the same event bus.

6. (Optional) For Description, enter a description for the rule.

7. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select default. When an AWS service in your account emits an event, it always goes to your account's default event bus.

8. (Optional) Turn off the rule on the selected bus if you don't want to run the rule immediately.

9. For Rule type, choose Rule with an event pattern.

10. Choose Next.

11. For Event Source, choose AWS event or EventBridge partner events.

12. (Optional) For Sample event:

   a. For Sample event type, choose AWS events.
   b. For Sample events, choose Batch Job State Change.

13. For Creation method, choose Use pattern form.

14. For Event pattern:

   a. For Event source, choose AWS services.
   b. For AWS service, choose Batch.
   c. For Event type, choose Batch Job State Change.

15. Choose Next.

16. For Target types, choose AWS service.

17. For Select a target, choose a target type. For example, choose Batch job queue. Then specify the following:
• **Job queue:** Enter the Amazon Resource Name (ARN) of the job queue to schedule your job in.

• **Job definition:** Enter the name and revision or full ARN of the job definition to use for your job.

• **Job name:** Enter a name for your job.

• **Array size:** (Optional) Enter an array size for your job to run more than one copy. For more information, see [Array jobs](#).

• **Job attempts:** (Optional) Enter the number of times to retry your job if it fails. For more information, see [Automated job retries](#).

18. For **Batch job queue** target types, EventBridge needs permission to send events to the target. EventBridge can create the IAM role needed for your rule to run. Do one of the following:

   • To create an IAM role automatically, choose **Create a new role for this specific resource**.

   • To use an IAM role that you created before, choose **Use existing role**.

19. (Optional) Expand **Additional settings**.

   a. For **Configure target input**, choose how text from an event is processed.

   b. For **Maximum age of event**, specify the time interval for how long unprocessed events are kept.

   c. For **Retry attempts**, enter the number of times that an event is retried.

   d. For **Dead-letter queue**, choose an option for how unprocessed events are handled. If necessary, specify the Amazon SQS queue to use as the dead-letter queue.

20. (Optional) Choose **Add another target** to add an additional target.

21. Choose **Next**.

22. (Optional) For **Tags**, choose **Add new tag** to add a resource label. For more information, see [Amazon EventBridge tags](#) in the **Amazon EventBridge User Guide**.

23. Choose **Next**.

24. For **Review and create**, review the configuration steps. If you need to make changes, choose **Edit**. After you're finished, choose **Create rule**.

   For more information about creating rules, see [Creating Amazon EventBridge rules that react to events](#) in the **Amazon EventBridge User Guide**.
Passing event information to an AWS Batch Target on a schedule using the EventBridge input transformer

You can use the EventBridge input transformer to pass event information to AWS Batch in a job submission. This can be especially valuable if you invoke jobs as a result of other AWS event information. One example is an object upload to an Amazon S3 bucket. You can also use a job definition with parameter substitution values in the container's command. The EventBridge input transformer can provide the parameter values based on the event data.

Then, afterwards, you create an AWS Batch event target that parses information from the event that starts it and transforms it into a parameters object. When the job runs, the parameters from the trigger event are passed to the command of the job container.

Note
In this scenario, all of the AWS resources (such as Amazon S3 buckets, EventBridge rules, and CloudTrail logs) must be in the same Region.

To create an AWS Batch target that uses the input transformer

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Rules.
4. Choose Create rule.
5. For Name, specify a unique name for your compute environment. The name can contain up to 64 characters. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).

   Note
   A rule can't have the same name as another rule in the same AWS Region and on the same event bus.

6. (Optional) For Description, enter a description for the rule.
7. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **default**. When an AWS service in your account emits an event, it always goes to your account's default event bus.

8. (Optional) Turn off the rule on the selected bus if you don't want to run the rule immediately.

9. For **Rule type**, choose **Schedule**.

10. Choose **Continue to create rule** or **Next**.

11. For **Schedule pattern**, do one of the following:

    - Choose **A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month** and then enter a cron expression. For more information, see **Cron Expressions** in the **Amazon EventBridge User Guide**.
    - Choose **A schedule that runs at a regular rate, such as every 10 minutes**. and then enter a rate expression.

12. Choose **Next**.

13. For **Target types**, choose **AWS service**.

14. For **Select a target**, choose **Batch job queue**. Then, configure the following:

    - **Job queue**: Enter the Amazon Resource Name (ARN) of the job queue to schedule your job in.
    - **Job definition**: Enter the name and revision or full ARN of the job definition to use for your job.
    - **Job name**: Enter a name for your job.
    - **Array size**: (Optional) Enter an array size for your job to run more than one copy. For more information, see **Array jobs**.
    - **Job attempts**: (Optional) Enter the number of times to retry your job if it fails. For more information, see **Automated job retries**.

15. For **Batch job queue** target types, EventBridge needs permission to send events to the target. EventBridge can create the IAM role needed for your rule to run. Do one of the following:

    - To create an IAM role automatically, choose **Create a new role for this specific resource**.
    - To use an IAM role that you've already created, choose **Use existing role**.

16. (Optional) Expand **Additional settings**.

17. In the **Additional settings** section, for **Configure target input**, choose **Input Transformer**.

18. Choose **Configure input transformer**.
19. (Optional) For **Sample event**:
   
   a. For **Sample event type**, choose **AWS events**.
   
   b. For **Sample events**, choose **Batch Job State Change**.

20. In the **Target input transformer** section, for **Input path**, specify the values to parse from the triggering event. For example, to parse **Batch Job State Change** event, use the following JSON format.

   ```json
   {
      "instance": ".detail.jobId",
      "state": ".detail.status"
   }
   ```

21. For **Template**, enter the following.

   ```json
   {
      "instance": <jobId> ,
      "status": <status>
   }
   ```

22. Choose **Confirm**.

23. For **Maximum age of event**, specify the time interval for how long unprocessed events are kept.

24. For **Retry attempts**, enter the number of times that an event is retried.

25. For **Dead-letter queue**, choose an option for how unprocessed events are handled. If necessary, specify the Amazon SQS queue to use as the dead-letter queue.

26. (Optional) Choose **Add another target** to add an additional target.

27. Choose **Next**.

28. (Optional) For **Tags**, choose **Add new tag** to add a resource label. For more information, see [Amazon EventBridge tags](#) in the **Amazon EventBridge User Guide**.

29. Choose **Next**.

30. For **Review and create**, review the configuration steps. If you need to make changes, choose **Edit**. After you're finished, choose **Create rule**.
Tutorial: Listening for AWS Batch EventBridge

In this tutorial, you set up a simple AWS Lambda function that listens for AWS Batch job events and writes them out to a CloudWatch Logs log stream.

Prerequisites

This tutorial assumes that you have a working compute environment and job queue that are ready to accept jobs. If you don't have a running compute environment and job queue to capture events from, follow the steps in Getting Started with AWS Batch to create one. At the end of this tutorial, you can optionally submit a job to this job queue 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 AWS Batch event stream messages.

To create a target Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function, Author from scratch.
3. For Function name, enter batch-event-stream-handler.
4. For Runtime, choose Python 3.8.
5. Choose Create function.
6. In the Code source section, edit the sample code to match the following example:

```python
import json

def lambda_handler(event, _context):
    # _context is not used
    del _context
    if event['source'] != "aws.batch":
        raise ValueError("Function only supports input from events with a source type of: aws.batch")
    print(json.dumps(event))
```
This is a simple Python 3.8 function that prints the events sent by AWS Batch. If everything is configured correctly, at the end of this tutorial, the event details appear in the CloudWatch Logs log stream that's associated with this Lambda function.

7. Choose Deploy.

**Step 2: Register Event Rule**

In this section, you create an EventBridge event rule that captures job events that are coming from your AWS Batch resources. This rule captures all events coming from AWS Batch within the account where it's defined. The job messages themselves contain information about the event source, including the job queue where it was submitted. You can use this information to filter and sort events programmatically.

**Note**

If you use the AWS Management Console to create an event rule, the console automatically adds the IAM permissions for EventBridge to call your Lambda function. However, if you're creating an event rule using the AWS CLI, you must grant permissions explicitly. For more information, see [Events and Event Patterns](https://docs.aws.amazon.com/AmazonEventBridge/latest/userguide/what-is-events.html) in the *Amazon EventBridge User Guide*.

**To create your EventBridge rule**

1. Open the Amazon EventBridge console at [https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/).
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule.
   
   A rule can't have the same name as another rule in the same Region and on the same event bus.
5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **AWS default event bus**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For **Event source**, choose **Other**.
9. For **Event pattern**, select **Custom patterns (JSON editor)**.
10. Paste the following event pattern into the text area.

    ```json
    {
        "source": [
            "aws.batch"
        ]
    }
    ```

    This rule applies across all of your AWS Batch groups and to every AWS Batch event. Alternatively, you can create a more specific rule to filter out some results.

11. Choose **Next**.
12. For **Target types**, choose **AWS service**.
13. For **Select a target**, choose **Lambda function**, and select your Lambda function.
14. (Optional) For **Additional settings**, do the following:

    a. For **Maximum age of event**, enter a value between one minute (00:01) and 24 hours (24:00).
    b. For **Retry attempts**, enter a number between 0 and 185.
    c. For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
       - Choose **None** to not use a dead-letter queue.
       - Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the dropdown.
       - Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it. For more information, see [Granting permissions to the dead-letter queue](#) in the *Amazon EventBridge User Guide*.

15. Choose **Next**.
16. (Optional) Enter one or more tags for the rule. For more information, see [Amazon EventBridge tags](#) in the *Amazon EventBridge User Guide*.
17. Choose **Next**.
18. Review the details of the rule and choose **Create rule**.

**Step 3: Test Your Configuration**

You can now test your EventBridge configuration by submitting a job to your job queue. If everything is configured properly, your Lambda function is triggered and it writes the event data to a CloudWatch Logs log stream for the function.

**To test your configuration**

2. Submit a new AWS Batch job. For more information, see **Submitting a job**.
4. On the navigation pane, choose **Logs** and select the log group for your Lambda function (for example, `/aws/lambda/my-function`).
5. Select a log stream to view the event data.

**Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events**

In this tutorial, you configure an EventBridge event rule that only captures job events where the job has moved to a **FAILED** status. At the end of this tutorial, you can optionally also submit a job to this job queue. This is to test that you have configured your Amazon SNS alerts correctly.

**Prerequisites**

This tutorial assumes that you have a working compute environment and job queue that are ready to accept jobs. If you don't have a running compute environment and job queue to capture events from, follow the steps in **Getting Started with AWS Batch** to create one.

**Step 1: Create and Subscribe to an Amazon SNS Topic**

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.
To create an Amazon SNS topic

2. Choose Topics, Create topic.
3. For Type, choose Standard.
4. For Name, enter JobFailedAlert and choose Create topic.
5. On the JobFailedAlert screen, choose Create subscription.
6. For Protocol, choose Email.
7. For Endpoint, enter an email address that you currently have access to and choose Create subscription.
8. Check your email account, and wait to receive a subscription confirmation email message. When you receive it, choose Confirm subscription.

Step 2: Register Event Rule

Next, register an event rule that captures only job-failed events.

To register your EventBridge rule

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose Rules.
3. Choose Create rule.
4. Enter a name and description for the rule.

A rule can't have the same name as another rule in the same Region and on the same event bus.

5. For Event bus, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select AWS default event bus. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. For Rule type, choose Rule with an event pattern.
7. Choose Next.
8. For Event source, choose Other.
9. For Event pattern, select Custom patterns (JSON editor).
10. Paste the following event pattern into the text area.
This code defines an EventBridge rule that matches any event where the job status is FAILED. For more information about event patterns, see Events and Event Patterns in the Amazon EventBridge User Guide.

11. Choose Next.

12. For Target types, choose AWS service.

13. For Select a target, choose SNS topic, and for Topic, choose JobFailedAlert.

14. (Optional) For Additional settings, do the following:

   a. For Maximum age of event, enter a value between one minute (00:01) and 24 hours (24:00).

   b. For Retry attempts, enter a number between 0 and 185.

   c. For Dead-letter queue, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:

      • Choose None to not use a dead-letter queue.

      • Choose Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue and then select the queue to use from the dropdown.

      • Choose Select an Amazon SQS queue in an other AWS account as a dead-letter queue and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it. For more information, see Granting permissions to the dead-letter queue in the Amazon EventBridge User Guide.
15. Choose **Next**.

16. (Optional) Enter one or more tags for the rule. For more information, see [Amazon EventBridge tags](https://docs.aws.amazon.com/eventbridge/latest/userguide/events-events.html) in the *Amazon EventBridge User Guide*.

17. Choose **Next**.

18. Review the details of the rule and choose **Create rule**.

### Step 3: Test Your Rule

To test your rule, submit a job that exits shortly after it starts with a non-zero exit code. If your event rule is configured correctly, you should receive an email message within a few minutes with the event text.

**To test a rule**


2. Submit a new AWS Batch job. For more information, see [Submitting a job](https://docs.aws.amazon.com/batch/latest/userguide/submit-job.html). For the job's command, substitute this command to exit the container with an exit code of 1.

   ```bash
   /bin/sh, -c, 'exit 1'
   ```

3. Check your email to confirm that you received an email alert for the failed job notification.

### Alternate rule: Batch Job Queue Blocked

To create an event rule that monitors for *Batch Job Queue Blocked*, repeat the steps in this tutorial with the following alterations:

1. **In Step 1**, use `BlockedJobQueue` as the topic name.

2. **In Step 2**, use the following pattern in the JSON editor:

   ```json
   {
     "detail-type": [
       "Batch Job Queue Blocked"
     ],
     "source": [
       "aws.batch"
     ]
   }
   ```
Alternate rule: Batch Job Queue Blocked
Using CloudWatch Logs with AWS Batch

You can configure your AWS Batch jobs on EC2 resources to send detailed log information and metrics to CloudWatch Logs. Doing this, you can view different logs from your jobs in one convenient location. For more information about CloudWatch Logs, see [What is Amazon CloudWatch Logs?](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/) in the *Amazon CloudWatch User Guide*.

**Note**  
By default, CloudWatch Logs is turned on for AWS Fargate containers.

To turn on and customize CloudWatch Logs logging, review the following one-time configuration tasks:

- For AWS Batch compute environments that are based on EC2 resources, add an IAM policy to the *ecsInstanceRole* role. For more information, see [the section called “Add a CloudWatch Logs IAM policy”](https).

- Create an Amazon EC2 launch template that includes detailed CloudWatch monitoring, then specify the template when you create your AWS Batch compute environment. You can also install the CloudWatch agent on an existing image and then specify the image in the AWS Batch first-run wizard.

- (Optional) Configure the *awslogs* driver. You can add parameters that change the default behavior on both EC2 and Fargate resources. For more information, see [the section called “Using the *awslogs* log driver”](https).

**Add a CloudWatch Logs IAM policy**

Before your jobs can send log data and detailed metrics to CloudWatch Logs, you must create an IAM policy that uses the CloudWatch Logs APIs. After you create the IAM policy, attach it to the *ecsInstanceRole* role.
AWS Batch compute environments use Amazon EC2 resources. When you create a compute environment using the AWS Batch first-run wizard, AWS Batch creates the `ecsInstanceRole` role and configures the environment with it.

If you aren't using the first-run wizard, you can specify the `ecsInstanceRole` role when you create a compute environment in the AWS Command Line Interface or AWS Batch API. For more information, see the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/batch.html) or [AWS Batch API Reference](https://docs.aws.amazon.com/batch/latest/api/).

**To create the ECS-CloudWatchLogs IAM policy**

2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. Choose **JSON**, then enter the following policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": ["arn:aws:logs:*:":*"],
        }
    ]
}
```

**Add a CloudWatch Logs IAM policy**
5. Choose **Next: Tags**.

6. (Optional) For **Add tags**, choose **Add tag** to add a tag to the policy.

7. Choose **Next: Review**.

8. On the **Review policy** page, for **Name**, enter **ECS-CloudWatchLogs**, and then enter an optional **Description**.

9. Choose **Create policy**.

---

**To attach the ECS-CloudWatchLogs policy to ecsInstanceRole**


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

3. Choose **ecsInstanceRole**. If the role doesn't exist, follow the procedures in [Amazon ECS instance role](https://console.aws.amazon.com/iam/) to create the role.

4. Choose **Add Permissions**, then choose **Attach policies**.

5. Choose the **ECS-CloudWatchLogs** policy and then choose **Attach policy**.

---

**Install and configure the CloudWatch agent**

You can create an Amazon EC2 launch template that includes CloudWatch monitoring. For more information, see [Launch an instance from a launch template](https://console.aws.amazon.com/iam/) and [Advanced details](https://console.aws.amazon.com/iam/) in the [Amazon EC2 User Guide for Linux Instances](https://console.aws.amazon.com/iam/).

You can also install the CloudWatch agent on an existing Amazon EC2 AMI and then specify the image in the AWS Batch first-run wizard. For more information, see [Installing the CloudWatch agent](https://console.aws.amazon.com/iam/) and [Getting Started with AWS Batch](https://console.aws.amazon.com/iam/).

---

**Note**

Launch templates are not supported on AWS Fargate resources.

---

**View CloudWatch Logs**

You can view and search CloudWatch Logs logs in the AWS Management Console.
Note
It might take a few minutes for data to display in CloudWatch Logs.

To view your CloudWatch Logs data

2. In the left navigation pane, choose Logs, then choose Log groups.
3. Choose a log group to view.
4. Choose a log stream to view. By default, the streams are identified by the first 200 characters of the job name and the Amazon ECS task ID.
Tip
To download log stream data, choose Actions.
Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs

You can use Amazon CloudWatch Logs to monitor, store, and view all your log files in one location. Using CloudWatch Logs, you can search, filter, and analyze log data from multiple sources.

You can download an AWS for Fluent Bit image that includes a plugin to monitor AWS Batch on Amazon EKS jobs in CloudWatch Logs. Fluent Bit is an open-source log processor and forwarder that's both Docker and Kubernetes compatible. We recommend that you use Fluent Bit as your log router because it's less resource intensive than Fluentd. For more information, see Using the AWS for Fluent Bit image.

Prerequisites

Attach the CloudWatchAgentServerPolicy policy to the AWS Identity and Access Management policy of your worker nodes. For more information, see Verify prerequisites.

Install AWS for Fluent Bit

For instructions on how to install AWS for Fluent Bit and create the CloudWatch groups, see Setting up Fluent Bit or Quick Start with the CloudWatch agent and Fluent Bit.

Tip

Remember that Fluent Bit uses .5 CPU and 100 MB of memory on AWS Batch nodes. This reduces the total available capacity for AWS Batch jobs. Consider this when you size your jobs.

Turn on Fluent Bit for AWS Batch nodes

To ensure the Fluent Bit logging DaemonSet runs on AWS Batch managed nodes, modify the Fluent Bit DaemonSet tolerations:

tolerations:
- key: "batch.amazonaws.com/batch-node"
operator: "Exists"
AWS Batch CloudWatch Container Insights

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your AWS Batch compute environments and jobs. The metrics include CPU, memory, disk, and network utilization. You can add these metrics to CloudWatch dashboards.

Operational data is collected as performance log events. These are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates higher-level aggregated metrics at the compute environment and job level as CloudWatch metrics. For more information, see Container Insights Structured Logs for Amazon ECS in the Amazon CloudWatch User Guide.

⚠ Important
CloudWatch Container Insights are charged as custom metrics by CloudWatch. For more information, see Amazon CloudWatch Events pricing.

Turn on Container Insights

You can turn on Container Insights for AWS Batch compute environments.

1. Open the AWS Batch console.
2. Choose Compute Environments.
3. Choose the compute environment that you want.
4. For Container Insights, turn on Container Insights for the compute environment.

⚠ Tip
You can select a default interval to aggregate the metrics or create a custom interval.

By default, the following metrics are displayed. For a full list of Amazon ECS Container Insights metrics, see Amazon ECS Container Insights Metrics in the Amazon CloudWatch User Guide.
• **JobCount** – The number of jobs that run in the compute environment.

• **ContainerInstanceCount** – The number of Amazon Elastic Compute Cloud instances that run the Amazon ECS agent and are registered in the compute environment.

• **MemoryReserved** – The memory that's reserved by compute environment jobs. This metric is collected only for the jobs that have a defined memory reservation in their job definition.

• **MemoryUtilized** – The memory that's being used by compute environment jobs. This metric is collected only for jobs that have a defined memory reservation in their job definition.

• **CpuReserved** – The CPU units that are reserved by compute environment jobs. This metric is collected only for jobs that have a defined CPU reservation in their job definition.

• **CpuUtilized** – The CPU units used by jobs in the compute environment. This metric is collected only for jobs that have a defined CPU reservation in their job definition.

• **NetworkRxBytes** - The number of bytes that are received. This metric is available only for containers in jobs that use the `awsvpc` or bridge network modes.

• **NetworkTxBytes** – The number of bytes that are transmitted. This metric is available only for containers in jobs that use the `awsvpc` or bridge network modes.

• **StorageReadBytes** – The number of bytes that are read from storage.

• **StorageWriteBytes** – The number of bytes that are written to storage.
Logging AWS Batch API Calls with AWS CloudTrail

AWS Batch is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Batch. CloudTrail captures all API calls for AWS Batch as events. The calls captured include calls from the AWS Batch console and code calls to the AWS Batch API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Batch. 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 AWS Batch, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS Batch Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Batch, 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 AWS Batch, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS 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 following:

- 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 AWS Batch actions are logged by CloudTrail and are documented in the https://docs.aws.amazon.com/batch/latest/APIReference/. For example, calls to the SubmitJob, ListJobs and DescribeJobs sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding AWS Batch Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the CreateComputeEnvironment action.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
        "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
        "accountId": "012345678910",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2017-12-20T00:48:46Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
```
"arn": "arn:aws:iam::012345678910:role/Admin",
"accountId": "012345678910",
"userName": "Admin"
}
}
,"eventTime": "2017-12-20T00:48:46Z",
"eventSource": "batch.amazonaws.com",
"eventName": "CreateComputeEnvironment",
"awsRegion": "us-east-1",
"sourceIPAddress": "203.0.113.1",
"userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
"requestParameters": {
  "computeResources": {
    "subnets": [
      "subnet-5eda8e04"
    ],
    "tags": {
      "testBatchTags": "CLI testing CE"
    },
    "desiredvCpus": 0,
    "minvCpus": 0,
    "instanceTypes": [
      "optimal"
    ],
    "securityGroupIds": [
      "sg-aba9e8db"
    ],
    "instanceRole": "ecsInstanceRole",
    "maxvCpus": 128,
    "type": "EC2"
  },
  "state": "ENABLED",
  "type": "MANAGED",
  "computeEnvironmentName": "Test"
},
"responseElements": {
  "computeEnvironmentName": "Test",
  "computeEnvironmentArn": "arn:aws:batch:us-east-1:012345678910:compute-environment/Test"
},
"requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
"eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
"readOnly": false,
"eventType": "AwsApiCall",
"recipientAccountId": "012345678910"
}
Creating a virtual private cloud

Compute resources in your compute environments need external network access to communicate with AWS Batch and Amazon ECS service endpoints. However, you might have jobs that you want to run in private subnets. To have the flexibility to run jobs in either a public or private subnet, create a VPC that has both public and private subnets.

You can use Amazon Virtual Private Cloud (Amazon VPC) to launch AWS resources into a virtual network that you define. This topic provides a link to the Amazon VPC wizard and a list of the options to select.

Create a VPC

For information about how to create an Amazon VPC, see Create a VPC only in the Amazon VPC User Guide and use the following table to determine what options to select.

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

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

After you have created your VPC, consider the following next steps:

- Create security groups for your public and private resources if they require inbound network access. For more information, see [Work with security groups](#) in the *Amazon VPC User Guide*.

- Create an AWS Batch managed compute environment that launches compute resources into your new VPC. For more information, see [Creating a compute environment](#). If you use the compute environment creation wizard in the AWS Batch console, you can specify the VPC that you just created and the public or private subnets that you want to launch your instances into.

- Create an AWS Batch job queue that's mapped to your new compute environment. For more information, see [Creating a job queue](#).

- Create a job definition to run your jobs with. For more information, see [Creating a single-node job definition](#).

- Submit a job with your job definition to your new job queue. This job lands in the compute environment that you created with your new VPC and subnets. For more information, see [Submitting a job](#).
Security in AWS Batch

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 AWS Batch, 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 AWS Batch. The following topics show you how to configure AWS Batch to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Batch resources.

**Topics**

- [Identity and Access Management for AWS Batch](#)
- [Access AWS Batch using an interface endpoint](#)
- [Compliance Validation for AWS Batch](#)
- [Infrastructure Security in AWS Batch](#)

**Identity and Access Management for AWS Batch**

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 AWS Batch resources. IAM is an AWS service that you can use with no additional charge.

**Topics**

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS Batch works with IAM
- AWS Batch execution IAM role
- Identity-based policy examples for AWS Batch
- Cross-service confused deputy prevention
- Troubleshooting AWS Batch identity and access
- Using service-linked roles for AWS Batch
- AWS managed policies for AWS Batch

**Audience**

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Batch.

**Service user** – If you use the AWS Batch service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Batch 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 AWS Batch, see [Troubleshooting AWS Batch identity and access](#).

**Service administrator** – If you're in charge of AWS Batch resources at your company, you probably have full access to AWS Batch. It's your job to determine which AWS Batch 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 AWS Batch, see [How AWS Batch works with IAM](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Batch. To view example AWS Batch identity-based policies that you can use in IAM, see [Identity-based policy examples for AWS Batch](#).
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 (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 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 IAM User 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. 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 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 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.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **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 AWS Batch works with IAM

Before you use IAM to manage access to AWS Batch, learn what IAM features are available to use with AWS Batch.
IAM features you can use with AWS Batch

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>AWS Batch support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource-based policies</td>
<td>No</td>
</tr>
<tr>
<td>Policy actions</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy resources</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy condition keys</td>
<td>Yes</td>
</tr>
<tr>
<td>ACLs</td>
<td>No</td>
</tr>
<tr>
<td>ABAC (tags in policies)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>Principal permissions</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles</td>
<td>Yes</td>
</tr>
</tbody>
</table>

To get a high-level view of how AWS Batch 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 AWS Batch

| 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 AWS Batch

To view examples of AWS Batch identity-based policies, see Identity-based policy examples for AWS Batch.

Policy actions for AWS Batch

<table>
<thead>
<tr>
<th>Supports policy actions</th>
<th>Yes</th>
</tr>
</thead>
</table>

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 AWS Batch actions, see Actions Defined by AWS Batch in the Service Authorization Reference.

Policy actions in AWS Batch use the following prefix before the action:

```
batch
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
    "batch:action1",
```
You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action:

```
"Action": "batch:Describe*"
```

To view examples of AWS Batch identity-based policies, see [Identity-based policy examples for AWS Batch](#).

**Policy resources for AWS Batch**

<table>
<thead>
<tr>
<th>Supports policy resources</th>
<th>Yes</th>
</tr>
</thead>
</table>

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)](https://aws.amazon.com/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 AWS Batch resource types and their ARNs, see [Resources Defined by AWS Batch](#) in the [Service Authorization Reference](#). To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS Batch](#).

To view examples of AWS Batch identity-based policies, see [Identity-based policy examples for AWS Batch](#).
**Policy condition keys for AWS Batch**

| 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/id-policy-template-variables.html) 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/id-policy-template-context-variables.html) in the *IAM User Guide*.

To see a list of AWS Batch condition keys, see [Condition Keys for AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/service-authorization.html#condition-keys) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions Defined by AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/service-authorization.html#actions-defined-by-a-service).

To view examples of AWS Batch identity-based policies, see [Identity-based policy examples for AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/service-authorization.html#identity-based-policy-examples).

### Attribute-based access control (ABAC) with AWS Batch

| 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**.


### Using Temporary credentials with AWS Batch

| 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/what-is-iam.html) 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)](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial-switching-roles.html) 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html).
Cross-service principal permissions for AWS Batch

| Supports forward access sessions (FAS) | Yes |

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS Batch

| 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 AWS Batch functionality. Edit service roles only when AWS Batch provides guidance to do so.

Service-linked roles for AWS Batch

| 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 service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the Service-linked role column. Choose the Yes link to view the service-linked role documentation for that service.

**AWS Batch execution IAM role**

The execution role grants the Amazon ECS container and AWS Fargate agents permission to make AWS API calls on your behalf.

![Note]
The execution role is supported by Amazon ECS container agent version 1.16.0 and later.

The execution IAM role is required depending on the requirements of your task. You can have multiple execution roles for different purposes and services associated with your account.

![Note]
For information about the Amazon ECS instance role, see Amazon ECS instance role. For information about service roles, see How AWS Batch works with IAM.

Amazon ECS provides the AmazonECSTaskExecutionRolePolicy managed policy. This policy contains the required permissions for the common use cases described above. It might be necessary to add inline policies to your execution role for the special use cases outlined below.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ]
        }
    ]
}
```
You can use the following procedure to check that your account already has the execution role and to attach the managed IAM policy, if needed.

**To check for the `ecsTaskExecutionRole` in the IAM console**

2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsTaskExecutionRole`. If you can't find the role, see [Creating the execution IAM role](#). If you found the role, choose the role to view the attached policies.
4. On the **Permissions** tab, verify that the `AmazonECSTaskExecutionRolePolicy` managed policy is attached to the role. If the policy is attached, your execution role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Add permissions**, then choose **Attach policies**.
   b. Search for `AmazonECSTaskExecutionRolePolicy`.
   c. Check the box to the left of the `AmazonECSTaskExecutionRolePolicy` policy and choose **Attach policies**.
5. Choose **Trust relationships**.
6. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, the role is configured correctly. If the trust relationship does not match, choose **Edit trust policy**, enter the following, and choose **Update policy**.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```
Creating the execution IAM role

If your account doesn't already have an execution role, use the following steps to create the role.

To create the ecsTaskExecutionRole IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Choose Create role.
4. For Trusted entity type, choose AWS service.
5. For Service or use case, choose EC2. Then choose EC2 again.
6. Choose Next.
7. For Permissions policies, search for AmazonECSTaskExecutionRolePolicy.
8. Choose the check box to the left of the AmazonECSTaskExecutionRolePolicy policy, and then choose Next.
9. For Role Name, enter ecsTaskExecutionRole and then choose Create role.

Identity-based policy examples for AWS Batch

By default, users and roles don't have permission to create or modify AWS Batch 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 AWS Batch, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS Batch in the Service Authorization Reference.

Topics
Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Batch 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](https://docs.aws.amazon.com/IAM/latest/userguide/using-policies-template-variables.html) or [AWS managed policies for job functions](https://docs.aws.amazon.com/batch/latest/userguide/using-security-policies.html) 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](https://docs.aws.amazon.com/IAM/latest/userguide/policies-and-permissions.html) 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](https://docs.aws.amazon.com/IAM/latest/userguide/policies-common-elements.html) 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](https://docs.aws.amazon.com/IAM/latest/userguide/policies-validation.html) 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](https://docs.aws.amazon.com/IAM/latest/userguide/using-mfa-iam.html) in the *IAM User Guide*.
For more information about best practices in IAM, see Security best practices in IAM in the IAM User Guide.

Using the AWS Batch console

To access the AWS Batch console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Batch resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS Batch console, also attach the AWS Batch ConsoleAccess or ReadOnly AWS managed policy to the entities. For more information, see Adding permissions to a user 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",
```
The confused deputy problem is a security issue where an entity that doesn’t have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to use its permissions to act on another customer’s resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Batch gives another service to the resource. If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the resource that AWS Batch stores.
The most effective way to protect against the confused deputy problem is to use the aws:SourceArn global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the aws:SourceArn global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, arn:aws:service*:123456789012:*

The following examples shows how you can use the aws:SourceArn and aws:SourceAccount global condition context keys in AWS Batch to prevent the confused deputy problem.

**Example 1: Role for accessing only one compute environment**

The following role can only be used to access one compute environment. The job name must be specified as * because the job queue can be associated with multiple compute environments.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "batch.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:batch:us-east-1:123456789012:job/**
          ]
        }
      }
    }
  ]
}
```

**Example 2: Role for accessing multiple compute environments**

The following role can be used to access multiple compute environments. The job name must be specified as * because the job queue can be associated with multiple compute environments.
Troubleshooting AWS Batch identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Batch and IAM.

Topics

- I am not authorized to perform an action in AWS Batch
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS Batch resources

I am not authorized to perform an action in AWS Batch

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.
The following example error occurs when the mateojackson user tries to use the console to view details about a fictional my-example-widget resource but does not have the fictional batch:GetWidget permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
batch:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the my-example-widget resource using the batch:GetWidget action. For more information about granting permissions to pass a role, see Granting a user permissions to pass a role to an AWS service.

**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 AWS Batch.

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 AWS Batch. 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 AWS Batch 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 AWS Batch supports these features, see How AWS Batch works with IAM.
- 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.

Using service-linked roles for AWS Batch

AWS Batch 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 AWS Batch. Service-linked roles are predefined by AWS Batch and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS Batch easier because you don’t have to manually add the necessary permissions. AWS Batch defines the permissions of its service-linked roles, and unless defined otherwise, only AWS Batch 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.

>Note

Do one of the following to specify a service role for an AWS Batch compute environment.

- Use an empty string for the service role. This lets AWS Batch create the service role.
- Specify the service role in the following format:

  `arn:aws:iam::account_number:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch`

For more information, see the section called “Incorrect role name or ARN” in the AWS Batch User Guide.
You can delete a service-linked role only after first deleting their related resources. This protects your AWS Batch resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.

**Service-linked role permissions for AWS Batch**

AWS Batch uses the service-linked role named **AWSServiceRoleForBatch**. The **AWSServiceRoleForBatch** role allows AWS Batch to create and manage AWS resources on your behalf.

The **AWSServiceRoleForBatch** service-linked role trusts the batch.amazonaws.com service principal to assume the role.

The IAM policy named **BatchServiceRolePolicy** allows AWS Batch to complete the following actions on specific resources:

- **autoscaling** – Allows AWS Batch to create and manage Amazon EC2 Auto Scaling resources. AWS Batch creates and manages Amazon EC2 Auto Scaling groups for most compute environments.

- **ec2** – Allows AWS Batch to control the lifecycle of Amazon EC2 instances as well as create and manage launch templates and tags. AWS Batch creates and manages EC2 Spot Fleet requests for some EC2 Spot compute environments.

- **ecs** - Allows AWS Batch to create and managed Amazon ECS clusters, task-definitions and tasks for job execution.

- **eks** - Allows AWS Batch to describe the Amazon EKS cluster resource for validations.

- **iam** - Allows AWS Batch to validate and pass roles provided by owner to Amazon EC2, Amazon EC2 Auto Scaling and Amazon ECS.

- **logs** – Allows AWS Batch to create and manage log groups and log streams for AWS Batch jobs.

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 AWS Batch

You don't need to manually create a service-linked role. When you `CreateComputeEnvironment` in the AWS Management Console, the AWS CLI, or the AWS API, and don't specify a value for the `serviceRole` parameter, AWS Batch 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. Also, if you were using the AWS Batch service before March 10, 2021, when it began supporting service-linked roles, then AWS Batch created the `AWSServiceRoleForBatch` role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

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 `CreateComputeEnvironment`, AWS Batch creates the service-linked role for you again.

Editing a service-linked role for AWS Batch

With AWS Batch, you can't edit the `AWSServiceRoleForBatch` service-linked role. After you create a service-linked role, you can't 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*.

To allow an IAM entity to edit the description of the `AWSServiceRoleForBatch` service-linked role

Add the following statement to the permissions policy. This allows the IAM entity to edit the description of a service-linked role.

```json
{
    "Effect": "Allow",
    "Action": ["iam:UpdateRoleDescription"],
    "Resource": "arn:aws:iam::*:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch",
}
```
Deleting a service-linked role for AWS Batch

We recommend, if you no longer need to use a feature or service that requires a service-linked role, you delete that role. That way, you don’t have an unused entity that’s not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

To allow an IAM entity to delete the AWSServiceRoleForBatch service-linked role

Add the following statement to the permissions policy. This allows the IAM entity to delete a service-linked role.

```
{
   "Effect": "Allow",
   "Action": [
      "iam:DeleteServiceLinkedRole",
      "iam:GetServiceLinkedRoleDeletionStatus"
   ],
   "Resource": "arn:aws:iam::*:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch",
   "Condition": {"StringLike": {"iam:AWSServiceName": "batch.amazonaws.com"}}
}
```

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and delete all of the AWS Batch compute environments that use the role in all AWS Regions in a single partition.

To check whether the service-linked role has an active session

2. In the navigation pane, choose Roles and then the AWSServiceRoleForBatch 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 don't know whether AWS Batch is using the AWSServiceRoleForBatch role, you can try to delete the role. If the service is using the role, then the role will fail to delete. 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 can't revoke the session for a service-linked role.

---

**To remove AWS Batch resources used by the AWSServiceRoleForBatch service-linked role**

You must delete all AWS Batch compute environments that use the AWSServiceRoleForBatch role in all AWS Regions before you can delete the AWSServiceRoleForBatch role.

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Compute environments**.
4. Select the compute environment.
5. Choose **Disable**. Wait for the **State** to change to **DISABLED**.
6. Select the compute environment.
7. Choose **Delete**. Confirm that you want to delete the compute environment by choosing **Delete compute environment**.
8. Repeat steps 1–7 for all compute environments that use the service-linked role in all Regions.

**Deleting a service-linked role in IAM (Console)**

You can use the IAM console to delete a service-linked role.

**To delete a service-linked role (console)**

1. Sign in to the AWS Management Console and open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).
2. In the navigation pane of the IAM console, choose **Roles**. Then select the check box next to **AWSServiceRoleForBatch**, not the name or row itself.
3. Choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. If you want to proceed, choose Yes, Delete to submit the service-linked role for deletion.

5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, after you submit the role for deletion, the deletion task can succeed or fail.
   - If the task succeeds, then the role is removed from the list and a notification of success appears at the top of the page.
   - If the task fails, you can choose View details or View Resources from the notifications to learn why the deletion failed. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   **Note**
   You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources.

   - If the task fails and the notification does not include a list of resources, then the service might not return that information. To learn how to clean up the resources for that service, see AWS services that work with IAM. Find your service in the table, and choose the Yes link to view the service-linked role documentation for that service.

**Deleting a service-linked role in IAM (AWS CLI)**

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

**To delete a service-linked role (CLI)**

1. Because a service-linked role can't be deleted if it's being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions aren't met. You
must capture the deletion-task-id from the response to check the status of the deletion task. Enter the following command to submit a service-linked role deletion request:

```bash
$ aws iam delete-service-linked-role --role-name AWSServiceRoleForBatch
```

2. Use the following command to check the status of the deletion task:

```bash
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service’s resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

### Note

You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them. Or, it might not report any resources. To learn how to clean up the resources for a service that doesn't report any resources, see AWS services that work with IAM. Find your service in the table, and choose the Yes link to view the service-linked role documentation for that service.

### Deleting a service-linked role in IAM (AWSAPI)

You can use the IAM API to delete a service-linked role.

#### To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call `DeleteServiceLinkedRole`. In the request, specify the AWSServiceRoleForBatch role name.
Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the DeletionTaskId from the response to check the status of the deletion task.

2. To check the status of the deletion, call `GetServiceLinkedRoleDeletionStatus`. In the request, specify the DeletionTaskId.

The status of the deletion task can be NOT_STARTED, IN PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

**Note**
You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not report any resources, see [AWS services that work with IAM](#). Find your service in the table, and choose the Yes link to view the service-linked role documentation for that service.

**Supported Regions for AWS Batch service-linked roles**

AWS Batch supports using service-linked roles in all of the Regions where the service is available. For more information, see [AWS Batch endpoints](#).

**AWS managed policies for AWS Batch**

You can use AWS managed policies for simpler identity access management for your team and provisioned AWS resources. AWS managed policies cover a variety of common use cases, are
available by default in your AWS account, and are maintained and updated on your behalf. You can't change the permissions in AWS managed policies. If you require greater flexibility, you can alternatively choose to create IAM customer managed policies. This way, you can provide your team provisioned resources with only the exact permissions they need.

For more information about AWS managed policies, see [AWS managed policies](#) in the IAM User Guide.

AWS services maintain and update AWS managed policies on your behalf. Periodically, AWS services add additional permissions to an AWS managed policy. AWS managed policies are most likely updated when a new feature launch or operation becomes available. These updates automatically affect all identities (users, groups, and roles) where the policy is attached. However, they don't remove permissions or 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.

### AWS managed policy: BatchServiceRolePolicy

The BatchServiceRolePolicy managed IAM policy is used by the AWSServiceRoleForBatch service-linked role. This allows AWS Batch to perform actions on your behalf. You can't attach this policy to your IAM entities. For more information, see [Using service-linked roles for AWS Batch](#).

This policy allows AWS Batch to complete the following actions on specific resources:

- **autoscaling** – Allows AWS Batch to create and manage Amazon EC2 Auto Scaling resources. AWS Batch creates and manages Amazon EC2 Auto Scaling groups for most compute environments.

- **ec2** – Allows AWS Batch to control the lifecycle of Amazon EC2 instances as well as create and manage launch templates and tags. AWS Batch creates and manages EC2 Spot Fleet requests for some EC2 Spot compute environments.
- **ecs** - Allows AWS Batch to create and managed Amazon ECS clusters, task-definitions and tasks for job execution.

- **eks** - Allows AWS Batch to describe the Amazon EKS cluster resource for validations.

- **iam** - Allows AWS Batch to validate and pass roles provided by owner to Amazon EC2, Amazon EC2 Auto Scaling and Amazon ECS.

- **logs** – Allows AWS Batch to create and manage log groups and log streams for AWS Batch jobs.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "AWSBatchPolicyStatement1",
           "Effect": "Allow",
           "Action": [
               "ec2:DescribeAccountAttributes",
               "ec2:DescribeInstances",
               "ec2:DescribeInstanceStatus",
               "ec2:DescribeInstanceAttribute",
               "ec2:DescribeSubnets",
               "ec2:DescribeSecurityGroups",
               "ec2:DescribeKeyPairs",
               "ec2:DescribeImages",
               "ec2:DescribeImageAttribute",
               "ec2:DescribeSpotInstanceRequests",
               "ec2:DescribeSpotFleetInstances",
               "ec2:DescribeSpotFleetRequests",
               "ec2:DescribeSpotPriceHistory",
               "ec2:DescribeSpotFleetRequestHistory",
               "ec2:DescribeVpcClassicLink",
               "ec2:DescribeLaunchTemplateVersions",
               "ec2:RequestSpotFleet",
               "autoscaling:DescribeAccountLimits",
               "autoscaling:DescribeAutoScalingGroups",
               "autoscaling:DescribeLaunchConfigurations",
               "autoscaling:DescribeAutoScalingInstances",
               "autoscaling:DescribeScalingActivities",
               "eks:DescribeCluster",
               "ecs:DescribeClusters",
               "ecs:DescribeContainerInstances",
               "ecs:DescribeTaskDefinition",
               "ecs:DescribeTasks",
```
"ecs:ListClusters",
"ecs:ListContainerInstances",
"ecs:ListTaskDefinitionFamilies",
"ecs:ListTaskDefinitions",
"ecs:ListTasks",
"ecs:DeregisterTaskDefinition",
"ecs:TagResource",
"ecs:ListAccountSettings",
"logs:DescribeLogGroups",
"iam:GetInstanceProfile",
"iam:GetRole"
],
"Resource": "*
",
{
"Sid": "AWSBatchPolicyStatement2",
"Effect": "Allow",
"Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream"
],
"Resource": "arn:aws:logs:*:*:log-group:/aws/batch/job*"
},
{
"Sid": "AWSBatchPolicyStatement3",
"Effect": "Allow",
"Action": [
    "logs:PutLogEvents"
],
"Resource": "arn:aws:logs:*:*:log-group:/aws/batch/job*:log-stream:*"
},
{
"Sid": "AWSBatchPolicyStatement4",
"Effect": "Allow",
"Action": [
    "autoscaling:CreateOrUpdateTags"
],
"Resource": "*
",
"Condition": {
    "Null": {
        "aws:RequestTag/AWSBatchServiceTag": "false"
    }
}
}
AWS Batch

User Guide

{
"Sid": "AWSBatchPolicyStatement5",
"Effect": "Allow",
"Action": "iam:PassRole",
"Resource": [
"*"
],
"Condition": {
"StringEquals": {
"iam:PassedToService": [
"ec2.amazonaws.com",
"ec2.amazonaws.com.cn",
"ecs-tasks.amazonaws.com"
]
}
}
},
{
"Sid": "AWSBatchPolicyStatement6",
"Effect": "Allow",
"Action": "iam:CreateServiceLinkedRole",
"Resource": "*",
"Condition": {
"StringEquals": {
"iam:AWSServiceName": [
"spot.amazonaws.com",
"spotfleet.amazonaws.com",
"autoscaling.amazonaws.com",
"ecs.amazonaws.com"
]
}
}
},
{
"Sid": "AWSBatchPolicyStatement7",
"Effect": "Allow",
"Action": [
"ec2:CreateLaunchTemplate"
],
"Resource": "*",
"Condition": {
"Null": {
"aws:RequestTag/AWSBatchServiceTag": "false"
}

AWS managed policies

413


} 
,
{
  "Sid": "AWSBatchPolicyStatement8",
  "Effect": "Allow",
  "Action": [
    "ec2:TerminateInstances",
    "ec2:CancelSpotFleetRequests",
    "ec2:ModifySpotFleetRequest",
    "ec2:DeleteLaunchTemplate"
  ],
  "Resource": "*",
  "Condition": {
    "Null": {
      "aws:ResourceTag/AWSBatchServiceTag": "false"
    }
  }
}
,
{
  "Sid": "AWSBatchPolicyStatement9",
  "Effect": "Allow",
  "Action": [
    "autoscaling:CreateLaunchConfiguration",
    "autoscaling:DeleteLaunchConfiguration"
  ],
  "Resource": "arn:aws:autoscaling::*:*:launchConfiguration::*:launchConfigurationName/AWSBatch*"
},
{
  "Sid": "AWSBatchPolicyStatement10",
  "Effect": "Allow",
  "Action": [
    "autoscaling:CreateAutoScalingGroup",
    "autoscaling:UpdateAutoScalingGroup",
    "autoscaling:SetDesiredCapacity",
    "autoscaling:DeleteAutoScalingGroup",
    "autoscaling:SuspendProcesses",
    "autoscaling:PutNotificationConfiguration",
    "autoscaling:TerminateInstanceInAutoScalingGroup"
  ],
  "Resource": "arn:aws:autoscaling::*:*:autoScalingGroup::*:autoScalingGroupName/AWSBatch*"
}
}
"Sid": "AWSBatchPolicyStatement11",
"Effect": "Allow",
"Action": [
    "ecs:DeleteCluster",
    "ecs:DeregisterContainerInstance",
    "ecs:RunTask",
    "ecs:StartTask",
    "ecs:StopTask"
],
"Resource": "arn:aws:ecs:*::*:cluster/AWSBatch*"
},
{
"Sid": "AWSBatchPolicyStatement12",
"Effect": "Allow",
"Action": [
    "ecs:RunTask",
    "ecs:StartTask",
    "ecs:StopTask"
],
"Resource": "arn:aws:ecs:*::*:task-definition/*"
},
{
"Sid": "AWSBatchPolicyStatement13",
"Effect": "Allow",
"Action": [
    "ecs:StopTask"
],
"Resource": "arn:aws:ecs:*::*:task/*/*"
},
{
"Sid": "AWSBatchPolicyStatement14",
"Effect": "Allow",
"Action": [
    "ecs:CreateCluster",
    "ecs:RegisterTaskDefinition"
],
"Resource": "*",
"Condition": {
    "Null": {
        "aws:RequestTag/AWSBatchServiceTag": "false"
    }
}
"Sid": "AWSBatchPolicyStatement15",
"Effect": "Allow",
"Action": "ec2:RunInstances",
"Resource": [
    "arn:aws:ec2:*::*:image/**",
    "arn:aws:ec2:*::*:snapshot/**",
    "arn:aws:ec2:*::*:subnet/**",
    "arn:aws:ec2:*::*:network-interface/**",
    "arn:aws:ec2:*::*:security-group/**",
    "arn:aws:ec2:*::*:volume/**",
    "arn:aws:ec2:*::*:key-pair/**",
    "arn:aws:ec2:*::*:launch-template/**",
    "arn:aws:ec2:*::*:placement-group/**",
    "arn:aws:ec2:*::*:capacity-reservation/**",
    "arn:aws:ec2:*::*:elastic-gpu/**",
    "arn:aws:elastic-inference:***:elastic-inference-accelerator/**",
    "arn:aws:resource-groups:***:group/**"
],

"Sid": "AWSBatchPolicyStatement16",
"Effect": "Allow",
"Action": "ec2:RunInstances",
"Resource": "arn:aws:ec2:*:*:instance/**",
"Condition": {
    "Null": {
        "aws:RequestTag/AWSBatchServiceTag": "false"
    }
},

"Sid": "AWSBatchPolicyStatement17",
"Effect": "Allow",
"Action": ["ec2:CreateTags"],
"Resource": ["***"],
"Condition": {
    "StringEquals": {
        "ec2:CreateAction": ["RunInstances",
        "CreateLaunchTemplate",
        "CreateLaunchTemplateInstance"],

"Sid": "AWSBatchPolicyStatement18",
"Effect": "Allow",
"Action": ["ec2:ModifyInstanceAttribute"],
"Condition": {
    "StringEquals": {
        "ec2:InstanceId": ["i-0123456789abcdef0", "i-0123456789abcdef1", "i-0123456789abcdef2", "i-0123456789abcdef3"]
    }
},

"Sid": "AWSBatchPolicyStatement19",
"Effect": "Allow",
"Action": ["ec2:StopInstances"],
"Condition": {
    "StringEquals": {
        "ec2:InstanceId": ["i-0123456789abcdef0", "i-0123456789abcdef1", "i-0123456789abcdef2", "i-0123456789abcdef3"]
    }
}
AWS managed policy: AWSBatchServiceRole policy

The role permissions policy named **AWSBatchServiceRole** allows AWS Batch to complete the following actions on specific resources:

The **AWSBatchServiceRole** managed IAM policy is often used by a role named **AWSBatchServiceRole** and includes the following permissions. Following the standard security advice of granting least privilege, the **AWSBatchServiceRole** 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. This AWS Batch managed policy and role can be used with most compute environment types, but service-linked role usage is preferred for a less-error-prone, better scoped and improved managed experience.

- **autoscaling** – Allows AWS Batch to create and manage Amazon EC2 Auto Scaling resources. AWS Batch creates and manages Amazon EC2 Auto Scaling groups for most compute environments.
- **ec2** – Allows AWS Batch to manage the lifecycle of Amazon EC2 instances as well as create and manage launch templates and tags. AWS Batch creates and manages EC2 Spot Fleet requests for some EC2 Spot compute environments.
- **ecs** - Allows AWS Batch to create and managed Amazon ECS clusters, task-definitions and tasks for job execution.
- **iam** - Allows AWS Batch to validate and pass roles provided by owner to Amazon EC2, Amazon EC2 Auto Scaling and Amazon ECS.
- **logs** – Allows AWS Batch to create and manage log groups and log streams for AWS Batch jobs.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSBatchPolicyStatement1",
```
"Effect": "Allow",
"Action": [
  "ec2:DescribeAccountAttributes",
  "ec2:DescribeInstances",
  "ec2:DescribeInstanceStatus",
  "ec2:DescribeInstanceAttribute",
  "ec2:DescribeSubnets",
  "ec2:DescribeSecurityGroups",
  "ec2:DescribeKeyPairs",
  "ec2:DescribeImages",
  "ec2:DescribeImageAttribute",
  "ec2:DescribeSpotInstanceRequests",
  "ec2:DescribeSpotFleetInstances",
  "ec2:DescribeSpotFleetRequests",
  "ec2:DescribeSpotPriceHistory",
  "ec2:DescribeSpotFleetRequestHistory",
  "ec2:DescribeVpcClassicLink",
  "ec2:DescribeLaunchTemplateVersions",
  "ec2:CreateLaunchTemplate",
  "ec2:DeleteLaunchTemplate",
  "ec2:RequestSpotFleet",
  "ec2:CancelSpotFleetRequests",
  "ec2:ModifySpotFleetRequest",
  "ec2:TerminateInstances",
  "ec2:RunInstances",
  "autoscaling:DescribeAccountLimits",
  "autoscaling:DescribeAutoScalingGroups",
  "autoscaling:DescribeLaunchConfigurations",
  "autoscaling:DescribeAutoScalingInstances",
  "autoscaling:DescribeScalingActivities",
  "autoscaling:CreateLaunchConfiguration",
  "autoscaling:CreateAutoScalingGroup",
  "autoscaling:UpdateAutoScalingGroup",
  "autoscaling:SetDesiredCapacity",
  "autoscaling:DeleteLaunchConfiguration",
  "autoscaling:DeleteAutoScalingGroup",
  "autoscaling:CreateOrUpdateTags",
  "autoscaling:SuspendProcesses",
  "autoscaling:PutNotificationConfiguration",
  "autoscaling:TerminateInstanceInAutoScalingGroup",
  "ecs:DescribeClusters",
  "ecs:DescribeContainerInstances",
  "ecs:DescribeTaskDefinition",
  "ecs:DescribeTasks"}
"ecs:ListAccountSettings",
"ecs:ListClusters",
"ecs:ListContainerInstances",
"ecs:ListTaskDefinitionFamilies",
"ecs:ListTaskDefinitions",
"ecs:ListTasks",
"ecs:CreateCluster",
"ecs:DeleteCluster",
"ecs:RegisterTaskDefinition",
"ecs:DeregisterTaskDefinition",
"ecs:RunTask",
"ecs:StartTask",
"ecs:StopTask",
"ecs:UpdateContainerAgent",
"ecs:DeregisterContainerInstance",
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:PutLogEvents",
"logs:DescribeLogGroups",
"iam:GetInstanceProfile",
"iam:GetRole"
]
,
"Resource": "*"
},
{
"Sid": "AWSBatchPolicyStatement2",
"Effect": "Allow",
"Action": "ecs:TagResource",
"Resource": [
   "arn:aws:ecs:*:*:task/*_Batch_*"
]
},
{
"Sid": "AWSBatchPolicyStatement3",
"Effect": "Allow",
"Action": "iam:PassRole",
"Resource": [
   "*"
],
"Condition": {
   "StringEquals": {
      "iam:PassedToService": [
         "ec2.amazonaws.com",
         "ec2.amazonaws.com.cn",
         "ec2.ap-southeast-1.amazonaws.com",
         "ec2.ap-southeast-2.amazonaws.com",
         "ec2.eu-central-1.amazonaws.com",
         "ec2.eu-west-1.amazonaws.com",
         "ec2.eu-west-2.amazonaws.com",
         "ec2.us-east-1.amazonaws.com",
         "ec2.us-west-1.amazonaws.com",
         "ec2.us-west-2.amazonaws.com"
      ]
   }
}
AWS managed policy: AWSBatchFullAccess

The AWSBatchFullAccess policy grants AWS Batch actions full access to AWS Batch resources. It also grants describe and list action access for Amazon EC2, Amazon ECS, Amazon EKS, CloudWatch,
and IAM services. This is so that IAM identities, either users or roles, can view AWS Batch managed resources that were created on their behalf. Last, this policy also allows for selected IAM roles to be passed to those services.

You can attach **AWSBatchFullAccess** to your IAM entities. AWS Batch also attaches this policy to a service role that allows AWS Batch to perform actions on your behalf.

```json
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect":"Allow",
      "Action": [
        "batch:*",
        "cloudwatch:GetMetricStatistics",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeImages",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ecs:DescribeClusters",
        "ecs:Describe*",
        "ecs:List*",
        "eks:DescribeCluster",
        "eks:ListClusters",
        "logs:Describe*",
        "logs:Get*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "iam:ListInstanceProfiles",
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Effect":"Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AWSBatchServiceRole",
```
AWS Batch updates to AWS managed policies

View details about updates to AWS managed policies for AWS Batch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS Batch Document history page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BatchServiceRolePolicy</strong> policy updated</td>
<td>Updated to add support for describing Spot Fleet request history and Amazon EC2 Auto Scaling activities.</td>
<td>December 5, 2023</td>
</tr>
<tr>
<td><strong>AWSBatchServiceRole</strong> policy added</td>
<td>Updated to add statement IDs, grant AWS Batch permissions to <code>ec2:DescribeSpotFleetRequestHistory</code></td>
<td>December 5, 2023</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>and autoscaling:DescribeScalingActivities</td>
<td></td>
</tr>
<tr>
<td><strong>BatchServiceRolePolicy</strong> policy updated</td>
<td>Updated to add support for describing Amazon EKS clusters.</td>
<td>October 20, 2022</td>
</tr>
<tr>
<td><strong>AWSBatchFullAccess</strong> policy updated</td>
<td>Updated to add support for listing and describing Amazon EKS clusters.</td>
<td>October 20, 2022</td>
</tr>
<tr>
<td><strong>BatchServiceRolePolicy</strong> policy updated</td>
<td>Updated to add support for Amazon EC2 Capacity Reservation groups that are managed by AWS Resource Groups. For more information, see <a href="https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/capacity-reservation-working.html">Work with Capacity Reservation groups</a> in <em>Amazon EC2 User Guide for Linux Instances.</em></td>
<td>May 18, 2022</td>
</tr>
<tr>
<td><strong>BatchServiceRolePolicy</strong> and <strong>AWSBatchServiceRole</strong> policies updated</td>
<td>Updated to add support for describing the status of AWS Batch managed instances in Amazon EC2 so that unhealthy instances are replaced.</td>
<td>December 6, 2021</td>
</tr>
<tr>
<td><strong>BatchServiceRolePolicy</strong> policy updated</td>
<td>Updated to add support for placement group, capacity reservation, elastic GPU, and Elastic Inference resources in Amazon EC2.</td>
<td>March 26, 2021</td>
</tr>
<tr>
<td><strong>BatchServiceRolePolicy</strong> policy added</td>
<td>With the <strong>BatchServiceRolePolicy</strong> managed policy for the <strong>AWSServiceRoleForBatch</strong> service-linked role, you can use a service-linked role managed by AWS Batch. With this policy, you don't need to maintain your own role for use in your compute environments.</td>
<td>March 10, 2021</td>
</tr>
</tbody>
</table>
Access AWS Batch using an interface endpoint

You can use AWS PrivateLink to create a private connection between your VPC and AWS Batch. You can access AWS Batch as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS Batch.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for AWS Batch.

For more information, see [Interface VPC endpoints](#) in the [AWS PrivateLink Guide](#).

Considerations for AWS Batch

Before you set up an interface endpoint for AWS Batch, review [Interface endpoint properties and limitations](#) in the [AWS PrivateLink Guide](#).

AWS Batch supports making calls to all of its API actions through the interface endpoint.

Before you set up interface VPC endpoints for AWS Batch, be aware of the following considerations:

- Jobs using Fargate resources launch type don't require the interface VPC endpoints for Amazon ECS, but you might need interface VPC endpoints for AWS Batch, Amazon ECR, Secrets Manager, or Amazon CloudWatch Logs described in the following points.
• To run jobs, you must create the interface VPC endpoints for Amazon ECS. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide.

• To allow your jobs 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.

• To allow your jobs 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 jobs 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.

• Jobs using the EC2 resources 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 in the Amazon Elastic Container Service Developer 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 AWS Batch.

• 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 port 443 from the private subnet of the VPC.

• AWS Batch does not support VPC interface endpoints in the following AWS Regions:
  • Asia Pacific (Osaka) (ap-northeast-3)
  • Asia Pacific (Jakarta) (ap-southeast-3)

Create an interface endpoint for AWS Batch

You can create an interface endpoint for AWS Batch using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the AWS PrivateLink Guide.

Create an interface endpoint for AWS Batch using the following service name:
For example:

```
com.amazonaws.us-east-2.batch
```

In the aws-cn partition, the format is different:

```
cn.com.amazonaws.region.batch
```

For example:

```
cn.com.amazonaws.cn-northwest-1.batch
```

If you enable private DNS for the interface endpoint, you can make API requests to AWS Batch using its default Regional DNS name. For example, batch.us-east-1.amazonaws.com.

For more information, see Access a service through an interface endpoint in the AWS PrivateLink Guide.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to AWS Batch through the interface endpoint. To control the access allowed to AWS Batch from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the AWS PrivateLink Guide.

Example: VPC endpoint policy for AWS Batch actions
The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed AWS Batch actions for all principals on all resources.

```json
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob",
                "batch:ListJobs",
                "batch:DescribeJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

### Compliance Validation for AWS Batch

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.

- **AWS Customer Compliance Guides** – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- **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.

## Infrastructure Security in AWS Batch

As a managed service, AWS Batch 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 AWS Batch 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, but AWS Batch does support resource-based access policies, which can include restrictions based on the source IP address. You can also use AWS Batch policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given AWS Batch resource from only the specific VPC within the AWS network.
Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your AWS Batch services to help you track each service's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to AWS Batch and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value.

You can work with tags using the AWS Management Console, the AWS CLI, and the AWS Batch API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.
Tagging your resources

You can tag new or existing AWS Batch compute environments, jobs, job definitions, job queues, and scheduling policies.

If you're using the AWS Batch console, you can apply tags to new resources when they are created or to existing resources at any time using the Tags tab on the relevant resource page.

If you're using the AWS Batch API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the tags parameter on the relevant API action or to existing resources using the TagResource API action. For more information, see TagResource.

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, the resource creation process fails. This ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after resource creation.

The following table describes the AWS Batch resources that can be tagged, and the resources that can be tagged on creation.

### Tagging support for AWS Batch resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Supports tags</th>
<th>Supports tag propagation</th>
<th>Supports tagging on creation (AWS Batch API, AWS CLI, AWS SDK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Batch compute environments</td>
<td>Yes</td>
<td>No. Compute environment tags do not propagate to any other resources. Tags for the resources are specified in the tags member of the computeResources object passed in the CreateCom</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values, as it is reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags-per-resource limit.
Working with tags using the console

Using the AWS Batch console, you can manage the tags associated with new or existing compute environments, jobs, job definitions, and job queues.

Adding tags on an individual resource on creation

You can add tags to AWS Batch compute environments, jobs, job definitions, job queues, and scheduling policies when you create them.

Adding and deleting tags on an individual resource

AWS Batch allows you to add or delete tags associated with your clusters directly from the resource's page.

To add or delete a tag on an individual resource

2. From the navigation bar, choose the Region to use.
3. In the navigation pane, choose a resource type (for example, Job Queues).
4. Choose a specific resource, then choose Edit tags.
5. Add or delete your tags as necessary.
   - To add a tag — specify the key and value in the empty text boxes at the end of the list.
   - To delete a tag — choose the Delete icon button next to the tag.
6. Repeat this process for each tag you want to add or delete, and then choose Edit tags to finish.

Working with tags using the CLI or API

Use the following AWS CLI commands or AWS Batch API operations to add, update, list, and delete the tags for your resources.
Tagging support for AWS Batch resources

<table>
<thead>
<tr>
<th>Task</th>
<th>API action</th>
<th>AWS CLI</th>
<th>AWS Tools for Windows PowerShell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add or overwrite one or more tags.</td>
<td>TagResource</td>
<td>tag-resource</td>
<td>Add-BATResourceTag</td>
</tr>
<tr>
<td>Delete one or more tags.</td>
<td>UntagResource</td>
<td>untag-resource</td>
<td>Remove-BATResourceTag</td>
</tr>
<tr>
<td>List tags for a resource</td>
<td>ListTagsForResource</td>
<td>list-tags-for-resource</td>
<td>Get-BATResourceTag</td>
</tr>
</tbody>
</table>

The following examples show how to tag or untag resources using the AWS CLI.

**Example 1: Tag an existing resource**

The following command tags an existing resource.

```bash
aws batch tag-resource --resource-arn resource_ARN --tags team=devs
```

**Example 2: Untag an existing resource**

The following command deletes a tag from an existing resource.

```bash
aws batch untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

**Example 3: List tags for a resource**

The following command lists the tags associated with an existing resource.

```bash
aws batch list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.
<table>
<thead>
<tr>
<th>Task</th>
<th>API action</th>
<th>AWS CLI</th>
<th>AWS Tools for Windows PowerShell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a compute environment</td>
<td><strong>CreateComputeEnvironment</strong></td>
<td><strong>create-compute-environment</strong></td>
<td><strong>New-BATComputeEnvironment</strong></td>
</tr>
<tr>
<td>Create a job queue</td>
<td><strong>CreateJobQueue</strong></td>
<td><strong>create-job-queue</strong></td>
<td><strong>New-BATJobQueue</strong></td>
</tr>
<tr>
<td>Create a scheduling policy</td>
<td><strong>CreateSchedulingPolicy</strong></td>
<td><strong>create-scheduling-policy</strong></td>
<td><strong>New-BATSchedulingPolicy</strong></td>
</tr>
<tr>
<td>Register a job definition</td>
<td><strong>RegisterJobDefinition</strong></td>
<td><strong>register-job-definition</strong></td>
<td><strong>Register-BATJobDefinition</strong></td>
</tr>
<tr>
<td>Submit a job</td>
<td><strong>SubmitJob</strong></td>
<td><strong>submit-job</strong></td>
<td><strong>Submit-BATJob</strong></td>
</tr>
</tbody>
</table>

Working with tags using the CLI or API
AWS Batch service quotas

The following table provides the service quotas for AWS Batch that can't be changed. Each quota is Region specific.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of job queues. For more information, see Job queues.</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of compute environments across Amazon ECS and Amazon EKS. For more information, see Compute environment.</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of compute environments per Amazon EKS cluster.</td>
<td>5</td>
</tr>
<tr>
<td>Maximum number of compute environments for each job queue</td>
<td>3</td>
</tr>
<tr>
<td>Maximum number of job dependencies for a job</td>
<td>20</td>
</tr>
<tr>
<td>Maximum job definition size (for RegisterJobDefinition API operations)</td>
<td>24 KiB</td>
</tr>
<tr>
<td>Maximum job payload size (for SubmitJob API operations)</td>
<td>30 KiB</td>
</tr>
<tr>
<td>Maximum array size for array jobs</td>
<td>10000</td>
</tr>
<tr>
<td>Maximum number of jobs in SUBMITTED state</td>
<td>1000000</td>
</tr>
<tr>
<td>Maximum number of transactions per second (TPS) for each account for SubmitJob operations</td>
<td>50</td>
</tr>
</tbody>
</table>

Depending on how you use AWS Batch, additional quotas might apply. To learn about Amazon EC2 quotas, see Amazon EC2 Service Quotas in the AWS General Reference. For more information about Amazon ECS quotas, see Amazon ECS Service Quotas in the AWS General Reference. For more information about Amazon EKS quotas, see Amazon EKS Service Quotas in the AWS General Reference.
Troubleshooting AWS Batch

You might need to troubleshoot issues that are related to your compute environments, job queues, job definitions, or jobs. This chapter describes how to troubleshoot and resolve such issues in your AWS Batch environment.

AWS Batch uses IAM policies, roles, and permissions, and runs on Amazon EC2, Amazon ECS, AWS Fargate, and Amazon Elastic Kubernetes Service infrastructure. To troubleshoot issues that are related to these services, see the following:

- [Troubleshooting IAM](#) in the IAM User Guide
- [Amazon ECS troubleshooting](#) in the Amazon Elastic Container Service Developer Guide
- [Amazon EKS troubleshooting](#) in the Amazon EKS User Guide
- [Troubleshoot EC2 instances](#) in the Amazon EC2 User Guide for Linux Instances

Contents

- AWS Batch
  - INVALID compute environment
    - Incorrect role name or ARN
    - Repairing an INVALID compute environment
  - Jobs stuck in a RUNNABLE status
  - Spot Instances not tagged on creation
  - Spot Instances not scaling down
    - Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role in the AWS Management Console
    - Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role with the AWS CLI
  - Can't retrieve Secrets Manager secrets
  - Can't override job definition resource requirements
  - Error message when you update the desiredvCpus setting
- AWS Batch on Amazon EKS
  - INVALID compute environment
    - Unsupported Kubernetes version
• Instance profile doesn't exist
• Invalid Kubernetes namespace
• Deleted compute environment
• Nodes don't join the Amazon EKS cluster
• AWS Batch on Amazon EKS job is stuck in RUNNABLE status
• Verify that the aws-auth ConfigMap is configured correctly
• RBAC permissions or bindings aren't configured properly

AWS Batch

INVALID compute environment

It's possible that you might have incorrectly configured a managed compute environment. If you did, the compute environment enters an INVALID state and can't accept jobs for placement. The following sections describe the possible causes and how to troubleshoot based on the cause.

Incorrect role name or ARN

The most common cause for a compute environment to enter an INVALID state is that the AWS Batch service role or the Amazon EC2 Spot Fleet role has an incorrect name or Amazon Resource Name (ARN). This is more common with compute environments that are created using the AWS CLI or the AWS SDKs. When you create a compute environment in the AWS Management Console, AWS Batch helps you choose the correct service or Spot Fleet roles. However, suppose that you manually enter the name or the ARN and enter them incorrectly. Then, the resulting compute environment is also INVALID.

However, suppose that you manually enter the name or ARN for an IAM resource in an AWS CLI command or your SDK code. In this case, AWS Batch can't validate the string. Instead, AWS Batch must accept the bad value and attempt to create the environment. If AWS Batch fails to create the environment, the environment moves to an INVALID state, and you see the following errors.

For an invalid service role:

CLIENT_ERROR - Not authorized to perform sts:AssumeRole (Service: AWSSecurityTokenService; Status Code: 403; Error Code: AccessDenied; Request ID: dc0e2d28-2e99-11e7-b372-7fcc6fb65fe7)
For an invalid Spot Fleet role:

CLIENT_ERROR - Parameter: SpotFleetRequestConfig.IamFleetRole is invalid. (Service: AmazonEC2; Status Code: 400; Error Code: InvalidSpotFleetRequestConfig; Request ID: 331205f0-5ae3-4cea-bac4-897769639f8d) Parameter: SpotFleetRequestConfig.IamFleetRole is invalid

One common cause for this issue is the following scenario. You only specify the name of an IAM role when using the AWS CLI or the AWS SDKs, instead of the full Amazon Resource Name (ARN). Depending on how you created the role, the ARN might contain a `aws-service-role` path prefix. For example, if you manually create the AWS Batch service role using the procedures in Using service-linked roles for AWS Batch, your service role ARN might look like the following.

```
arn:aws:iam::123456789012:role/AWSBatchServiceRole
```

However, if you created the service role as part of the console first run wizard today, your service role ARN might look like the following.

```
arn:aws:iam::123456789012:role/aws-service-role/AWSBatchServiceRole
```

This issue can also occur if you attach the the AWS Batch service-level policy (AWSBatchServiceRole) to a non-service role. For example, you may receive an error message that resembles the following in this scenario:

```
CLIENT_ERROR - User: arn:aws:sts::account_number:assumed-role/batch-replacement-role/aws-batch is not authorized to perform: action on resource ...
```

To resolve this issue, do one of the following.

- Use an empty string for the service role when you create the AWS Batch compute environment.
- Specify the service role in the following format: `arn:aws:iam::account_number:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch`.

When you only specify the name of an IAM role when using the AWS CLI or the AWS SDKs, AWS Batch assumes that your ARN doesn't use the `aws-service-role` path prefix. Because of
this, we recommend that you specify the full ARN for your IAM roles when you create compute environments.

To repair a compute environment that's misconfigured this way, see Repairing an INVALID compute environment.

**Repairing an INVALID compute environment**

When you have a compute environment in an INVALID state, update it to repair the invalid parameter. For an Incorrect role name or ARN, update the compute environment using the correct service role.

**To repair a misconfigured compute environment**

2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose Compute environments.
4. On the Compute environments page, select the radio button next to the compute environment to edit, and then choose Edit.
5. On the Update compute environment page, for Service role, choose the IAM role to use with your compute environment. The AWS Batch console only displays roles that have the correct trust relationship for compute environments.
6. Choose Save to update your compute environment.

**Jobs stuck in a RUNNABLE status**

Suppose that your compute environment contains compute resources, but your jobs don't progress beyond the RUNNABLE status. Then, it's likely that something is preventing the jobs from being placed on a compute resource and causing your job queues to be blocked. Here's how to know if your job is waiting for its turn or stuck and blocking the queue.

If AWS Batch detects that you have a RUNNABLE job at the head and blocking the queue, you'll receive a blocked job queue event from Amazon CloudWatch Events with the reason. The same reason is also updated into the statusReason field as a part of ListJobs and DescribeJobs API calls.

Optionally, you can configure the jobStateTimeLimitActions parameter through CreateJobQueue and UpdateJobQueue API actions.
Note
Currently, the only action you can use with jobStateLimitActions.action is to cancel a job.

The jobStateTimeLimitActions parameter is used to specify a set of actions that AWS Batch performs on jobs in a specific state. You can set a time threshold in seconds through the maxTimeSeconds field.

When a job has been in a RUNNABLE state with the defined statusReason, AWS Batch performs the action specified after maxTimeSeconds have elapsed.

For example, you can set the jobStateTimeLimitActions parameter to wait up to 4 hours for any job in the RUNNABLE state that is waiting for sufficient capacity to become available. You can do this by setting statusReason to CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY and maxTimeSeconds to 144000 before cancelling the job and allowing the next job to advance to the head of the job queue.

The following are the reasons that AWS Batch provides when it detects that a job queue is blocked. This list provides the messages returned from the ListJobs and DescribeJobs API actions. These are also the same values you can define for the jobStateLimitActions.statusReason parameter.

1. **Reason:** All connected compute environments have insufficient capacity errors. When requested, AWS Batch detects Amazon EC2 instances that experience insufficient capacity errors. Canceling the job, either manually or by setting the jobStateTimeLimitActions parameter on statusReason, allows the subsequent job to move to the head of the queue.

   • **statusReason message while the job is stuck:**
     CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY - Service cannot fulfill the capacity requested for instance type [instanceTypeName]

   • **reason used for jobStateTimeLimitActions:**
     CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY

   • **statusReason message after the job is canceled:** Canceled by JobStateTimeLimit action due to reason: CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY

   Note:
a. The AWS Batch service role requires autoscaling:DescribeScalingActivities permission for this detection to work. If you use the AWSServiceRoleForBatch service-linked role (SLR) or the AWSBatchServiceRolePolicy managed policy, then you don’t need to take any action because their permission policies are updated.

b. If you use the SLR or the managed policy, you must add the autoscaling:DescribeScalingActivities and ec2:DescribeSpotFleetRequestHistory permissions so that you can receive blocked job queue events and updated job status when in RUNNABLE. In addition, AWS Batch needs these permissions to perform cancellation actions through the jobStateTimeLimitActions parameter even if they are configured on the job queue.

c. In the case of a multi-node parallel (MNP) job, if the attached high-priority, Amazon EC2 compute environment experiences insufficient capacity errors, it blocks the queue even if a lower priority compute environment does experience this error.

2. **Reason:** All compute environments have a maxvCpus parameter that is smaller than the job requirements. Canceling the job, either manually or by setting the jobStateTimeLimitActions parameter on statusReason, allows the subsequent job to move to the head of the queue. Optionally, you can increase the maxvCpus parameter of the primary compute environment to meet the needs of the blocked job.

   • **statusReason message while the job is stuck:**
     MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE - CE(s) associated with the job queue cannot meet the CPU requirement of the job.

   • **reason used for jobStateTimeLimitActions:**
     MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE

   • **statusReason message after the job is canceled:** Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE

3. **Reason:** None of the compute environments have instances that meet the job requirements. When a job requests resources, AWS Batch detects that no attached compute environment is able to accommodate the incoming job. Canceling the job, either manually or by setting the jobStateTimeLimitActions parameter on statusReason, allows the subsequent job to move to the head of the queue. Optionally, you can redefine the compute environment’s allowed instance types to add the necessary job resources.

   • **statusReason message while the job is stuck:**
     MISCONFIGURATION:JOBRESOURCE_REQUIREMENT - The job resource
requirement (vCPU/memory/GPU) is higher than that can be met by the CE(s) attached to the job queue.

• **reason used for jobStateTimeLimitActions:**
  MISCONFIGURATION:JOB_RESOURCE_REQUIREMENT

• **statusReason message after the job is canceled:** Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:JOB_RESOURCE_REQUIREMENT

4. **Reason:** All compute environments have service role issues. To resolve this, compare your service role permissions to the [AWS Batch managed service role permissions](https://aws.amazon.com/batch/managed-service-role/) and address any gaps.

   It's a best practice to use the [AWS Batch SLR for compute environments](https://aws.amazon.com/batch/service-role/) to avoid similar errors.

   Canceling the job, either manually or by setting the jobStateTimeLimitActions parameter on statusReason, allows the subsequent job to move to the head of the queue. Without resolving the service role issue(s), it is likely that the next job will also be blocked as well. It's best to manually investigate and resolve this issue.

   • **statusReason message while the job is stuck:**
     MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS – Batch service role has a permission issue.

   • **reason used for jobStateTimeLimitActions:**
     MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS

   • **statusReason message after the job is canceled:** Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS

5. **Reason:** All compute environments are invalid. For more information, see [INVALID compute environment](https://aws.amazon.com/batch/invalid-compute-environment/). Note: You can't configure a programmable action through the jobStateTimeLimitActions parameter to resolve this error.

   • **statusReason message while the job is stuck:** ACTION_REQUIRED - CE(s) associated with the job queue are invalid.

6. **Reason:** AWS Batch has detected a blocked queue, but is unable to determine the reason.

   Note: You can't configure a programmable action through the jobStateTimeLimitActions parameter to resolve this error. For more information about troubleshooting, see [Why is my AWS Batch job stuck in RUNNABLE on AWS](https://aws.amazon.com/batch/troubleshooting/undo-job-blocked-in-runnable-status/) in re:Post.

   • **statusReason message while the job is stuck:** UNDETERMINED - Batch job is blocked, root cause is undetermined.
In case you did not receive an event from CloudWatch Events or you received the unknown reason event, here are some common causes for this issue.

The **awslogs log driver isn't configured on your compute resources**

AWS Batch jobs send their log information to CloudWatch Logs. To enable this, you must configure your compute resources to use the `awslogs` log driver. Suppose that you base your compute resource AMI off of the Amazon ECS optimized AMI (or Amazon Linux). Then, this driver is registered by default with the `ecs-init` package. Now suppose that you use a different base AMI. Then, you must verify that the `awslogs` log driver is specified as an available log driver with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable when the Amazon ECS container agent is started. For more information, see Compute resource AMI specification and Creating a compute resource AMI.

**Insufficient resources**

If your job definitions specify more CPU or memory resources than your compute resources can allocate, then your jobs aren't ever placed. For example, suppose that your job specifies 4 GiB of memory, and your compute resources have less than that available. Then it's the case that the job can't be placed on those compute resources. In this case, you must reduce the specified memory in your job definition or add larger compute resources to your environment. Some memory is reserved for the Amazon ECS container agent and other critical system processes. For more information, see Compute Resource Memory Management.

**No internet access for compute resources**

Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses.

For more information about interface VPC endpoints, see Amazon ECS Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide.

If you do not have an interface VPC endpoint configured and your compute resources 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. For more information, see the section called “Create a VPC”.

**Amazon EC2 instance limit reached**

The number of Amazon EC2 instances that your account can launch in an AWS Region is determined by your EC2 instance quota. Certain instance types also have a per-instance-type
quota. For more information about your account's Amazon EC2 instance quota including how to request a limit increase, see Amazon EC2 Service Limits in the Amazon EC2 User Guide for Linux Instances.

Amazon ECS container agent isn't installed

The Amazon ECS container agent must be installed on the Amazon Machine Image (AMI) to let AWS Batch run jobs. The Amazon ECS container agent is installed by default on Amazon ECS optimized AMIs. For more information about the Amazon ECS container agent, see Amazon ECS container agent in the Amazon Elastic Container Service Developer Guide.

For more information, see Why is my AWS Batch job stuck in RUNNABLE status? in re:Post.

Spot Instances not tagged on creation

Spot Instance tagging for AWS Batch compute resources is supported as of October 25, 2017. Before, the recommended IAM managed policy (AmazonEC2SpotFleetRole) for the Amazon EC2 Spot Fleet role didn't contain permissions to tag Spot Instances at launch. The new recommended IAM managed policy is called AmazonEC2SpotFleetTaggingRole. It supports tagging Spot Instances at launch.

To fix Spot Instance tagging on creation, follow the following procedure to apply the current recommended IAM managed policy to your Amazon EC2 Spot Fleet role. That way, any future Spot Instances that are created with that role have permissions to apply instance tags when they're created.

To apply the current IAM managed policy to your Amazon EC2 Spot Fleet role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Roles, and choose your Amazon EC2 Spot Fleet role.
3. Choose Attach policy.
4. Select the AmazonEC2SpotFleetTaggingRole and choose Attach policy.
5. Choose your Amazon EC2 Spot Fleet role again to remove the previous policy.
6. Select the x to the right of the AmazonEC2SpotFleetRole policy, and choose Detach.
Spot Instances not scaling down

AWS Batch introduced the `AWSServiceRoleForBatch` service-linked role on March 10, 2021. If no role is specified in the `serviceRole` parameter of the compute environment, this service-linked role is used as the service role. However, suppose that the service-linked role is used in an EC2 Spot compute environment, but the Spot role used doesn't include the `AmazonEC2SpotFleetTaggingRole` managed policy. Then, the Spot Instance doesn't scale down. As a result, you will receive an error with the following message: "You are not authorized to perform this operation." Use the following steps to update the spot fleet role that you use in the `spotIamFleetRole` parameter. For more information, see [Using service-linked roles](https://docs.aws.amazon.com/iam/latest/userguide/service-linked-roles.html) and [Creating a role to delegate permissions to an AWS Service](https://docs.aws.amazon.com/ iam/latest/userguide/delegation-iam-roles.html) in the *IAM User Guide*.

**Topics**

- Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role in the AWS Management Console
- Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role with the AWS CLI

**Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role in the AWS Management Console**

To apply the current IAM managed policy to your Amazon EC2 Spot Fleet role

2. Choose **Roles**, and choose your Amazon EC2 Spot Fleet role.
3. Choose **Attach policy**.
4. Select the **AmazonEC2SpotFleetTaggingRole** and choose **Attach policy**.
5. Choose your Amazon EC2 Spot Fleet role again to remove the previous policy.
6. Select the `x` to the right of the **AmazonEC2SpotFleetRole** policy, and choose **Detach**.

**Attach AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role with the AWS CLI**

The example commands assume that your Amazon EC2 Spot Fleet role is named `AmazonEC2SpotFleetRole`. If your role uses a different name, adjust the commands to match.
To attach the AmazonEC2SpotFleetTaggingRole managed policy to your Spot Fleet role

1. To attach the **AmazonEC2SpotFleetTaggingRole** managed IAM policy to your *AmazonEC2SpotFleetRole* role, run the following command using the AWS CLI.

   ```bash
   $ aws iam attach-role-policy \\
   --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \\
   --role-name AmazonEC2SpotFleetRole
   ```

2. To detach the **AmazonEC2SpotFleetRole** managed IAM policy from your *AmazonEC2SpotFleetRole* role, run the following command using the AWS CLI.

   ```bash
   $ aws iam detach-role-policy \\
   --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetRole \\
   --role-name AmazonEC2SpotFleetRole
   ```

**Can't retrieve Secrets Manager secrets**

If you use an AMI with an Amazon ECS agent that's earlier than version 1.16.0-1, then you must use the Amazon 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 to a new container instance when you create that instance. Or, you can add it to an existing instance. If you add it to an existing instance, you must restart the ECS agent after you add it. For more information, see [Amazon ECS Container Agent Configuration](https://docs.aws.amazon.com/elasticcontainerservice/latest/dg/ecs-agent-configuration-statements.html) in the *Amazon Elastic Container Service Developer Guide*.

**Can't override job definition resource requirements**

The memory and vCPU overrides that are specified in the `memory` and `vcpus` members of the `containerOverrides` structure, which passed to `SubmitJob`, can't override the memory and vCPU requirements that are specified in the `resourceRequirements` structure in the job definition.

If you try to override these resource requirements, you might see the following error message:

"This value was submitted in a deprecated key and may conflict with the value provided by the job definition's resource requirements."
To correct this, specify the memory and vCPU requirements in the `resourceRequirements` member of the `containerOverrides`. For example, if your memory and vCPU overrides are specified in the following lines.

```json
"containerOverrides": {
  "memory": 8192,
  "vcpus": 4
}
```

Change them to the following:

```json
"containerOverrides": {
  "resourceRequirements": [
    {
      "type": "MEMORY",
      "value": "8192"
    },
    {
      "type": "VCPU",
      "value": "4"
    }
  ],
}
```

Do the same change to the memory and vCPU requirements that are specified in the `containerProperties` object in the job definition. For example, if your memory and vCPU requirements are specified in the following lines.

```json
{  
  "containerProperties": {
    "memory": 4096,
    "vcpus": 2,
  }
}
```

Change them to the following:

```json
"containerProperties": {
  "resourceRequirements": [
    {
      "type": "MEMORY",
      "value": "4096"
    }
  ]
}
```
Error message when you update the desiredvCpus setting

You see the following error message when you use the AWS Batch API to update the desired vCPUs (desiredvCpus) setting.

Manually scaling down compute environment is not supported. Disconnecting job queues from compute environment will cause it to scale-down to minvCpus.

This issue occurs if the updated desiredvCpus value is less than the current desiredvCpus value. When you update the desiredvCpus value, both of the following must be true:

- The desiredvCpus value must be between the minvCpus and maxvCpus values.
- The updated desiredvCpus value must be greater than or equal to the current desiredvCpus value.

AWS Batch on Amazon EKS

Topics

- INVALID compute environment
- AWS Batch on Amazon EKS job is stuck in RUNNABLE status
- Verify that the aws-auth ConfigMap is configured correctly
- RBAC permissions or bindings aren't configured properly

INVALID compute environment

It's possible that you might have incorrectly configured a managed compute environment. If you did, the compute environment enters an INVALID state and can't accept jobs for placement. The following sections describe the possible causes and how to troubleshoot based on the cause.
Unsupported Kubernetes version

You might see an error message that resembles the following when you use the CreateComputeEnvironment API operation or UpdateComputeEnvironment API operation to create or update a compute environment. This issue occurs if you specify an unsupported Kubernetes version in EC2Configuration.

At least one imageKubernetesVersion in EC2Configuration is not supported.

To resolve this issue, delete the compute environment and then re-create it with a supported Kubernetes version.

You can perform a minor version upgrade on your Amazon EKS cluster. For example, you can upgrade the cluster from 1.xx to 1.yy even if the minor version isn't supported.

However, the compute environment status might change to INVALID after a major version update. For example, if you perform a major version upgrade from 1.xx to 2.yy. If the major version isn't supported by AWS Batch, you see an error message that resembles the following.

reason=CLIENT_ERROR - ... EKS Cluster version [2.yy] is unsupported

To resolve this issue, specify a supported Kubernetes version when you use an API operation to create or update a compute environment.

AWS Batch on Amazon EKS currently supports the following Kubernetes versions:

- 1.29
- 1.28
- 1.27
- 1.26
- 1.25
- 1.24
- 1.23
Instance profile doesn't exist

If the specified instance profile does not exist, the AWS Batch on Amazon EKS compute environment status is changed to INVALID. You see an error set in the statusReason parameter that resembles the following.

CLIENT_ERROR - Instance profile arn:aws:iam::...:instance-profile/<name> does not exist

To resolve this issue, specify or create a working instance profile. For more information, see Amazon EKS node IAM role in the Amazon EKS User Guide.

Invalid Kubernetes namespace

If AWS Batch on Amazon EKS can't validate the namespace for the compute environment, the compute environment status is changed to INVALID. For example, this issue can occur if the namespace doesn't exist.

You see an error message set in the statusReason parameter that resembles the following.

CLIENT_ERROR - Unable to validate Kubernetes Namespace

This issue can occur if any of the following are true:

- The Kubernetes namespace string in the CreateComputeEnvironment call doesn't exist. For more information, see CreateComputeEnvironment.
- The required Role-Based Access Control (RBAC) permissions to manage the namespace are not configured correctly.
- AWS Batch doesn't have access to the Amazon EKS Kubernetes API server endpoint.

To resolve this issue, see Verify that the aws-auth ConfigMap is configured correctly. For more information, see Getting started with AWS Batch on Amazon EKS.

Deleted compute environment

Suppose that you delete an Amazon EKS cluster before you delete the attached AWS Batch on Amazon EKS compute environment. Then, the compute environment status is changed to INVALID. In this scenario, the compute environment doesn't work properly if you re-create the Amazon EKS cluster with the same name.
To resolve this issue, delete and then re-create the AWS Batch on Amazon EKS compute environment.

**Nodes don't join the Amazon EKS cluster**

AWS Batch on Amazon EKS scales down a compute environment if it determines that not all nodes joined the Amazon EKS cluster. When AWS Batch on Amazon EKS scales down the compute environment, the compute environment status is changed to INVALID.

*i Note*

AWS Batch doesn't change the compute environment status immediately so that you can debug the issue.

You see an error message set in the statusReason parameter that resembles ones of the following:

Your compute environment has been INVALIDATED and scaled down because none of the instances joined the underlying ECS Cluster. Common issues preventing instances joining are the following: VPC/Subnet configuration preventing communication to ECS, incorrect Instance Profile policy preventing authorization to ECS, or customized AMI or LaunchTemplate configurations affecting ECS agent.

Your compute environment has been INVALIDATED and scaled down because none of the nodes joined the underlying Amazon EKS Cluster. Common issues preventing nodes joining are the following: networking configuration preventing communication to Amazon EKS Cluster, incorrect Amazon EKS Instance Profile or Kubernetes RBAC policy preventing authorization to Amazon EKS Cluster, customized AMI or LaunchTemplate configurations affecting Amazon EKS/Kubernetes node bootstrap.

When using a default Amazon EKS AMI, the most common causes of this issue are the following:

- The instance role isn't configured correctly. For more information, see [Amazon EKS node IAM role](#) in the *Amazon EKS User Guide*.
- The subnets aren't configured correctly. For more information, see [Amazon EKS VPC and subnet requirements and considerations](#) in the *Amazon EKS User Guide*.
• The security group isn't configured correctly. For more information, see Amazon EKS security group requirements and considerations in the Amazon EKS User Guide.

Note
You may also see an error notification in the Personal Health Dashboard (PHD).

AWS Batch on Amazon EKS job is stuck in RUNNABLE status

An aws-auth ConfigMap is automatically created and applied to your cluster when you create a managed node group or a node group using eksctl. An aws-auth ConfigMap is initially created to allow nodes to join your cluster. However, you also use the aws-auth ConfigMap to add role-based access control (RBAC) access to users and roles.

To verify that the aws-auth ConfigMap is configured correctly:

1. Retrieve the mapped roles in the aws-auth ConfigMap:

   $ kubectl get configmap -n kube-system aws-auth -o yaml

2. Verify that the roleARN is configured as follows.

   rolearn: arn:aws:iam::aws_account_number:role/AWSServiceRoleForBatch

Note
You can also review the Amazon EKS control plane logs. For more information, see Amazon EKS control plane logging in the Amazon EKS User Guide.

To resolve an issue where a job is stuck in a RUNNABLE status, we recommend that you use kubectl to re-apply the manifest. For more information, see Step 1: Preparing your Amazon EKS cluster for AWS Batch. Or, you can use kubectl to manually edit the aws-auth ConfigMap. For more information, see Enabling IAM user and role access to your cluster in the Amazon EKS User Guide.
Verify that the aws-auth ConfigMap is configured correctly

To verify that the aws-auth ConfigMap is configured correctly:

1. Retrieve the mapped roles in the aws-auth ConfigMap.

   ```bash
   $ kubectl get configmap -n kube-system aws-auth -o yaml
   ```

2. Verify that the roleARN is configured as follows.

   ```bash
   roleARN: arn:aws:iam::aws_account_number:role/AWSServiceRoleForBatch
   ```

   **Note**
   
   The path aws-service-role/batch.amazonaws.com/ has been removed from the ARN of the service-linked role. This is because of an issue with the aws-auth configuration map. For more information, see Roles with paths do not work when the path is included in their ARN in the aws-auth-configmap.

   **Note**
   
   You can also review the Amazon EKS control plane logs. For more information, see Amazon EKS control plane logging in the Amazon EKS User Guide.

To resolve an issue where a job is stuck in a RUNNABLE status, we recommend that you use kubectl to re-apply the manifest. For more information, see Step 1: Preparing your Amazon EKS cluster for AWS Batch. Or, you can use kubectl to manually edit the aws-auth ConfigMap. For more information, see Enabling IAM user and role access to your cluster in the Amazon EKS User Guide.

**RBAC permissions or bindings aren't configured properly**

If you experience any RBAC permissions or binding issues, verify that the aws-batch Kubernetes role can access the Kubernetes namespace:

```bash
$ kubectl get namespace namespace --as=aws-batch
```
$ kubectl auth can-i get ns --as=aws-batch

You can also use the `kubectl describe` command to view the authorizations for a cluster role or Kubernetes namespace.

$ kubectl describe clusterrole aws-batch-cluster-role

The following is example output.

<table>
<thead>
<tr>
<th>Name:</th>
<th>aws-batch-cluster-role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labels:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Annotations:</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>PolicyRule:</td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td></td>
</tr>
<tr>
<td>Verbs</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>configmaps</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>nodes</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>pods</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>daemonsets.apps</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>deployments.apps</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>replicaset.apps</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>statefulsets.apps</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list watch]</td>
<td>[ ]</td>
</tr>
<tr>
<td>clusterrolebindings.rbac.authorization.k8s.io</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list]</td>
<td>[ ]</td>
</tr>
<tr>
<td>clusterroles.rbac.authorization.k8s.io</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get list]</td>
<td>[ ]</td>
</tr>
<tr>
<td>namespaces</td>
<td>[ ]</td>
</tr>
<tr>
<td>[get]</td>
<td></td>
</tr>
</tbody>
</table>

$ kubectl describe role aws-batch-compute-environment-role -n my-aws-batch-namespace

The following is example output.

RBAC permissions or bindings aren't configured properly
To resolve this issue, re-apply the RBAC permissions and rolebinding commands. For more information, see [Step 1: Preparing your Amazon EKS cluster for AWS Batch](#).
Best Practices for AWS Batch

You can use AWS Batch to run a variety of demanding computational workloads at scale without managing a complex architecture. AWS Batch jobs can be used in a wide range of use cases in areas such as epidemiology, gaming, and machine learning.

This topic covers the best practices to consider while using AWS Batch and guidance on how to run and optimize your workloads when using AWS Batch.

Topics

- When to use AWS Batch
- Checklist to run at scale
- Optimize containers and AMIs
- Choose the right compute environment resource
- Amazon EC2 On-Demand or Amazon EC2 Spot
- Use Amazon EC2 Spot best practices for AWS Batch
- Common errors and troubleshooting

When to use AWS Batch

AWS Batch runs jobs at scale and at low cost, and provides queuing services and cost-optimized scaling. However, not every workload is suitable to be run using AWS Batch.

- **Short jobs** – If a job runs for only a few seconds, the overhead to schedule the batch job might take longer than the runtime of the job itself. As a workaround, binpack your tasks together before you submit them in AWS Batch. Then, configure your AWS Batch jobs to iterate over the tasks. For example, stage the individual task arguments into an Amazon DynamoDB table or as a file in an Amazon S3 bucket. Consider grouping tasks so the jobs run 3-5 minutes each. After you binpack the jobs, loop through your task groups within your AWS Batch job.

- **Jobs that must be run immediately** – AWS Batch can process jobs quickly. However, AWS Batch is a scheduler and optimizes for cost performance, job priority, and throughput. AWS Batch might require time to process your requests. If you need a response in under a few seconds, then a service-based approach using Amazon ECS or Amazon EKS is more suitable.
Checklist to run at scale

Before you run a large workload on 50 thousand or more vCPUs, consider the following checklist.

⚠️ Note
If you plan to run a large workload on a million or more vCPUs or need guidance running at large scale, contact your AWS team.

- **Check your Amazon EC2 quotas** – Check your Amazon EC2 quotas (also known as limits) in the Service Quotas panel of the AWS Management Console. If necessary, request a quota increase for your peak number of Amazon EC2 instances. Remember that Amazon EC2 Spot and Amazon On-Demand instances have separate quotas. For more information, see [Getting started with Service Quotas](#).

- **Verify your Amazon Elastic Block Store quota for each Region** – Each instance uses a GP2 or GP3 volume for the operating system. By default, the quota for each AWS Region is 300 TiB. However, each instance uses counts as part of this quota. So, make sure to factor this in when you verify your Amazon Elastic Block Store quota for each Region. If your quota is reached, you can’t create more instances. For more information, see [Amazon Elastic Block Store endpoints and quotas](#).

- **Use Amazon S3 for storage** – Amazon S3 provides high throughput and helps to eliminate the guesswork on how much storage to provision based on the number of jobs and instances in each Availability Zone. For more information, see [Best practices design patterns: optimizing Amazon S3 performance](#).

- **Scale gradually to identify bottlenecks early** – For a job that runs on a million or more vCPUs, start lower and gradually increase so that you can identify bottlenecks early. For example, start by running on 50 thousand vCPUs. Then, increase the count to 200 thousand vCPUs, and then 500 thousand vCPUs, and so on. In other words, continue to gradually increase the vCPU count until you reach the desired number of vCPUs.

- **Monitor to identify potential issues early** – To avoid potential breaks and issues when running at scale, make sure to monitor both your application and architecture. Breaks might occur even when scaling from 1 thousand to 5 thousand vCPUs. You can use Amazon CloudWatch Logs to review log data or use CloudWatch Embedded Metrics using a client library. For more information, see [CloudWatch Logs agent reference](#) and [aws-embedded-metrics](#).
Optimize containers and AMIs

Container size and structure are important for the first set of jobs that you run. This is especially true if the container is larger than 4 GB. Container images are built in layers. The layers are retrieved in parallel by Docker using three concurrent threads. You can increase the number of concurrent threads using the `max-concurrent-downloads` parameter. For more information, see the [Dockerd documentation](#).

Although you can use larger containers, we recommend that you optimize container structure and size for faster startup times.

- **Smaller containers are fetched faster** – Smaller containers can lead to faster application start times. To decrease container size, offload libraries or files that are updated infrequently to the Amazon Machine Image (AMI). You can also use bind mounts to give access to your containers. For more information, see [Bind mounts](#).

- **Create layers that are even in size and break up large layers** – Each layer is retrieved by one thread. So, a large layer might significantly impact your job startup time. We recommend a maximum layer size of 2 GB as a good tradeoff between larger container size and faster startup times. You can run the `docker history your_image_id` command to check your container image structure and layer size. For more information, see the [Docker documentation](#).

- **Use Amazon Elastic Container Registry as your container repository** – When you run thousands of jobs in parallel, a self-managed repository can fail or throttle throughput. Amazon ECR works at scale and can handle workloads with up to over a million vCPUs.
Choose the right compute environment resource

AWS Fargate requires less initial setup and configuration than Amazon EC2 and is likely easier to use, particularly if it's your first-time. With Fargate, you don't need to manage servers, handle capacity planning, or isolate container workloads for security.

If you have the following requirements, we recommend you use Fargate instances:

- Your jobs must start quickly, specifically less than 30 seconds.
- The requirements of your jobs are 16 vCPUs or less, no GPUs, and 120 GiB of memory or less.

For more information, see When to use Fargate.

If you have the following requirements, we recommend that you use Amazon EC2 instances:

- You require increased control over the instance selection or require using specific instance types.
- Your jobs require resources that AWS Fargate can't provide, such as GPUs, more memory, a custom AMI, or the Amazon Elastic Fabric Adapter.
- You require a high level of throughput or concurrency.
- You need to customize your AMI, Amazon EC2 Launch Template, or access to special Linux parameters.

With Amazon EC2, you can more finely tune your workload to your specific requirements and run at scale if needed.

Amazon EC2 On-Demand or Amazon EC2 Spot

Most AWS Batch customers use Amazon EC2 Spot instances because of the savings over On-Demand instances. However, if your workload runs for multiple hours and can't be interrupted, On-Demand instances might be more suitable for you. You can always try Spot instances first and switch to On-Demand if necessary.

If you have the following requirements and expectations, use Amazon EC2 On-Demand instances:

- The runtime of your jobs is more than an hour, and you can't tolerate interruptions to your workload.
• You have a strict SLO (service-level objective) for your overall workload and can’t increase computational time.

• The instances that you require are more likely to see interruptions.

If you have the following requirements and expectations, use Amazon EC2 Spot instances:

• The runtime for your jobs is typically 30 minutes or less.

• You can tolerate potential interruptions and job rescheduling as a part of your workload. For more information, see Spot Instance advisor.

• Long running jobs can be restarted from a checkpoint if interrupted.

You can mix both purchasing models by submitting on Spot instance first and then use On-Demand instance as a fallback option. For example, submit your jobs on a queue that’s connected to compute environments that are running on Amazon EC2 Spot instances. If a job gets interrupted, catch the event from Amazon EventBridge and correlate it to a Spot instance reclamation. Then, resubmit the job to an On-Demand queue using an AWS Lambda function or AWS Step Functions. For more information, see Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events, Best practices for handling Amazon EC2 Spot Instance interruptions and Manage AWS Batch with Step Functions.

**Important**

Use different instance types, sizes, and Availability Zones for your On-Demand compute environment to maintain Amazon EC2 Spot instance pool availability and decrease the interruption rate.

### Use Amazon EC2 Spot best practices for AWS Batch

When you choose Amazon Elastic Compute Cloud (EC2) Spot instances, you likely can optimize your workflow to save costs, sometimes significantly. For more information, see Best practices for Amazon EC2 Spot.

To optimize your workflow to save costs, consider the following Amazon EC2 Spot best practices for AWS Batch:
• **Choose the SPOT_CAPACITY_OPTIMIZED allocation strategy** – AWS Batch chooses Amazon EC2 instances from the deepest Amazon EC2 Spot capacity pools. If you’re concerned about interruptions, this is a suitable choice. For more information, see Allocation strategies.

• **Diversify instance types** – To diversify your instance types, consider compatible sizes and families, then let AWS Batch choose based on price or availability. For example, consider c5.24xlarge as an alternative to c5.12xlarge or c5a, c5n, c5d, m5, and m5d families. For more information, see Be flexible about instance types and Availability Zones.

• **Reduce job runtime or checkpoint** – We advise against running jobs that take an hour or more when using Amazon EC2 Spot instances to avoid interruptions. If you divide or checkpoint your jobs into smaller parts that consist of 30 minutes or less, you can significantly reduce the possibility of interruptions.

• **Use automated retries** – To avoid disruptions to AWS Batch jobs, set automated retries for jobs. Batch jobs can be disrupted for any of the following reasons: a non-zero exit code is returned, a service error occurs, or an instance reclamation occurs. You can set up to 10 automated retries. For a start, we recommend that you set at least 1-3 automated retries. For information about tracking Amazon EC2 Spot interruptions, see Spot Interruption Dashboard.

For AWS Batch, if you set the retry parameter, the job is placed at the front of the job queue. That is, the job is given priority. When you create the job definition or you submit the job in the AWS CLI, you can configure a retry strategy. For more information, see submit-job.

$$
aws batch submit-job --job-name MyJob
   --job-queue MyJQ
   --job-definition MyJD
   --retry-strategy attempts=2
$$

• **Use custom retries** – You can configure a job retry strategy to a specific application exit code or instance reclamation. In the following example, if the host causes the failure, the job can be retried up to five times. However, if the job fails for a different reason, the job exits and the status is set to FAILED.

```
"retryStrategy": {
   "attempts": 5,
   "evaluateOnExit": [
   {
      "onStatusReason": "Host EC2*",
      "action": "RETRY"
   },
```

Use Amazon EC2 Spot best practices for AWS Batch
Common errors and troubleshooting

Errors in AWS Batch often occur at the application level or are caused by instance configurations that don’t meet your specific job requirements. Other issues include jobs getting stuck in the RUNNABLE status or compute environments getting stuck in an INVALID state. For more information about troubleshooting jobs getting stuck in RUNNABLE status, see Jobs stuck in a RUNNABLE status. For information about troubleshooting compute environments in an INVALID state, see INVALID compute environment.

- **Check Amazon EC2 Spot vCPU quotas** – Verify that your current service quotas meet the job requirements. For example, suppose that your current service quota is 256 vCPUs and the job requires 10,000 vCPUs. Then, the service quota doesn’t meet the job requirement. For more information and troubleshooting instructions, see Amazon EC2 service quotas and How do I increase the service quota of my Amazon EC2 resources?

- **Jobs fail before the application runs** – Some jobs might fail because of a DockerTimeoutError error or a CannotPullContainerError error. For troubleshooting information, see How do I resolve the "DockerTimeoutError" error in AWS Batch?

- **Insufficient IP addresses** – The number of IP addresses in your VPC and subnets can limit the number of instances that you can create. Use Classless Inter-Domain Routings (CIDRs) to provide more IP addresses than are required to run your workloads. If necessary, you can also build a dedicated VPC with a large address space. For example, you can create a VPC with multiple CIDRs in 10.x.0.0/16 and a subnet in every Availability Zone with a CIDR of 10.x.y.0/17. In this example, x is between 1-4 and y is either 0 or 128. This configuration provides 36,000 IP addresses in every subnet.
• **Verify that instances are registered with Amazon EC2** – If you see your instances in the Amazon EC2 console, but no Amazon Elastic Container Service container instances in your Amazon ECS cluster, the Amazon ECS agent might not be installed on an Amazon Machine Image (AMI). The Amazon ECS Agent, the Amazon EC2 Data in your AMI, or the launch template might also not be configured correctly. To isolate the root cause, create a separate Amazon EC2 instance or connect to an existing instance using SSH. For more information, see Amazon ECS container agent configuration, Amazon ECS Log File Locations, and Compute resource AMIs.

• **Review the AWS Dashboard** – Review the AWS Dashboard to verify that the expected job states and that the compute environment scales as expected. You can also review the job logs in CloudWatch.
• **Verify that your instance is created** – If an instance is created, it means that your compute environment scaled as expected. If your instances aren’t created, find the associated subnets in your compute environment to change. For more information, see [Verify a scaling activity for an Auto Scaling group](#).

We also recommend that you verify that your instances can fulfill your related job requirements. For example, a job might require 1 TiB of memory, but the compute environment uses a C5 instance type that's limited to 192 GB of memory.

• **Verify that your instances are being requested by AWS Batch** – Check Auto Scaling group history to verify that your instances are being requested by AWS Batch. This is an indication of how Amazon EC2 tries to acquire instances. If you receive an error stating the Amazon EC2 Spot can’t acquire an instance in a specific Availability Zone, this might be because the Availability Zone doesn’t offer a specific instance family.

• **Verify that instances register with Amazon ECS** – If you see instances in the Amazon EC2 console, but no Amazon ECS container instances in your Amazon ECS cluster, the Amazon ECS agent might not be installed on the Amazon Machine Image (AMI). Moreover, the Amazon ECS Agent, the Amazon EC2 Data in your AMI, or the launch template might not be configured correctly. To isolate the root cause, create a separate Amazon EC2 instance or connect to an existing instance using SSH. For more information, see [CloudWatch agent configuration file: Logs section](#), [Amazon ECS Log File Locations](#), and [Compute resource AMIs](#).

• **Open a support ticket** – If you're still experiencing issues after some troubleshooting and have a support plan, open a support ticket. In the support ticket, make sure to include information about the issue, workload specifics, the configuration, and test results. For more information, see [Compare AWS Support Plans](#).

• **Review the AWS Batch and HPC forums** – For more information, see the [AWS Batch](#) and [HPC](#) forums.

• **Review the AWS Batch Runtime Monitoring Dashboard** – This dashboard uses a serverless architecture to capture events from Amazon ECS, AWS Batch, and Amazon EC2 to provide insights into jobs and instances. For more information, see [AWS Batch Runtime Monitoring Dashboards Solution](#).
## Document history

The following table describes the important changes to the documentation since the initial release of AWS Batch. We also update the documentation frequently to address the feedback that you send us.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Updated AWS Batch supported Amazon EKS versions</strong></td>
<td>Updated the Amazon EKS versions that AWS Batch supports to remove version 1.22.</td>
<td>March 11, 2024</td>
</tr>
<tr>
<td><strong>Updated AWS Batch supported Amazon EKS versions</strong></td>
<td>Updated the Amazon EKS versions that AWS Batch supports to include version 1.29.</td>
<td>February 29, 2024</td>
</tr>
<tr>
<td><strong>Automated job retries</strong></td>
<td>Corrected the code sample.</td>
<td>February 29, 2024</td>
</tr>
<tr>
<td><strong>Adds support for multi-container jobs for AWS Batch</strong></td>
<td>Adds support for multi-container jobs for AWS Batch for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, and AWS Fargate.</td>
<td>February 28, 2024</td>
</tr>
<tr>
<td><strong>Updated AWS Batch supported Amazon EKS versions</strong></td>
<td>Updated the Amazon EKS versions that AWS Batch supports to include version 1.28.</td>
<td>January 27, 2024</td>
</tr>
<tr>
<td><strong>Updated BatchServiceRolePolicy and AWSBatchServiceRole</strong></td>
<td>Updated to add support for describing Spot Fleet request history and</td>
<td>December 5, 2023</td>
</tr>
</tbody>
</table>
Amazon EC2 Auto Scaling activities.

**AWSBatchServiceRole**

Updated to add statement IDs, grant AWS Batch permissions to `ec2:DescribeSpotFleetRequestHistory` and `autoscaling:DescribeScalingActivities`.

<table>
<thead>
<tr>
<th><strong>AWS Batch on Amazon EKS</strong></th>
<th>AWS Batch adds support for running jobs on Amazon EKS clusters.</th>
<th>October 25, 2022</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cross-service confused deputy prevention for AWS Batch</strong></td>
<td>AWS Batch now provides a workaround for the confused deputy security issue, which arises when an entity (a service or an account) is coerced by a different entity to perform an action.</td>
<td>June 6, 2022</td>
</tr>
<tr>
<td><strong>Interface VPC Endpoints (AWS PrivateLink)</strong></td>
<td>Added support for configuring interface VPC endpoints powered by AWS PrivateLink. This means you can create a private connection between your VPC and AWS Batch without requiring access through a NAT instance, a VPN connection, or AWS Direct Connect.</td>
<td>April 15, 2022</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Enhanced compute environment updates</td>
<td>AWS Batch enhanced support updates to compute environments.</td>
<td>April 14, 2022</td>
</tr>
<tr>
<td>AWS managed policy updates</td>
<td>AWS Batch updated existing managed policies.</td>
<td>December 6, 2021</td>
</tr>
<tr>
<td>Fair share scheduling</td>
<td>AWS Batch adds support for adding scheduling policies to job queues.</td>
<td>November 9, 2021</td>
</tr>
<tr>
<td>Amazon EFS</td>
<td>AWS Batch adds support for adding Amazon EFS file systems to your job definitions.</td>
<td>April 1, 2021</td>
</tr>
<tr>
<td>Added service-linked role</td>
<td>AWS Batch adds the AWSServiceRoleForBatch service-linked role.</td>
<td>March 10, 2021</td>
</tr>
<tr>
<td>AWS Fargate support</td>
<td>AWS Batch adds support for running jobs on Fargate resources.</td>
<td>December 3, 2020</td>
</tr>
<tr>
<td>Amazon Linux 2 support</td>
<td>AWS Batch adds support for automatic selection of Amazon Linux 2 AMIs in the compute environment using the EC2 Configuration parameters.</td>
<td>November 24, 2020</td>
</tr>
<tr>
<td>Enhanced retry strategy</td>
<td>AWS Batch enhances the retry strategy for jobs. Now jobs can be retried or stop further retries by matching the ExitCode, Reason, or StatusReason of a job with patterns.</td>
<td>October 20, 2020</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Resource tagging</td>
<td>AWS Batch adds support for adding metadata tags to your compute environments, job definitions, job queues, and jobs.</td>
<td>October 7, 2020</td>
</tr>
<tr>
<td>Secrets</td>
<td>AWS Batch adds support for passing secrets to jobs.</td>
<td>October 1, 2020</td>
</tr>
<tr>
<td>Logging</td>
<td>AWS Batch adds support for specifying additional log drivers for jobs.</td>
<td>October 1, 2020</td>
</tr>
<tr>
<td>Allocation strategies</td>
<td>AWS Batch adds support for multiple strategies to choose instance types.</td>
<td>October 16, 2019</td>
</tr>
<tr>
<td>EFA support</td>
<td>AWS Batch adds support for Elastic Fabric Adapter (EFA) devices.</td>
<td>August 2, 2019</td>
</tr>
<tr>
<td>GPU scheduling</td>
<td>AWS Batch adds GPU scheduling. With this feature, you can specify the number of GPUs each job requires, and AWS Batch scales up instances accordingly.</td>
<td>April 4, 2019</td>
</tr>
<tr>
<td>Multi-node parallel jobs</td>
<td>AWS Batch adds support for multi-node parallel jobs. You can use this feature run single jobs that span over multiple Amazon EC2 instances.</td>
<td>November 19, 2018</td>
</tr>
<tr>
<td>Resource-level permissions</td>
<td>AWS Batch supports resource-level permissions on several API operations.</td>
<td>November 12, 2018</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><strong>Amazon EC2 Launch template support</strong></td>
<td>AWS Batch adds support for using launch templates with compute environments.</td>
<td>November 12, 2018</td>
</tr>
<tr>
<td><strong>AWS Batch job timeouts</strong></td>
<td>AWS Batch adds support for job timeout. With this support, you can configure a specific timeout duration for your jobs so that if a job runs longer than they should, AWS Batch terminates the job.</td>
<td>April 5, 2018</td>
</tr>
<tr>
<td><strong>AWS Batch jobs as EventBridge targets</strong></td>
<td>AWS Batch jobs are made available as EventBridge targets. By creating simple rules, you can match events and submit AWS Batch jobs in response to them.</td>
<td>March 1, 2018</td>
</tr>
<tr>
<td><strong>CloudTrail auditing for AWS Batch</strong></td>
<td>CloudTrail can audit calls made to AWS Batch API actions.</td>
<td>January 10, 2018</td>
</tr>
<tr>
<td><strong>Array jobs</strong></td>
<td>AWS Batch adds support for array jobs. You can use array jobs for parameter sweep and Monte Carlo workloads.</td>
<td>November 28, 2017</td>
</tr>
<tr>
<td><strong>Expanded AWS Batch tagging</strong></td>
<td>AWS Batch expands support for the tagging function. You can use this function to specify tags for Amazon EC2 Spot Instances launched within managed compute environments.</td>
<td>October 26, 2017</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>AWS Batch event stream for EventBridge</strong></td>
<td>AWS Batch adds the event stream for EventBridge. You can use AWS Batch event stream to receive near real-time notifications regarding the state of jobs that are submitted to your job queues.</td>
<td>October 24, 2017</td>
</tr>
<tr>
<td><strong>Automated job retries</strong></td>
<td>AWS Batch adds support for job retries. With this update, you can apply a retry strategy to your jobs and job definitions that allows your jobs to be automatically retried if they fail.</td>
<td>March 28, 2017</td>
</tr>
<tr>
<td><strong>AWS Batch general availability</strong></td>
<td>AWS Batch is introduced, designed as a means for you to run batch computing workloads on the AWS Cloud.</td>
<td>January 5, 2017</td>
</tr>
</tbody>
</table>