User Guide

AWS Batch

AWS Batch: User Guide

Table of Contents

What is AWS Batch? 1
Are you a first-time AWS Batch user? 3
Related services 3
Accessing AWS Batch 3
Components of AWS Batch 4
Compute environment 4
Job queues 5
Job definitions 5
Jobs 5
Scheduling policy 5
Consumable resources 6
Service Environment 6
Service job 6
Setting up AWS Batch 7
Create IAM account and administrative user 7
Sign up for an AWS account 7
Create a user with administrative access 8
Create IAM roles 9
Create a key pair 10
Create a VPC 12
Create a security group 13
Install the AWS CLI 15
Getting started tutorials 16
Getting started with Amazon EC2 using the Wizard 16
Overview 16
Prerequisites 17
Step 1: Create a compute environment 17
Step 2: Create a job queue 18
Step 3: Create a job definition 18
Step 4: Create a job 19
Step 5: Review and create 19
Step 6: View the Job's output 19
Step 7: Clean up your tutorial resources 20
Additional resources 20

- Getting started with Fargate orchestration using the Wizard 21
  - Overview 21
  - Prerequisites 22
  - Step 1: Create a compute environment 22
  - Step 2: Create a job queue 23
  - Step 3: Create a job definition 23
  - Step 4: Create a job 24
  - Step 5: Review and create 24
  - Step 6: View the Job's output 24
  - Step 7: Clean up your tutorial resources 25
  - Additional resources 25
- Getting started with AWS Batch and Fargate using the AWS CLI 26
  - Prerequisites 26
  - Create an IAM execution role 27
  - Create a compute environment 28
  - Create a job queue 29
  - Create a job definition 30
  - Submit and monitor a job 31
  - View job output 32
  - Clean up resources 32
  - Going to production 34
  - Next steps 34
- Getting started with Amazon EKS 35
  - Overview 36
  - Prerequisites 37
  - Step 1: Create your Amazon EKS cluster for AWS Batch 38
  - Step 2: Prepare your Amazon EKS cluster for AWS Batch 38
  - Step 3: Create an Amazon EKS compute environment 42
  - Step 4: Create a job queue and attach the compute environment 44
  - Step 5: Create a job definition 45
  - Step 6: Submit a job 46
  - Step 7: View the Job's output 46
  - Step 8: (Optional) Submit a job with overrides 47
  - Step 9: Clean up your tutorial resources 48
  - Additional resources 48
- Getting started with AWS Batch on Amazon EKS Private Clusters 49

    - Overview ... 36
    - Prerequisites ... 51
    - Step 1: Create your EKS cluster for AWS Batch ... 52
    - Step 2: Prepare your EKS cluster for AWS Batch ... 54
    - Step 3: Create an Amazon EKS compute environment ... 57
    - Step 4: Create a job queue and attach the compute environment ... 59
    - Step 5: Create an Amazon ECR with pull through cache ... 60
    - Step 6: Register a job definition ... 61
    - Step 7: Submit a job to run ... 62
    - Step 8: View the Job's output ... 62
    - Step 9: (Optional) Submit a job with overrides ... 62
    - Step 10: Clean up your tutorial resources ... 63
    - Additional resources ... 48
    - Troubleshooting ... 64
  - Getting started with AWS Batch on SageMaker AI ... 65
    - Overview ... 65
    - Prerequisites ... 66
    - Step 1: Create a SageMaker AI execution role ... 66
    - Step 2: Create your service environment ... 68
    - Step 3: Create your SageMaker job queue ... 69
    - Step 4: Create and submit a training job ... 71
    - Step 5: Monitor job status ... 72
    - Step 6: View job output ... 75
    - Step 7: Clean up your tutorial resources ... 77
    - Additional resources ... 78
- **The AWS Batch widget dashboard ... 79**
  - Add the Single job queue widget ... 79
  - How to add the CloudWatch Container Insights widget ... 80
  - Add the Job logs widget ... 80
- **Compute environments for AWS Batch ... 81**
  - Managed compute environments ... 81
    - Consideration when creating multi-node parallel jobs ... 84
  - Unmanaged compute environments ... 84
  - Create a compute environment ... 86
    - Tutorial: Create a managed compute environment using Fargate resources ... 86
    - Tutorial: Create a managed compute environment using Amazon EC2 resources ... 88

  - Tutorial: Create an unmanaged compute environment using Amazon EC2 resources 97
  - Tutorial: Create a managed compute environment using Amazon EKS resources 98
  - Tutorial: Create an unmanaged compute environment using Amazon EKS resources 103
  - Resource: Compute environment template 104
  - Instance type compute table 106
- Update a compute environment 108
  - Compute environment update strategies 109
  - Choosing the right update strategy 110
  - Perform scaling updates 112
  - Perform infrastructure updates 114
  - Perform Blue/green updates 119
- Compute resource AMIs 124
  - Compute resource AMI specification 126
  - Tutorial: Create a compute resource AMI 127
  - Use a GPU workload AMI 131
  - Amazon Linux deprecation 136
  - Amazon EKS Amazon Linux 2 AMI deprecation 137
  - Amazon ECS Amazon Linux 2 AMI deprecation 137
- Use Amazon EC2 launch templates 138
  - Default and override launch templates 140
  - Amazon EC2 user data in launch templates 141
  - Reference: Launch template examples 142
- Instance Metadata Service (IMDS) configuration 145
  - Configuration scenarios 145
- EC2 configurations 146
  - How to migrate from ECS AL2 to ECS AL2023 147
- Instance type allocation strategies 149
- Memory management 151
  - Reserve system memory 152
  - Tutorial: View compute resource memory 152
  - Memory and vCPU considerations for AWS Batch on Amazon EKS 153
- Fargate compute environments 158
  - When to use Fargate 158
  - Job definitions on Fargate 159
  - Job queues on Fargate 161
  - Compute environments on Fargate 161

Amazon EKS compute environments 162
  Amazon EKS 165
  Amazon EKS default AMI 165
  Mixed AMI environments 167
  Supported Kubernetes versions 168
  Update the Kubernetes version of the compute environment 168
  Shared responsibility of the Kubernetes nodes 169
  Run a DaemonSet on AWS Batch managed nodes 170
  Customize Amazon EKS launch templates 170
  How to upgrade from EKS AL2 to EKS AL2023 175

Service environments 177
What are service environments 177
  How service environments work with other AWS Batch components 178
  Best practices for service environments 178
Service environment states and lifecycle 179
  Service environment state definitions 180
Create a service environment 182
  Prerequisites 182
Update a service environment 183
Delete a service environment 185
  Deletion prerequisites 186

Job queues 188
Create a job queue 188
  Create an Amazon EC2 job queue 189
  Create a Fargate job queue 190
  Create an Amazon EKS job queue 191
  Create a SageMaker job queue 192
  Job queue template 195
View job queue status 196
  View job queue information 196
  Monitor service jobs 197
Delete a job queue 201
Fair-share scheduling policies 202
  Use share identifiers 202
  Use scheduling policies 203
  Use fair-share scheduling 203

    - Tutorial: Create a scheduling policy ... 204
    - Reference: Scheduling policy template ... 206
  - Resource-aware scheduling ... 207
    - Create consumable resources ... 208
    - Specify resources for a job ... 208
    - Check resource usage ... 210
    - Update in-use resource quantities ... 212
    - Find jobs requiring a consumable resource ... 213
    - Delete a consumable resource ... 213
- **Job definitions ... 215**
  - Create a single-node job definition ... 215
    - Tutorial: Create a single-node job definition on Amazon EC2 resources ... 216
    - Create a single-node job definition on Fargate resources ... 221
    - Create a single-node job definition on Amazon EKS resources ... 227
    - Create a single-node job definition with multiple containers on Amazon EC2 resources ... 232
  - Create a multi-node parallel job definition ... 238
    - Tutorial: Create a multi-node parallel job definition on Amazon EC2 resources ... 239
  - Reference: Job definition template using ContainerProperties ... 246
    - Job definition parameters for ContainerProperties ... 253
  - Create job definitions using EcsProperties ... 297
    - `ContainerProperties` versus `EcsProperties` job definitions ... 297
    - General changes to the AWS Batch APIs ... 298
    - Multi-container job definitions for Amazon ECS ... 299
    - Multi-container job definitions for Amazon EKS ... 300
    - Reference: AWS Batch job scenarios using EcsProperties ... 301
  - Use the awslogs log driver ... 307
    - awslogs log driver options in the AWS Batch JobDefiniton data type ... 308
    - Specify a log configuration in your job definition ... 310
  - Specify sensitive data ... 312
    - Use Secrets Manager ... 312
    - Use Systems Manager Parameter Store ... 320
  - Private registry authentication for jobs ... 324
    - Required IAM permissions for private registry authentication ... 325
    - Tutorial: Create a secret for private registry authentication ... 326
  - Amazon EFS volumes ... 327
    - Amazon EFS volume considerations ... 327

    - Use Amazon EFS access points 328
    - Specify an Amazon EFS file system in your job definition 329
  - Job definition examples 332
    - Environment variables 332
    - Parameter substitution 333
    - Test GPU functionality 334
    - Multi-node parallel job 335
- **Jobs 337**
  - Tutorial: submit a job 337
  - Service jobs 340
    - Service job payloads 341
    - Submit a service job 342
    - Service job status 344
    - Service job retry strategies 346
    - Monitor service jobs in the queue 348
  - Job states 349
  - Job environment variables 352
  - Automated job retries 354
  - Job dependencies 355
  - Job timeouts 356
  - Amazon EKS jobs 357
    - Tutorial: Map a running job to a pod and a node 358
    - Tutorial: Map a running pod back to its job 359
  - Multi-node parallel jobs 360
    - Environment variables 361
    - Node groups 362
    - Job lifecycle 363
    - Compute environment considerations 363
  - Multi-node parallel jobs on Amazon EKS 365
    - Running MNP jobs 365
    - Create an Amazon EKS MNP job definition 367
    - Submit an Amazon EKS MNP job 369
    - Override an Amazon EKS MNP job definition 370
  - Array jobs 370
    - Example of an array job workflow 373
    - Using array job index 377

- Run GPU jobs ... 382
  - Create a GPU-based Kubernetes cluster on Amazon EKS ... 387
  - Create an Amazon EKS GPU job definition ... 389
  - Run a GPU job in your Amazon EKS cluster ... 390
- View jobs in a job queue ... 390
- Search for jobs in a job queue ... 391
  - Search AWS Batch jobs (AWS Console) ... 391
  - Search and filter AWS Batch jobs (AWS CLI) ... 393
- Networking modes for AWS Batch jobs ... 393
- View job logs in CloudWatch Logs ... 395
- Review AWS Batch job information ... 396

Security in AWS Batch ... 397

- Identity and Access Management ... 398
  - Audience ... 398
  - Authenticating with identities ... 398
  - Managing access using policies ... 400
  - How AWS Batch works with IAM ... 401
  - Identity-based policy examples ... 406
  - AWS managed policies ... 408
- IAM policies, roles, and permissions ... 413
  - IAM policy structure ... 414
  - Resource: Example policies ... 417
  - Resource: AWS Batch managed policy ... 427
- AWS Batch IAM execution role ... 428
  - Supported resource-level permissions ... 429
  - Tutorial: Create the IAM execution role ... 429
  - Tutorial: Check the IAM execution role ... 430
  - Using service-linked roles ... 431
  - Amazon ECS instance role ... 445
  - Amazon EC2 spot fleet role ... 448
  - EventBridge IAM role ... 451
- Create a virtual private cloud ... 452
  - Create a VPC ... 453
  - Next steps ... 453
- VPC endpoints ... 454
  - Considerations ... 454

    - Create an interface endpoint 455
    - Create an endpoint policy 456
  - Compliance validation 457
  - Infrastructure security 458
  - Cross-service confused deputy prevention 458
    - Example: Role for accessing only one compute environment 459
    - Example: Role for accessing multiple compute environments 460
  - CloudTrail 461
    - AWS Batch information in CloudTrail 461
    - Reference: Understanding AWS Batch log file entries 462
  - Troubleshoot AWS Batch IAM 464
    - I am not authorized to perform an action in AWS Batch 464
    - I am not authorized to perform iam:PassRole 464
    - I want to allow people outside of my AWS account to access my AWS Batch resources 465
- **AWS Step Functions 466**
  - Tutorial: View state machine details 466
  - Tutorial: Edit a state machine 467
  - Tutorial: Run a state machine 467
- **Amazon EventBridge 468**
  - AWS Batch events 469
    - Job state change events 469
    - Job queue blocked events 471
    - Service job state change events 472
    - Service job queue blocked events 474
  - Tutorial: Use AWS user notifications with AWS Batch 475
  - AWS Batch jobs as EventBridge targets 476
    - Tutorial: Create a scheduled job 477
    - Tutorial: Create a rule with an event pattern 479
    - Tutorial: Pass input transformer 482
  - Tutorial: Listen for AWS Batch job events 485
    - Prerequisites 485
    - Tutorial: Create the Lambda function 485
    - Tutorial: Register the event rule 486
    - Tutorial: Test your configuration 488
  - Tutorial: Sending Amazon Simple Notification Service alerts for failed job events 488
    - Prerequisites 489

    - Tutorial: Create and subscribe to an Amazon SNS topic 489
    - Tutorial: Register an event rule 489
    - Tutorial: Test your rule 491
    - Alternate rule: Batch job queue blocked 492
- **Elastic Fabric Adapter 493**
- **Monitor AWS Batch 496**
  - CloudWatch Logs 496
    - Tutorial: Add a CloudWatch Logs IAM policy 497
    - Install and configure the CloudWatch agent 499
    - Tutorial: View CloudWatch Logs 499
  - CloudWatch Container Insights 501
    - Turn on Container Insights 502
  - Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs 503
    - Prerequisites 503
    - Install the add-on 503
- **Tag your resources 505**
  - Tag basics 505
  - Tag your resources 506
  - Tag restrictions 507
  - Tutorial: Manage tags using the console 508
    - Add tags on an individual resource on creation 508
    - Add and delete tags on an individual resource 508
  - Manage tags using the CLI or API 508
- **Best practices 511**
  - When to use AWS Batch 511
  - Checklist to run at scale 512
  - Optimize containers and AMIs 513
  - Choose the right compute environment resource 514
  - Amazon EC2 On-Demand or Amazon EC2 Spot 514
  - Use Amazon EC2 Spot best practices for AWS Batch 515
  - Common errors and troubleshooting 517
- **Troubleshooting 520**
  - AWS Batch 521
    - Optimal instance type configuration to receive automatic instance family updates 522
    - `INVALID` compute environment 522
    - Jobs stuck in a `RUNNABLE` status 525

- Spot Instances not tagged on creation 530
- Spot Instances not scaling down 531
- Can't retrieve Secrets Manager secrets 532
- Can't override job definition resource requirements 532
- Error message when you update the `desiredvCpus` setting 534
- AWS Batch on Amazon EKS 534
  - `INVALID` compute environment 535
  - AWS Batch on Amazon EKS job is stuck in `RUNNABLE` status 538
  - AWS Batch on Amazon EKS job is stuck in `STARTING` status 539
  - Verify that the `aws-auth ConfigMap` is configured correctly 540
  - RBAC permissions or bindings aren't configured properly 541

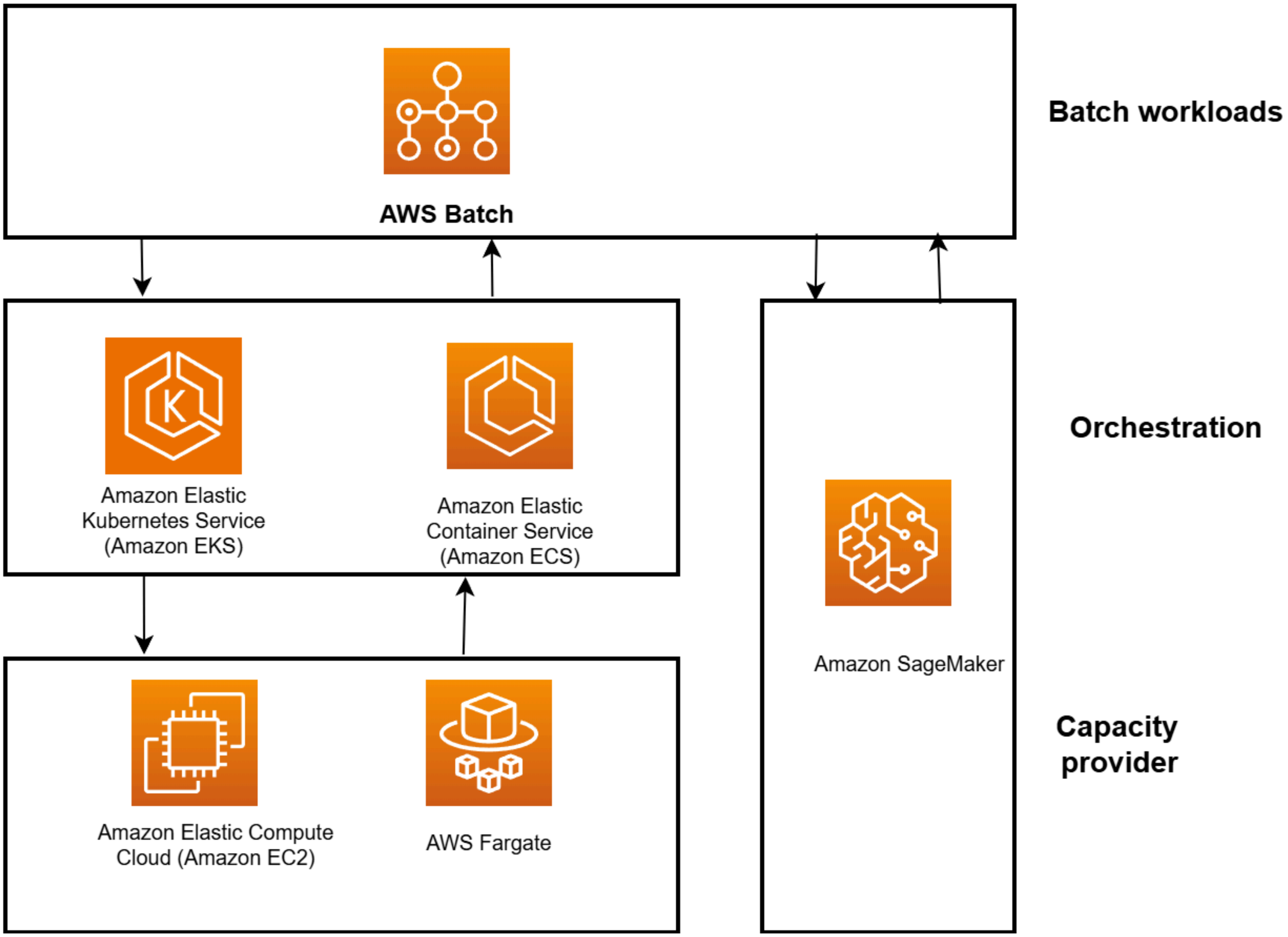
Resource: Service quotas 543

Document history 545

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.

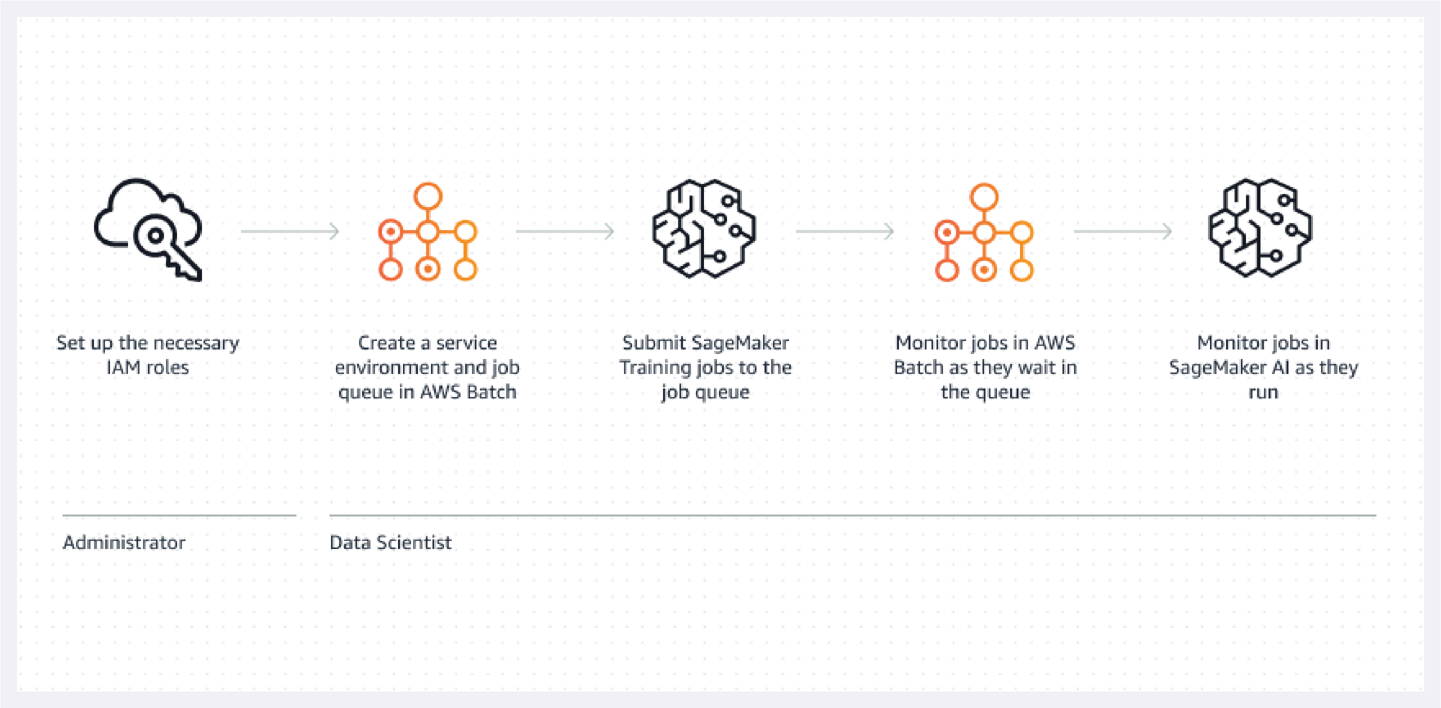

AWS Batch
Batch workloads
Amazon Elastic Kubernetes Service (Amazon EKS)
Amazon Elastic Container Service (Amazon ECS)
Orchestration
Amazon SageMaker
Capacity provider
Amazon Elastic Compute Cloud (Amazon EC2)
AWS Fargate

AWS Batch provides all of the necessary functionality to run high-scale, compute-intensive workloads on top of AWS managed container orchestration services, Amazon ECS and Amazon EKS. AWS Batch is able to scale compute capacity on Amazon EC2 instances and Fargate resources.

AWS Batch provides a fully managed service for batch workloads, and delivers the operational capabilities to optimize these types of workloads for throughput, speed, resource efficiency, and cost.

AWS Batch also enables SageMaker Training job queuing, allowing data scientists and ML engineers to submit Training jobs with priorities to configurable queues. This capability ensures that ML workloads run automatically as soon as resources become available, eliminating the need for manual coordination and improving resource utilization.

For machine learning workloads, AWS Batch provides queuing capabilities for SageMaker Training jobs. You can configure queues with specific policies to optimize cost, performance, and resource allocation for your ML Training workloads.


Set up the necessary IAM roles
Create a service environment and job queue in AWS Batch
Submit SageMaker Training jobs to the job queue
Monitor jobs in AWS Batch as they wait in the queue
Monitor jobs in SageMaker AI as they run
Administrator
Data Scientist

This provides a shared responsibility model where administrators set up the infrastructure and permissions, while data scientists can focus on submitting and monitoring their ML training workloads. Jobs are automatically queued and executed based on configured priorities and resource availability.

Are you a first-time AWS Batch user?

If you are a first-time user of AWS Batch, we recommend that you begin by reading the following sections:

- Components of AWS Batch
- Create IAM account and administrative user
- Setting up AWS Batch
- Getting started with AWS Batch tutorials
- Getting started with AWS Batch on SageMaker AI

Related services

AWS Batch is a fully managed batch computing service that plans, schedules, and runs your containerized batch ML, simulation, and analytics workloads across the full range of AWS compute offerings, such as Amazon ECS, Amazon EKS, AWS Fargate, and Spot or On-Demand Instances. For more information about each managed compute service, see:

- Amazon EC2 *User Guide*
- AWS Fargate *Developer Guide*
- Amazon EKS *User Guide*
- Amazon SageMaker AI *Developer Guide*

Accessing AWS Batch

You can access AWS Batch using the following:

AWS Batch console

The web interface where you create and manage resources.

AWS Command Line Interface

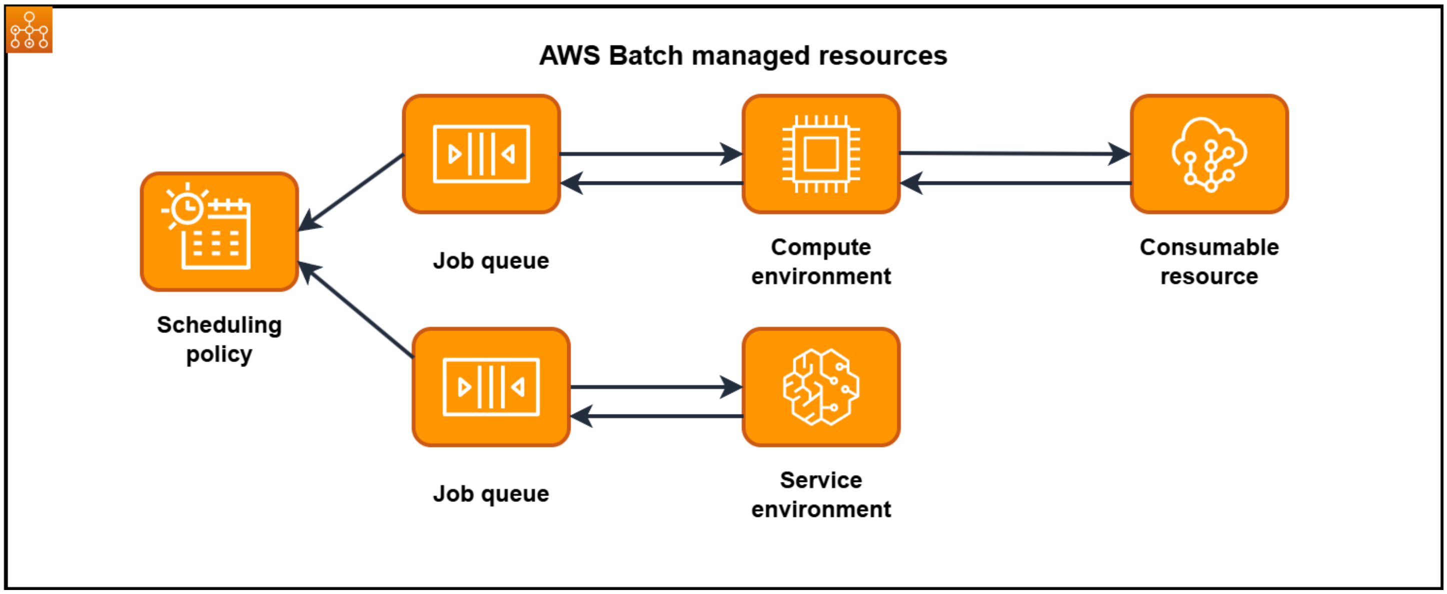
Interact with AWS services using commands in your command line shell. The AWS Command Line Interface is supported on Windows, macOS, and Linux. For more information about the AWS CLI, see AWS Command Line Interface User Guide. You can find the AWS Batch commands in the AWS CLI Command Reference.

AWS SDKs

> If you prefer to build applications using language-specific APIs instead of submitting a request over HTTP or HTTPS, use the libraries, sample code, tutorials, and other resources provided by AWS. These libraries provide basic functions that automate tasks, such as cryptographically signing your requests, retrying requests, and handling error responses. These functions make it more efficient for you to get started. For more information, see Tools to Build on AWS.

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.



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 environments for AWS Batch.

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. For more information, see Job queues.

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

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 or the availability of resources you specify. For more information, see Jobs.

Scheduling policy

You can use scheduling policies to configure how compute resources in a job queue are allocated between users or workloads. Using fair-share scheduling policies, you can assign different share identifiers to workloads or users. The AWS Batch job scheduler defaults to a first-in, first-out (FIFO) strategy. For more information, see Fair-share scheduling policies.

Consumable resources

A consumable resource is a resource that is needed to run your jobs, such as a 3rd party license token, database access bandwidth, the need to throttle calls to a third-party API, and so on. You specify the consumable resources which are needed for a job to run, and Batch takes these resource dependencies into account when it schedules a job. You can reduce the under-utilization of compute resources by allocating only the jobs that have all the required resources available. For more information, see Resource-aware scheduling .

Service Environment

A Service Environment define how AWS Batch integrates with SageMaker for job execution. Service Environments enable AWS Batch to submit and manage jobs on SageMaker while providing the queuing, scheduling, and priority management capabilities of AWS Batch. Service Environments define capacity limits for specific service types such as SageMaker Training jobs. The capacity limits control the maximum resources that can be used by service jobs in the environment. For more information, see Service environments for AWS Batch.

Service job

A service job is a unit of work that you submit to AWS Batch to run on a service environment. Service jobs leverage AWS Batch's queuing and scheduling capabilities while delegating actual execution to the external service. For example, SageMaker Training jobs submitted as service jobs are queued and prioritized by AWS Batch, but the SageMaker Training job execution occurs within SageMaker AI infrastructure. This integration enables data scientists and ML engineers to benefit from AWS Batch's automated workload management, and priority queuing, for their SageMaker AI Training workloads. Service jobs can reference other jobs by name or ID and support job dependencies. For more information, see Service jobs in AWS Batch.

Setting up 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.

Topics

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

Create IAM account and administrative user

To get started, you need to create an AWS account and a single user that is typically granted administrative rights. To accomplish this, complete the following tutorials:

Sign up for an AWS account

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

To sign up for an AWS account

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

   Part of the sign-up procedure involves receiving a phone call or text message 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 a 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 a user with administrative access

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 a user with administrative access

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 a 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 user with administrative access

- 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*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center 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 AWS Identity and Access Management role that provides these credentials to your compute environments and container instances, then associate that role with your compute environments.

> **Note**
>
> To verify that your AWS account has the required permissions, see Initial IAM service set up for your account.
> 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, Amazon ECS instance role, and Tutorial: Create the IAM execution role before creating your first compute environment.

Create a key pair for your instances

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*.

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
   ```

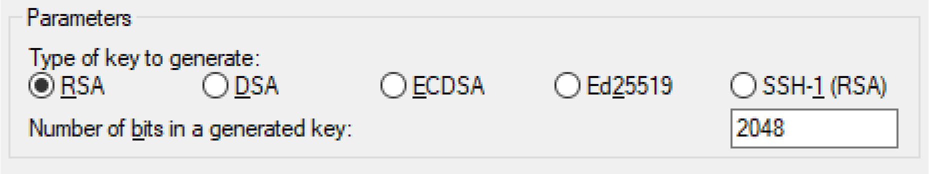
For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.

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

1. Download and install PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/. Be sure to install the entire suite.
2. Start PuTTYgen (for example, from the **Start** menu, choose **All Programs, PuTTY, and PuTTYgen**).
3. Under **Type of key to generate**, choose **RSA**. If you're using an earlier version of PuTTYgen, choose **SSH-2 RSA**.



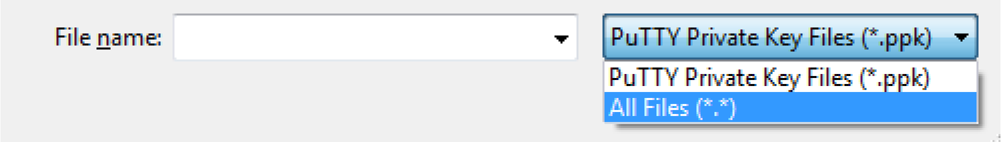
Parameters

Type of key to generate:

● RSA ○ DSA ○ ECDSA ○ Ed25519 ○ SSH-1 (RSA)

Number of bits in a generated key: 2048

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


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

5. Select the private key file that you created in the previous procedure and choose **Open**. Choose **OK** to dismiss the confirmation dialog box.
6. Choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the `.ppk` file extension.

Create a 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 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.

Option	Value
Resources to create	VPC only
Name	Optionally provide a name for your VPC.
IPv4 CIDR block	IPv4 CIDR manual input The CIDR block size must have a size between /16 and /28.

Option	Value
IPv6 CIDR block	No IPv6 CIDR block
Tenancy	Default

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

Create a security group

Security groups act as a firewall for associated 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*.

> **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.

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

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 tutorials

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.

Afterward, you can use the AWS Batch first-run wizard to create a compute environment, job queue, and submit a sample *Hello World* job.

Getting started with Amazon EC2 orchestration using the Wizard

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.

Overview

This tutorial demonstrates how to setup AWS Batch with the Wizard to configure Amazon EC2 and run `Hello World`.

Intended Audience

This tutorial is designed for system administrators and developers responsible for setting up, testing, and deploying AWS Batch.

Features Used

This tutorial shows you how to use the AWS Batch console wizard to:

- Create and configure an Amazon EC2 compute environment
- Create a job queue.

- Create a job definition
- Create and submit a job to run
- View the output of the job in CloudWatch

Time Required

It should take about 10–15 minutes to complete this tutorial.

Regional Restrictions

There are no country or regional restrictions associated with using this solution.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
Amazon EC2 instance	You pay for each Amazon EC2 instance that is created. For more information about pricing, see Amazon EC2 Pricing.

Prerequisites

Before you begin:

- Create an AWS account if you don't have one.
- Create the `ecsInstanceRole` Instance role.

Step 1: Create a compute environment

> ⚠ **Important**
>
> To get started as simply and quickly as possible, this tutorial includes steps with default settings. Before creating for production use, we recommend that you familiarize yourself with all settings and deploy with the settings that meet your requirements.

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

1. Open the AWS Batch console first-run wizard.
2. For **Configure job and 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 role that has the required IAM permissions attached. This instance role 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.

   The default name of the **Instance role** is `ecsInstanceRole`.
6. For **Instance configuration** you can leave the default settings.
7. For **Network configuration** use your default VPC for the AWS Region.
8. Choose **Next**.

Step 2: 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. For **Job queue configuration** for **Name**, specify a unique name for your job queue. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
2. For all other configuration options you can leave the default value.
3. Choose **Next**.

Step 3: 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. For **Create a job definition**
    a. for **Name**, specify a unique name for your job queue. The name can be up to 128 characters in length. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
    b. For **Command - *optional*** you can change `hello world` to a custom message or leave it as is.
2. For all other configuration options you can leave the default value.
3. Choose **Next**.

Step 4: 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. For all other configuration options you can leave the default value.
3. Choose **Next**.

Step 5: 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**.

1. For **Review and create** choose **Create resources**.
2. A window opens as AWS Batch starts to allocate your resources. Once complete choose **Go to dashboard**. On the dashboard you should see all of your allocated resources and that the job is in the `Runnable` state. Your job is scheduled to run and should complete in 2–3 minuets.

Step 6: View the Job's output

To view the Job's output, do the following:

1. In the navigation pane choose **Jobs**.
2. In the **Job queue** drop down choose the Job queue you created for the tutorial.
3. The **Jobs** table lists all of your Jobs and what their current status is. Once the Job's **Status** is **Succeeded** choose the **Name** of the Job to view the Job's details.
4. In the **Details** pane choose **Log stream name**. The CloudWatch console for the Job will open and there should be one event with the **Message** of `hello world` or your custom message.

Step 7: Clean up your tutorial resources

You are charged for the Amazon EC2 instance while it is enabled. You can delete the instance to stop incurring charges.

To delete the resources you created, do the following:

1. In the navigation pane choose **Job queue**.
2. In the **Job queue** table choose the Job queue you created for the tutorial.
3. Choose **Disable**. Once the Job queue **State** is Disabled you can choose **Delete**.
4. Once the Job queue is deleted, in the navigation pane choose **Compute environments**.
5. Choose the compute environment you created for this tutorial and then choose **Disable**. It may take 1–2 minuets for the compute environment to complete being disabled.
6. Once the compute environment’s **State** is Disabled, choose **Delete**. It may take 1–2 minuets for the compute environment to be deleted.

Additional resources

After you complete the tutorial, you might want to explore the following topics::

- Explore the AWS Batch core components. For more information, see Components of AWS Batch.
- Learn more about the different Compute Environments available in AWS Batch.
- Learn more about Job queues and their different scheduling options.
- Learn more about Job definitions and the different configuration options.
- Learn more about the different types of Jobs.

Getting started with AWS Batch and Fargate orchestration using the Wizard

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.

Overview

This tutorial demonstrates how to setup AWS Batch with the Wizard to configure AWS Fargate and run `Hello World`.

Intended Audience

This tutorial is designed for system administrators and developers responsible for setting up, testing, and deploying AWS Batch.

Features Used

This tutorial shows you how to use the AWS Batch console wizard to:

- Create and configure an AWS Fargate compute environment
- Create a job queue.
- Create a job definition
- Create and submit a job to run
- View the output of the job in CloudWatch

Time Required

It should take about 10–15 minutes to complete this tutorial.

Regional Restrictions

There are no country or regional restrictions associated with using this solution.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
Pricing is based on requested vCPU, memory, Operating Systems, CPU Architecture, and storage resources for the Task or Pod.	For more information about pricing, see Fargate Pricing.

Prerequisites

Before you begin:

- Create an AWS account if you don't have one.
- Create the task execution role. If you haven't already created the Task Execution Role then you can create it as part of this tutorial.

Step 1: Create a compute environment

> ⚠ **Important**
>
> To get started as simply and quickly as possible, this tutorial includes steps with default settings. Before creating for production use, we recommend that you familiarize yourself with all settings and deploy with the settings that meet your requirements.

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

1. Open the AWS Batch console first-run wizard.
2. For **Configure job and 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. For all other configuration options you can leave the default value.
6. Choose **Next**.

Step 2: 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 900 for the job queue.
3. For all other configuration options you can leave the default value.
4. Choose **Next**.

Step 3: Create a job definition

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. It can contain uppercase and lowercase letters, numbers, hyphens (-), and underscores (_).
2. In the **Fargate platform configuration** section:
    a. Turn on **Assign public IP** to assign a public IP address. You need a public IP to download the container image unless you've setup a private image repository.
    b. For **Execution role**, choose a task execution role that lets Amazon Elastic Container Service (Amazon ECS) agents make AWS calls on your behalf. Choose either **ecsTaskExecutionRole** or **BatchEcsTaskExecutionRole**.

       To create the **Execution role** choose **Create an execution role**. In the **Create IAM role** modal choose **Create IAM role**.

        i. The IAM console has the permission setting already configured for creating the execution role.
        ii. For **Trusted entity type** verify that **AWS service** is selected.

    iii. For **Service or user case** verify that **Elastic Container Service** is selected.
    iv. Choose **Next**.
    v. For **Permissions policies** verify that **AmazonECSTaskExecutionRolePolicy** is selected.
    vi. Choose **Next**.
    vii. For **Name, review, and create** verify that the role name is **BatchEcsTaskExecutionRole**.
    viii. Choose **Create role**.
    ix. In the AWS Batch console choose the refresh button next to **Execution role**. Choose the **BatchEcsTaskExecutionRole** execution role.
3. In the **Container configuration** section:

    - For **Command**, you can change `hello world` to a custom message or leave it as is.
4. For all other configuration options you can leave the default value.
5. Choose **Next**.

Step 4: 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. For all other configuration options you can leave the default value.
3. Choose **Next**.

Step 5: 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**.

Step 6: View the Job's output

To view the Job's output, do the following:

1. In the navigation pane choose **Jobs**.

2. In the **Job queue** drop down choose the Job queue you created for the tutorial.
3. The **Jobs** table lists all of your Jobs and what their current status is. Once the Job's **Status** is **Succeeded** choose the **Name** of the Job to view the Job's details.
4. In the **Details** pane choose **Log stream name**. The CloudWatch console for the Job will open and there should be one event with the **Message** of `hello world` or your custom message.

Step 7: Clean up your tutorial resources

You are charged for the Amazon EC2 instance while it is enabled. You can delete the instance to stop incurring charges.

To delete the resources you created, do the following:

1. In the navigation pane choose **Job queue**.
2. In the **Job queue** table choose the Job queue you created for the tutorial.
3. Choose **Disable**. Once the Job queue **State** is Disabled you can choose **Delete**.
4. Once the Job queue is deleted, in the navigation pane choose **Compute environments**.
5. Choose the compute environment you created for this tutorial and then choose **Disable**. It may take 1–2 minuets for the compute environment to complete being disabled.
6. Once the compute environment’s **State** is Disabled, choose **Delete**. It may take 1–2 minuets for the compute environment to be deleted.

Additional resources

After you complete the tutorial, you might want to explore the following topics::

- Learn more about the Best practices.
- Explore the AWS Batch core components. For more information, see Components of AWS Batch.
- Learn more about the different Compute Environments available in AWS Batch.
- Learn more about Job queues and their different scheduling options.
- Learn more about Job definitions and the different configuration options.
- Learn more about the different types of Jobs.

Getting started with AWS Batch and Fargate using the AWS CLI

This tutorial demonstrates how to set up AWS Batch with AWS Fargate orchestration and run a simple "Hello World" job using the AWS Command Line Interface (AWS CLI). You'll learn how to create compute environments, job queues, job definitions, and submit jobs to AWS Batch.

Topics

- Prerequisites
- Create an IAM execution role
- Create a compute environment
- Create a job queue
- Create a job definition
- Submit and monitor a job
- View job output
- Clean up resources
- Going to production
- Next steps

Prerequisites

Before you begin this tutorial, make sure you have the following.

1. The AWS CLI. If you need to install it, follow the AWS CLI installation guide. You can also use AWS CloudShell, which includes the AWS CLI.
2. Configured your AWS CLI with appropriate credentials. Run `aws configure` if you haven't set up your credentials yet.
3. Basic familiarity with command line interfaces and containerization concepts.
4. How AWS Batch works with IAM to create and manage AWS Batch resources, IAM roles, and VPC resources in your AWS account.
5. A subnet ID and security group ID from a VPC in your AWS account. If you don't have a VPC, you can create one. For more information about using the AWS CLI to retrieve these resource IDs, see describe-subnets and describe-security-groups in the *AWS CLI Command Reference*.

Time Required: Approximately 15-20 minutes to complete this tutorial.

Cost: This tutorial uses Fargate compute resources. The estimated cost for completing this tutorial is less than $0.01 USD, assuming you follow the cleanup instructions to delete resources immediately after completion. Fargate pricing is based on vCPU and memory resources consumed, charged per second with a 1-minute minimum. For current pricing information, see AWS Fargate pricing.

Create an IAM execution role

AWS Batch requires an execution role that allows Amazon Elastic Container Service (Amazon ECS) agents to make AWS API calls on your behalf. This role is necessary for Fargate tasks to pull container images and write logs to Amazon CloudWatch.

Create a trust policy document

First, create a trust policy that allows the Amazon ECS tasks service to assume the role.

```
cat > batch-execution-role-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

Create the execution role

The following command creates an IAM role named `BatchEcsTaskExecutionRoleTutorial` using the trust policy you just created.

```
aws iam create-role \
    --role-name BatchEcsTaskExecutionRoleTutorial \
    --assume-role-policy-document file://batch-execution-role-trust-policy.json
```

Attach the required policy

Attach the AWS managed policy that provides the necessary permissions for Amazon ECS task execution.

```
aws iam attach-role-policy \
    --role-name BatchEcsTaskExecutionRoleTutorial \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

The role is now ready to be used by AWS Batch for Fargate task execution.

Create a compute environment

A compute environment defines the compute resources where your batch jobs will run. For this tutorial, you'll create a managed Fargate compute environment that automatically provisions and scales resources based on job requirements.

Create the compute environment

The following command creates a Fargate compute environment. Replace the example subnet and security group IDs with your own per the Prerequisites.

```
aws batch create-compute-environment \
    --compute-environment-name my-fargate-compute-env \
    --type MANAGED \
    --state ENABLED \
    --compute-resources type=FARGATE,maxvCpus=128,subnets=subnet-
a123456b,securityGroupIds=sg-a12b3456
```

The following shows how the output looks when the command runs successfully.

```
{
    "computeEnvironmentName": "my-fargate-compute-env",
    "computeEnvironmentArn": "arn:aws:batch:us-west-2:123456789012:compute-environment/
my-fargate-compute-env"
}
```

Wait for the compute environment to be ready

Check the status of your compute environment to ensure it's ready before proceeding.

```
aws batch describe-compute-environments \
    --compute-environments my-fargate-compute-env \
    --query 'computeEnvironments[0].status'
```

```
"VALID"
```

When the status shows `VALID`, your compute environment is ready to accept jobs.

Create a job queue

A job queue stores submitted jobs until the AWS Batch scheduler runs them on resources in your compute environment. Jobs are processed in priority order within the queue.

Create the job queue

The following command creates a job queue with priority 900 that uses your Fargate compute environment.

```
aws batch create-job-queue \
    --job-queue-name my-fargate-job-queue \
    --state ENABLED \
    --priority 900 \
    --compute-environment-order order=1,computeEnvironment=my-fargate-compute-env
```

The following shows how the output looks when the command runs successfully.

```
{
    "jobQueueName": "my-fargate-job-queue",
    "jobQueueArn": "arn:aws:batch:us-west-2:123456789012:job-queue/my-fargate-job-
queue"
}
```

Verify the job queue is ready

Check that your job queue is in the `ENABLED` state and ready to accept jobs.

```
aws batch describe-job-queues \
    --job-queues my-fargate-job-queue \
    --query 'jobQueues[0].state' "ENABLED"
```

Create a job definition

A job definition specifies how jobs are to be run, including the Docker image to use, resource requirements, and other parameters. For Fargate, you'll use resource requirements instead of traditional vCPU and memory parameters.

Create the job definition

The following command creates a job definition that runs a simple "hello world" command using the busybox container image. Replace `123456789012` with your actual AWS account ID and replace the example AWS Region with your own.

```
aws batch register-job-definition \
    --job-definition-name my-fargate-job-def \
    --type container \
    --platform-capabilities FARGATE \
    --container-properties '{
        "image": "busybox",
        "resourceRequirements": [
            {"type": "VCPU", "value": "0.25"},
            {"type": "MEMORY", "value": "512"}
        ],
        "command": ["echo", "hello world"],
        "networkConfiguration": {
            "assignPublicIp": "ENABLED"
        },
        "executionRoleArn": "arn:aws:iam::123456789012:role/
BatchEcsTaskExecutionRoleTutorial"
    },
{
    "jobDefinitionName": "my-fargate-job-def",
    "jobDefinitionArn": "arn:aws:batch:us-west-2:123456789012:job-definition/my-
fargate-job-def:1",
    "revision": 1
}'
```

The job definition specifies 0.25 vCPU and 512 MB of memory, which are the minimum resources for a Fargate task. The `assignPublicIp` setting is enabled so the container can pull the busybox image from Docker Hub.

Submit and monitor a job

Now that you have all the necessary components, you can submit a job to your queue and monitor its progress.

Submit a job

The following command submits a job to your queue using the job definition you created.

```
aws batch submit-job \
    --job-name my-hello-world-job \
    --job-queue my-fargate-job-queue \
    --job-definition my-fargate-job-def
```

The following shows how the output looks when the command runs successfully.

```
{
    "jobArn": "arn:aws:batch:us-west-2:123456789012:job/my-hello-world-job",
    "jobName": "my-hello-world-job",
    "jobId": "1509xmpl-4224-4da6-9ba9-1d1acc96431a"
}
```

Make note of the `jobId` returned in the response, as you'll use it to monitor the job's progress.

Monitor job status

Use the job ID to check the status of your job. The job will progress through several states: `SUBMITTED`, `PENDING`, `RUNNABLE`, `STARTING`, `RUNNING`, and finally `SUCCEEDED` or `FAILED`.

```
aws batch describe-jobs --jobs 1509xmpl-4224-4da6-9ba9-1d1acc96431a
```

The following shows how the output looks when the command runs successfully.

```
{
    "jobs": [
        {
            "jobArn": "arn:aws:batch:us-west-2:123456789012:job/my-hello-world-job",
            "jobName": "my-hello-world-job",
            "jobId": "1509xmpl-4224-4da6-9ba9-1d1acc96431a",
            "jobQueue": "arn:aws:batch:us-west-2:123456789012:job-queue/my-fargate-job-
queue",
            "status": "SUCCEEDED",
            "createdAt": 1705161908000,
            "jobDefinition": "arn:aws:batch:us-west-2:123456789012:job-definition/my-
fargate-job-def:1"
        }
    ]
}
```

When the status shows `SUCCEEDED`, your job has completed successfully.

View job output

After your job completes, you can view its output in Amazon CloudWatch Logs.

Get the log stream name

First, retrieve the log stream name from the job details. Replace the example job ID with your own.

```
aws batch describe-jobs --jobs 1509xmpl-4224-4da6-9ba9-1d1acc96431a \
    --query 'jobs[0].attempts[0].containers[0].logStreamName' \
    --output text
```

```
my-fargate-job-def/default/1509xmpl-4224-4da6-9ba9-1d1acc96431a
```

View the job logs

Use the log stream name to retrieve the job's output from CloudWatch Logs.

```
aws logs get-log-events \
    --log-group-name /aws/batch/job \
    --log-stream-name my-fargate-job-def/default/1509xmpl-4224-4da6-9ba9-1d1acc96431a \
    --query 'events[*].message' \
    --output text
```

The output shows "hello world", confirming that your job ran successfully.

Clean up resources

To avoid ongoing charges, clean up the resources you created in this tutorial. You must delete resources in the correct order due to dependencies.

Disable and delete the job queue

First, disable the job queue, then delete it.

```
aws batch update-job-queue \
    --job-queue my-fargate-job-queue \
    --state DISABLED
```

```
aws batch delete-job-queue \
    --job-queue my-fargate-job-queue
```

Disable and delete the compute environment

After the job queue is deleted, disable and delete the compute environment.

```
aws batch update-compute-environment \
    --compute-environment my-fargate-compute-env \
    --state DISABLED
```

```
aws batch delete-compute-environment \
    --compute-environment my-fargate-compute-env
```

Clean up the IAM role

Remove the policy attachment and delete the IAM role.

```
aws iam detach-role-policy \
    --role-name BatchEcsTaskExecutionRoleTutorial \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

```
aws iam delete-role \
    --role-name BatchEcsTaskExecutionRoleTutorial
```

Remove temporary files

Delete the trust policy file you created.

```
rm batch-execution-role-trust-policy.json
```

All resources have been successfully cleaned up.

Going to production

This tutorial is designed to help you understand how AWS Batch works with Fargate. For production deployments, consider the following additional requirements:

Security considerations:

- Create dedicated security groups with minimal required access instead of using default security groups
- Use private subnets with NAT Gateway instead of public IP assignment for containers
- Store container images in Amazon ECR instead of using public repositories
- Implement VPC endpoints for AWS service communication to avoid internet traffic

Architecture considerations:

- Deploy across multiple Availability Zones for high availability
- Implement job retry strategies and dead letter queues for error handling
- Use multiple job queues with different priorities for workload management
- Configure auto scaling policies based on queue depth and resource utilization
- Implement monitoring and alerting for job failures and resource utilization

Operational considerations:

- Set up CloudWatch dashboards and alarms for monitoring
- Implement proper logging and audit trails
- Use CloudFormation or the AWS CDK for infrastructure as code
- Establish backup and disaster recovery procedures

For comprehensive guidance on production-ready architectures, see the AWS Well-Architected Framework and AWS Security Best Practices.

Next steps

Now that you've completed this tutorial, you can explore more advanced AWS Batch features:

- Job queues – Learn about job queue scheduling and priority management

- Job definitions – Explore advanced job definition configurations including environment variables, volumes, and retry strategies
- Compute environments for AWS Batch – Understand different compute environment types and scaling options
- Multi-node parallel jobs – Run jobs that span multiple compute nodes
- Array jobs – Submit large numbers of similar jobs efficiently
- Best practices for AWS Batch – Learn optimization techniques for production workloads

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

- Overview

- Prerequisites
- Step 1: Create your Amazon EKS cluster for AWS Batch
- Step 2: Prepare your Amazon EKS cluster for AWS Batch
- Step 3: Create an Amazon EKS compute environment
- Step 4: Create a job queue and attach the compute environment
- Step 5: Create a job definition
- Step 6: Submit a job
- Step 7: View the Job's output
- Step 8: (Optional) Submit a job with overrides
- Step 9: Clean up your tutorial resources
- Additional resources

Overview

This tutorial demonstrates how to setup AWS Batch with Amazon EKS using the AWS CLI, `kubectl` and `eksctl`.

Intended Audience

This tutorial is designed for system administrators and developers responsible for setting up, testing, and deploying AWS Batch.

Features Used

This tutorial shows you how to use the AWS CLI, to:

- Create and configure an Amazon EKS compute environment
- Create a job queue.
- Create a job definition
- Create and submit a job to run
- Submit a job with overrides

Time Required

It should take about 30–40 minutes to complete this tutorial.

Regional Restrictions

There are no country or regional restrictions associated with using this solution.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
You are charged by the cluster hour	Varies depending on Instance, see Amazon EKS pricing

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, 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.
- **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.

- **AWS account number** – You need to know your AWS account ID. Follow the directions in [Viewing your AWS account ID]().
- **(Optional) CloudWatch** – To examine the details of [(Optional) Submit a job with overrides](), logging must be configured. For more information, see [Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs]().

Step 1: Create your Amazon EKS cluster for AWS Batch

> ⚠ **Important**
>
> To get started as simply and quickly as possible, this tutorial includes steps with default settings. Before creating for production use, we recommend that you familiarize yourself with all settings and deploy with the settings that meet your requirements.

Once you have installed the prerequisites you need to create your cluster using `eksctl`. Creating the cluster can take between 10-15 minutes.

```
$  eksctl create cluster --name my-cluster-name --region region-code
```

In the preceding command replace:

- Replace *`my-cluster-name`* with the name you want to use for your cluster.
- Replace *`region-code`* with the AWS Region to create the cluster in, for example `us-west-2`.

The cluster name and region are needed for later in this tutorial.

Step 2: Prepare 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.

   ```
   $ 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.

```
$ 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: ["events"]
    verbs: ["list"]
  - 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:
  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
```

3. 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
```

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

   In the following command replace:

   - Replace *`<your-account-number>`* with your AWS account number.

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

Output:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account-number>: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 3: Create 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.

- For `subnets` run `eksctl get cluster my-cluster-name` to get the subnets used by the cluster.
- For `securityGroupIds` parameter you can use the same security group as the Amazon EKS cluster. This command retrieves the security group ID for the cluster.

```
$ aws eks describe-cluster \
    --name my-cluster-name \
    --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

- The `instanceRole` is created when you create the cluster. To find the `instanceRole` list all entities that use the `AmazonEKSWorkerNodePolicy` policy:

```
$  aws iam list-entities-for-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy
```

The name of the policy role contains the name of the cluster that you created `eksctl-my-cluster-name-nodegroup-example`.

To find the `instanceRole` arn run the following command:

```
$  aws iam list-instance-profiles-for-role --role-name eksctl-my-cluster-name-
nodegroup-example
```

Output:

```
INSTANCEPROFILES        arn:aws:iam::<your-account-number>:instance-profile/
eks-04cb2200-94b9-c297-8dbe-87f12example
```

For more information, 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.

```
$ cat <<EOF > ./batch-eks-compute-environment.json
{
  "computeEnvironmentName": "My-Eks-CE1",
  "type": "MANAGED",
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:region-code:your-account-number:cluster/my-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

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

Step 4: Create a job queue and attach the compute environment

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
```

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.

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 5: Create a job definition

The following Job definition instructs the pod to sleep for 60 seconds.

```
$ 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": {
          "environment": "test"
        }
      }
      }
    }
}
EOF
```

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

Notes

- There are considerations for the `cpu` and `memory` parameters. For more information, see Memory and vCPU considerations for AWS Batch on Amazon EKS.

Step 6: Submit a job

Run the following AWS CLI command to submit a new Job.

```
$ aws batch submit-job --job-queue My-Eks-JQ1 \
    --job-definition MyJobOnEks_Sleep --job-name My-Eks-Job1
```

To check the status of a Job:

```
$ aws batch describe-jobs --job <jobId-from-submit-response>
```

Notes

- For more information about running jobs on Amazon EKS resources, see Amazon EKS jobs.

Step 7: View the Job's output

To view the Job's output, do the following:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane choose **Jobs**.
3. In the **Job queue** drop down choose the Job queue you created for the tutorial.
4. The **Jobs** table lists all of your Jobs and what their current status is. Once the Job's **Status** is **Succeeded** choose the **Name** of the Job, *My-Eks-JQ1*, to view the Job's details.
5. In the **Details** pane the **Started at** and **Stopped at** times should be one minute apart.

Step 8: (Optional) Submit a job with overrides

This job overrides the command passed to the container. 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.

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

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

Notes

- 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.

To view the Job's output, do the following:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane choose **Jobs**.
3. In the **Job queue** drop down choose the Job queue you created for the tutorial.
4. The **Jobs** table lists all of your Jobs and what their current status is. Once the Job's **Status** is **Succeeded** choose the **Name** of the Job to view the Job's details.
5. In the **Details** pane choose **Log stream name**. The CloudWatch console for the Job will open and there should be one event with the **Message** of `hello world` or your custom message.

Step 9: Clean up your tutorial resources

You are charged for the Amazon EC2 instance while it is enabled. You can delete the instance to stop incurring charges.

To delete the resources you created, do the following:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane choose **Job queue**.
3. In the **Job queue** table choose the Job queue you created for the tutorial.
4. Choose **Disable**. Once the Job queue **State** is Disabled you can choose **Delete**.
5. Once the Job queue is deleted, in the navigation pane choose **Compute environments**.
6. Choose the compute environment you created for this tutorial and then choose **Disable**. It may take 1–2 minuets for the compute environment to complete being disabled.
7. Once the compute environment’s **State** is Disabled, choose **Delete**. It may take 1–2 minuets for the compute environment to be deleted.

Additional resources

After you complete the tutorial, you might want to explore the following topics::

- Learn more about the Best practices.
- Explore the AWS Batch core components. For more information, see Components of AWS Batch.
- Learn more about the different Compute Environments available in AWS Batch.
- Learn more about Job queues and their different scheduling options.

- Learn more about Job definitions and the different configuration options.
- Learn more about the different types of Jobs.

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.

Amazon EKS private only clusters by default have no inbound/outbound internet access, and you can only access the API server from within your VPC or a connected network. Amazon VPC endpoints are used to enable private access to other AWS services. `eksctl` supports creating fully private clusters using a pre-existing Amazon VPC and subnets. `eksctl` 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 `eksctl` 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

- Overview
- Prerequisites

- Step 1: Create your EKS cluster for AWS Batch
- Step 2: Prepare your EKS cluster for AWS Batch
- Step 3: Create an Amazon EKS compute environment
- Step 4: Create a job queue and attach the compute environment
- Step 5: Create an Amazon ECR with pull through cache
- Step 6: Register a job definition
- Step 7: Submit a job to run
- Step 8: View the Job's output
- Step 9: (Optional) Submit a job with overrides
- Step 10: Clean up your tutorial resources
- Additional resources
- Troubleshooting

Overview

This tutorial demonstrates how to setup AWS Batch with a private Amazon EKS using the AWS CloudShell, `kubectl` and `eksctl`.

Intended Audience

This tutorial is designed for system administrators and developers responsible for setting up, testing, and deploying AWS Batch.

Features Used

This tutorial shows you how to use the AWS CLI, to:

- Use Amazon Elastic Container Registry (Amazon ECR) to store container images
- Create and configure an Amazon EKS compute environment
- Create a job queue.
- Create a job definition
- Create and submit a job to run
- Submit a job with overrides

Time Required

It should take about 40–50 minutes to complete this tutorial.

Regional Restrictions

There are no country or regional restrictions associated with using this solution.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
You are charged by the cluster hour	Varies depending on Instance, see Amazon EKS pricing
Amazon EC2 instance	You pay for each Amazon EC2 instance that is created. For more information about pricing, see Amazon EC2 Pricing.

Prerequisites

This tutorial uses AWS CloudShell which is a browser-based, pre-authenticated shell that you launch directly from the AWS Management Console. This allows for access to the cluster once it no longer has public internet access. The AWS CLI, `kubectl`, and `eksctl` may already be installed as part of AWS CloudShell. For more information on AWS CloudShell, see the AWS CloudShell*User Guide*. An alternative to AWS CloudShell is to connect to your cluster's VPC or a connected network.

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.

- **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.
- **Permissions** – Users calling the CreateComputeEnvironment API operation to create a compute environment that uses Amazon EKS resources require permissions to the `eks:DescribeCluster` and `eks:ListClusters` API operation. You can attach the AWSBatchFullAccess managed policy to your user account by following the directions Adding and removing IAM identity permissions in the *IAM User Guide*.
- **InstanceRole** – You need to create an `InstanceRole` for your Amazon EKS nodes that has the `AmazonEKSWorkerNodePolicy` and `AmazonEC2ContainerRegistryPullOnly` polices. For directions on how to create the `InstanceRole` see Creating the Amazon EKS node IAM role. You will need the ARN of the `InstanceRole`.
- **AWS account ID** – You need to know your AWS account ID. Follow the directions in Viewing your AWS account ID.
- **(Optional) CloudWatch** – To examine the details of (Optional) Submit a job with overrides, logging must be configured. For more information, see Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs.

Step 1: Create your EKS cluster for AWS Batch

> **Important**
>
> To get started as simply and quickly as possible, this tutorial includes steps with default settings. Before creating for production use, we recommend that you familiarize yourself with all settings and deploy with the settings that meet your requirements.

We recommend you use `eksctl` and the following config file to create your cluster. To manually setup your cluster follow the directions in Deploy private clusters with limited internet access in the *Amazon EKS User Guide*.

1. Open the AWS CloudShell console and set the region to `us-east-1`. For the rest of the tutorial make sure you are using `us-east-1`.
2. Create a private EKS cluster in the `us-east-1` region using the sample `eksctl` config file. Save the yaml file to your AWS CloudShell environment and name it `clusterConfig.yaml` .You can change *`my-test-cluster`* with the name you want to use for your cluster.

```
kind: ClusterConfig
apiVersion: eksctl.io/v1alpha5
metadata:
    name: my-test-cluster
    region: us-east-1
availabilityZones:
    - us-east-1a
    - us-east-1b
    - us-east-1c
managedNodeGroups:
    - name: ng-1
      privateNetworking: true
privateCluster:
    enabled: true
    skipEndpointCreation: false
```

3. Create your resources using the command: `eksctl create cluster -f clusterConfig.yaml`. Cluster creation can take between 10–15 minutes.
4. Once the cluster has finished being created you have to add your AWS CloudShell IP address to the allow list. To find your AWS CloudShell IP address run the following command:

   ```
   curl http://checkip.amazonaws.com
   ```

   Once you have the public IP address you have to create an allow list rule:

   ```
   aws eks update-cluster-config \
     --name my-test-cluster \
     --region us-east-1 \
     --resources-vpc-config
    endpointPublicAccess=true,endpointPrivateAccess=true,publicAccessCidrs=["<Public
    IP>/32"]
   ```

   Then apply the update to the kubectl config file:

   ```
   aws eks update-kubeconfig --name my-test-cluster --region us-east-1
   ```

5. To test that you have access to the nodes run the following command:

   ```
   kubectl get nodes
   ```

The output of command is:

```
NAME                                 STATUS    ROLES     AGE     VERSION
ip-192-168-107-235.ec2.internal      Ready     none      1h    v1.32.3-eks-473151a
ip-192-168-165-40.ec2.internal       Ready     none      1h    v1.32.3-eks-473151a
ip-192-168-98-54.ec2.internal        Ready     none      1h    v1.32.1-eks-5d632ec
```

Step 2: Prepare your EKS cluster for AWS Batch

All steps are required and must be done in AWS CloudShell.

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

   Use `kubectl` to create a new namespace.

   ```
   $ 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.

```
$ 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: ["events"]
    verbs: ["list"]
  - 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:
  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-test-cluster \
    --arn "arn:aws:iam::<your-account-ID>:role/AWSServiceRoleForBatch" \
    --username aws-batch
```

Output:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account-ID>: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 3: Create 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.

- For `subnets` run `eksctl get cluster` *`my-test-cluster`* to get the subnets used by the cluster.
- For `securityGroupIds` parameter you can use the same security group as the Amazon EKS cluster. This command retrieves the security group ID for the cluster.

```
$ aws eks describe-cluster \
    --name my-test-cluster \
    --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

- Use the ARN of the `instanceRole` you created in the Prerequisites.

```
$ cat <<EOF > ./batch-eks-compute-environment.json
{
  "computeEnvironmentName": "My-Eks-CE1",
  "type": "MANAGED",
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:us-east-1:<your-account-ID>:cluster/my-test-cluster",
    "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

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

Step 4: Create a job queue and attach the compute environment

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
```

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.

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 5: Create an Amazon ECR with pull through cache

Because the cluster doesn't have public internet access you have to create an Amazon ECR for container images. The following directions create an Amazon ECR with a pull-through cache rule to store the image.

1. The following command create the pull-through cache rule. You can replace *`tutorial-prefix`* with a different prefix.

   ```
   aws ecr create-pull-through-cache-rule \
       --ecr-repository-prefix "my-prefix" \
       --upstream-registry-url "public.ecr.aws" \
       --region us-east-1
   ```

2. Authenticate with the public ECR.

   ```
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --
   password-stdin <your-account-ID>.dkr.ecr.us-east-1.amazonaws.com
   ```

   Now you can pull an image.

   ```
   docker pull <your-account-ID>.dkr.ecr.us-east-1.amazonaws.com/my-prefix/
   amazonlinux/amazonlinux:2
   ```

3. You can verify the repositry and image by running the following commands:

   ```
   aws ecr describe-repositories
   ```

   ```
   aws ecr describe-images --repository-name my-prefix/amazonlinux/amazonlinux
   ```

4. The image string to use for pulling the container is in the following format:

   ```
   <your-account-ID>.dkr.ecr.us-east-1.amazonaws.com/my-prefix/amazonlinux/
   amazonlinux:2
   ```

Step 6: Register a job definition

The following Job definition instructs the pod to sleep for 60 seconds.

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": [
        {
          "image": "<your-account-ID>.dkr.ecr.us-east-1.amazonaws.com/my-prefix/
amazonlinux/amazonlinux:2",
          "command": [
            "sleep",
            "60"
          ],
          "resources": {
            "limits": {
              "cpu": "1",
              "memory": "1024Mi"
            }
          }
        }
      ],
      "metadata": {
        "labels": {
          "environment": "test"
        }
      }
    }
  }
}
EOF
```

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

Notes

- There are considerations for the `cpu` and `memory` parameters. For more information, see Memory and vCPU considerations for AWS Batch on Amazon EKS.

Step 7: Submit a job to run

Run the following AWS CLI command in AWS CloudShell to submit a new Job and returns the unique JobID.

```
$ aws batch submit-job --job-queue My-Eks-JQ1 \
    --job-definition MyJobOnEks_Sleep - -job-name My-Eks-Job1
```

Notes

- For more information about running jobs on Amazon EKS resources, see Amazon EKS jobs.

Step 8: View the Job's output

To check the status of a Job:

```
$ aws batch describe-jobs --job <JobID-from-submit-response>
```

The `startedAt` and `stoppedAt` should be one minute apart.

Step 9: (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",
  "eksPropertiesOverride": {
    "podProperties": {
      "containers": [
        {
          "command": [
            "/bin/sh"
          ],
          "args": [
            "-c",
            "echo hello world"
          ]
        }
      ]
    }
  }
}
EOF
```

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

Notes

- 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.

Step 10: Clean up your tutorial resources

You are charged for the Amazon EC2 instance while it is enabled. You can delete the instance to stop incurring charges.

To delete the resources you created, do the following:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane choose **Job queue**.

3. In the **Job queue** table choose the Job queue you created for the tutorial.
4. From **Actions** choose **Disable**. Once the Job queue **State** is Disabled you can choose **Delete**.
5. Once the Job queue is deleted, in the navigation pane choose **Compute environments**.
6. Choose the compute environment you created for this tutorial and then choose **Disable** from **Actions**. It may take 1–2 minuets for the compute environment to complete being disabled.
7. Once the compute environment's **State** is Disabled, choose **Delete**. It may take 1–2 minuets for the compute environment to be deleted.

Additional resources

After you complete the tutorial, you might want to explore the following topics::

- Learn more about the Best practices.
- Explore the AWS Batch core components. For more information, see Components of AWS Batch.
- Learn more about the different Compute Environments available in AWS Batch.
- Learn more about Job queues and their different scheduling options.
- Learn more about Job definitions and the different configuration options.
- Learn more about the different types of Jobs.

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 based on pod status, see *How do I troubleshoot the pod status in Amazon EKS?*.

Getting started with AWS Batch on SageMaker AI

AWS Batch service jobs enable you to submit SageMaker Training jobs through AWS Batch job queues with scheduling, prioritization, and queuing capabilities. This tutorial demonstrates how to set up and run a simple SageMaker Training job using AWS Batch service jobs.

Contents

- Overview
- Prerequisites
- Step 1: Create a SageMaker AI execution role
- Step 2: Create your service environment
- Step 3: Create your SageMaker job queue
- Step 4: Create and submit a training job
- Step 5: Monitor job status
- Step 6: View job output
- Step 7: Clean up your tutorial resources
- Additional resources

Overview

This tutorial demonstrates how to setup AWS Batch service jobs for SageMaker Training jobs using the AWS CLI.

Intended Audience

This tutorial is designed for data scientists and developers responsible for setting up and running machine learning training jobs at scale.

Features Used

This tutorial shows you how to use the AWS CLI to:

- Create a service environment for SageMaker Training jobs
- Create a SageMaker Training job queue
- Submit service jobs using the `SubmitServiceJob` API
- Monitor job status and view outputs
- Access CloudWatch logs for training jobs

Time Required

It should take about 15 minutes to complete this tutorial.

Regional Restrictions

This tutorial can be completed in any AWS Region where both AWS Batch and SageMaker AI are available.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur costs for the following resources:

Description	Cost (US dollars)
SageMaker AI Training instances	You pay for each SageMaker AI Training instance used. For more information about pricing, see SageMaker AI Pricing.
Amazon S3 storage	Minimal cost for storing training job outputs. For more information, see Amazon S3 Pricing.

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 SageMaker AI resources.

- **AWS CLI** – A command line tool for working with AWS services, including AWS Batch and SageMaker AI. This guide requires that you use version 2.8.6 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*.

Step 1: Create a SageMaker AI execution role

SageMaker AI uses execution roles to perform operations on your behalf using other AWS services. You must create an execution role and grant SageMaker AI permissions to use the services and resources needed for training jobs. Use the `AmazonSageMakerFullAccess` managed policy as it includes permissions for Amazon S3.

> **Note**
>
> Use the following directions to create the SageMaker AI execution role for this tutorial. Before you create an execution role for your production environment we recommend you review, How to use SageMaker AI execution roles in the *SageMaker AI Developer guide*.

1. **Create the IAM role**

   Create a JSON file named `sagemaker-trust-policy.json` with the following trust policy:

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "sagemaker.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

   Create the IAM role using the trust policy:

   ```
   aws iam create-role \
       --role-name SageMakerExecutionRole \
       --assume-role-policy-document file://sagemaker-trust-policy.json \
       --description "Execution role for SageMaker training jobs"
   ```

2. **Attach managed policies**

   Attach the required managed policies to the role:

```
aws iam attach-role-policy \
    --role-name SageMakerExecutionRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonSageMakerFullAccess
```

```
aws iam attach-role-policy \
    --role-name SageMakerExecutionRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess
```

3. **Note the role ARN**

   Get the role ARN, which you'll need in later steps:

   ```
   aws iam get-role --role-name SageMakerExecutionRole --query 'Role.Arn' --output
    text
   ```

   Save this ARN as you'll use it when creating your training job payload.

Step 2: Create your service environment

A service environment defines the capacity constraints for SageMaker Training jobs. The service environment encapsulates the maximum number of training instances that can run concurrently.

> ⚠ **Important**
>
> When you create your first service environment for SageMaker Training, AWS Batch automatically creates a service-linked role called `AWSServiceRoleForAWSBatchWithSagemaker` in your account. This role allows AWS Batch to queue and manage SageMaker Training jobs on your behalf. For more information about this service-linked role and its permissions, see the section called “SageMaker integration role”.

Create a service environment that can handle up to 5 instances:

```
aws batch create-service-environment \
    --service-environment-name TutorialServiceEnvironment \
    --service-environment-type SAGEMAKER_TRAINING \
    --capacity-limits capacityUnit=NUM_INSTANCES,maxCapacity=5
```

Output:

```
{
    "serviceEnvironmentName": "TutorialServiceEnvironment",
    "serviceEnvironmentArn": "arn:aws:batch:your-region:your-account-id:service-
environment/TutorialServiceEnvironment"
}
```

Verify that your service environment was created successfully:

```
aws batch describe-service-environments --service-
environments TutorialServiceEnvironment
```

Output:

```
{
    "serviceEnvironments": [
        {
            "serviceEnvironmentName": "TutorialServiceEnvironment",
            "serviceEnvironmentArn": "arn:aws:batch:your-region:your-account-
id:service-environment/TutorialServiceEnvironment",
            "serviceEnvironmentType": "SAGEMAKER_TRAINING",
            "state": "ENABLED",
            "status": "VALID",
            "capacityLimits": [
                {
                    "maxCapacity": 5,
                    "capacityUnit": "NUM_INSTANCES"
                }
            ],
            "tags": {}
        }
    ]
}
```

For more information about service environments, see *Service environments*.

Step 3: Create your SageMaker job queue

A SageMaker job queue manages the scheduling and execution of service jobs. Jobs submitted to this queue will be dispatched to your service environment based on available capacity.

Create a SageMaker Training job queue:

```
aws batch create-job-queue \
    --job-queue-name my-sm-training-fifo-jq \
    --job-queue-type SAGEMAKER_TRAINING \
    --priority 1 \
    --service-environment-order order=1,serviceEnvironment=TutorialServiceEnvironment
```

Output:

```
{
    "jobQueueName": "my-sm-training-fifo-jq",
    "jobQueueArn": "arn:aws:batch:your-region:your-account-id:job-queue/my-sm-training-
fifo-jq"
}
```

Verify that your job queue was created successfully:

```
aws batch describe-job-queues --job-queues my-sm-training-fifo-jq
```

Output:

```
{
    "jobQueues": [
        {
            "jobQueueName": "my-sm-training-fifo-jq",
            "jobQueueArn": "arn:aws:batch:your-region:your-account-id:job-queue/my-sm-
training-fifo-jq",
            "state": "ENABLED",
            "status": "VALID",
            "statusReason": "JobQueue Healthy",
            "priority": 1,
            "computeEnvironmentOrder": [],
            "serviceEnvironmentOrder": [
                {
                    "order": 1,
                    "serviceEnvironment": "arn:aws:batch:your-region:your-account-
id:service-environment/TutorialServiceEnvironment"
                }
            ],
            "jobQueueType": "SAGEMAKER_TRAINING",
            "tags": {}
        }
    ]
}
```

For more information about SageMaker job queues, see [the section called “Create a SageMaker job queue”](#).

Step 4: Create and submit a training job

Now you'll create a simple training job and submit it to your job queue. This example uses a basic "hello world" training job that demonstrates the service job functionality.

Create a file named *`my_training_job.json`* with the following content. Replace *`your-account-id`* with your AWS account ID:

> **Note**
>
> `S3OutputPath` is required for creating the SageMaker Training job but the results of this tutorial are not stored in the Amazon S3 bucket and you can use the path in the following JSON. In your production environment you will need a valid Amazon S3 bucket to store output there if you choose to.

```
{
    "TrainingJobName": "my-simple-training-job",
    "RoleArn": "arn:aws:iam::your-account-id:role/SageMakerExecutionRole",
    "AlgorithmSpecification": {
        "TrainingInputMode": "File",
        "TrainingImage": "763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-
training:2.0.0-cpu-py310",
        "ContainerEntrypoint": [
            "echo",
            "hello world"
        ]
    },
    "ResourceConfig": {
        "InstanceType": "ml.c5.xlarge",
        "InstanceCount": 1,
        "VolumeSizeInGB": 1
    },
    "OutputDataConfig": {
        "S3OutputPath": "s3://your-s3-bucket/output"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 30
    }
}
```

Submit the training job using the SubmitServiceJob API:

```
aws batch submit-service-job \
    --job-queue my-sm-training-fifo-jq \
    --job-name my-batch-sm-job \
    --service-job-type SAGEMAKER_TRAINING \
    --retry-strategy attempts=1 \
    --timeout-config attemptDurationSeconds=60 \
    --service-request-payload file://my_training_job.json
```

Output:

```
{
    "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-job-id",
    "jobName": "my-batch-sm-job",
    "jobId": "your-job-id"
}
```

For more information about service job payloads, see the section called “Service job payloads”. For more information about submitting service jobs, see the section called “Submit a service job”.

Step 5: Monitor job status

You can monitor your training jobs using the following AWS Batch APIs: DescribeServiceJob, ListServiceJobs, and GetJobQueueSnapshot. This section shows different ways to check job status and queue information.

View running jobs in your queue:

```
aws batch list-service-jobs \
    --job-queue my-sm-training-fifo-jq --job-status RUNNING
```

Output:

```
{
    "jobSummaryList": [
        {
            "latestAttempt": {
                "serviceResourceId": {
                    "name": "TrainingJobArn",
                    "value": "arn:aws:sagemaker:your-region:your-account-id:training-
job/AWSBatch<my-simple-training-job><your-attempt-id>"
                }
            },
            "createdAt": 1753718760,
            "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-job-
id",
            "jobId": "your-job-id",
            "jobName": "my-batch-sm-job",
            "serviceJobType": "SAGEMAKER_TRAINING",
            "status": "RUNNING",
            "startedAt": 1753718820
        }
    ]
}
```

View jobs that are in the `RUNNABLE` state:

```
aws batch list-service-jobs \
    --job-queue my-sm-training-fifo-jq --job-status RUNNABLE
```

Get a snapshot of upcoming jobs in your queue:

```
aws batch get-job-queue-snapshot --job-queue my-sm-training-fifo-jq
```

Output:

```
{
    "frontOfQueue": {
        "jobs": [
            {
                "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-
job-id",
                "earliestTimeAtPosition": 1753718880
            },
            {
                "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-
job-id-2",
                "earliestTimeAtPosition": 1753718940
            }
        ],
        "lastUpdatedAt": 1753718970
    }
}
```

Search for jobs by name:

```
aws batch list-service-jobs \
    --job-queue my-sm-training-fifo-jq \
    --filters name=JOB_NAME,values="my-batch-sm-job"
```

Output:

```
{
    "jobSummaryList": [
        {
            "latestAttempt": {
                "serviceResourceId": {
                    "name": "TrainingJobArn",
                    "value": "arn:aws:sagemaker:your-region:your-account-id:training-
job/AWSBatch<my-simple-training-job><your-attempt-id>"
                }
            },
            "createdAt": 1753718760,
            "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-job-
id",
            "jobId": "your-job-id",
            "jobName": "my-batch-sm-job",
            "serviceJobType": "SAGEMAKER_TRAINING",
            "status": "RUNNING"
        }
    ]
}
```

For more information about job state mapping, see the section called “Service job status”.

Step 6: View job output

Once your job completes, you can view its output and logs through both AWS Batch and SageMaker AI APIs.

Get detailed information about your job from AWS Batch:

```
aws batch describe-service-job \
    --job-id your-job-id
```

Output:

```
{
    "attempts": [
        {
            "serviceResourceId": {
                "name": "TrainingJobArn",
                "value": "arn:aws:sagemaker:your-region:your-account-id:training-job/
AWSBatch<my-simple-training-job><your-attempt-id>"
            },
            "startedAt": 1753718820,
            "stoppedAt": 1753718880,
            "statusReason": "Received status from SageMaker: Training job completed"
        }
    ],
    "createdAt": 1753718760,
    "jobArn": "arn:aws:batch:your-region:your-account-id:service-job/your-job-id",
    "jobId": "your-job-id",
    "jobName": "my-batch-sm-job",
    "jobQueue": "arn:aws:batch:your-region:your-account-id:job-queue/my-sm-training-
fifo-jq",
    "latestAttempt": {
        "serviceResourceId": {
            "name": "TrainingJobArn",
            "value": "arn:aws:sagemaker:your-region:your-account-id:training-job/
AWSBatch<my-simple-training-job><your-attempt-id>"
        }
    },
    "retryStrategy": {
        "attempts": 1,
        "evaluateOnExit": []
    },
    "serviceRequestPayload": "your-training-job-request-json",
    "serviceJobType": "SAGEMAKER_TRAINING",
    "startedAt": 1753718820,
    "status": "SUCCEEDED",
    "statusReason": "Received status from SageMaker: Training job completed",
    "stoppedAt": 1753718880,
    "tags": {},
    "timeoutConfig": {
        "attemptDurationSeconds": 60
    }
}
```

This command returns comprehensive job information including the SageMaker Training job ARN, which you can use to access the job directly through SageMaker AI:

```
aws sagemaker describe-training-job \
    --training-job-name AWSBatch<my-simple-training-job><your-attempt-id>
```

To view the CloudWatch logs for your training job, first get the log stream name:

```
aws logs describe-log-streams \
    --log-group-name /aws/sagemaker/TrainingJobs \
    --log-stream-name-prefix AWSBatchmy-simple-training-job
```

Output:

```
{
    "logStreams": [
        {
            "logStreamName": "your-log-stream-name",
            "creationTime": 1753718830,
            "firstEventTimestamp": 1753718840,
            "lastEventTimestamp": 1753718850,
            "lastIngestionTime": 1753718860,
            "uploadSequenceToken": upload-sequence-token,
            "arn": "arn:aws:logs:your-region:your-account-id:log-group:/aws/sagemaker/
TrainingJobs:log-stream:AWSBatch<my-simple-training-job><your-attempt-id>/algo-1-algo-
id",
            "storedBytes": 0
        }
    ]
}
```

Then retrieve the logs using the log stream name from the previous response:

```
aws logs get-log-events \
    --log-group-name /aws/sagemaker/TrainingJobs \
    --log-stream-name your-log-stream-name
```

Output:

```
{
    "events": [
        {
            "timestamp": 1753718845,
            "message": "hello world",
            "ingestionTime": 1753718865
        }
    ],
    "nextForwardToken": "next-forward-token",
    "nextBackwardToken": "next-backward-token"
}
```

The log output shows the "hello world" message from your training job, confirming that the job executed successfully.

Step 7: Clean up your tutorial resources

When you're done with the tutorial, clean up the resources you created to avoid ongoing charges.

First, disable and delete the job queue:

```
aws batch update-job-queue \
    --job-queue my-sm-training-fifo-jq \
    --state DISABLED
```

Wait for the job queue to be disabled, then delete it:

```
aws batch delete-job-queue \
    --job-queue my-sm-training-fifo-jq
```

Next, disable and delete the service environment:

```
aws batch update-service-environment \
    --service-environment TutorialServiceEnvironment \
    --state DISABLED
```

Wait for the service environment to be disabled, then delete it:

```
aws batch delete-service-environment \
    --service-environment TutorialServiceEnvironment
```

Additional resources

After you complete the tutorial, you might want to explore the following topics:

- We recommend using the PySDK for service job creation and submission to your Job queue because PySDK has helper classes and utilities. For an example of using PySDK, see SageMaker AI examples on GitHub.
- Learn more about the section called “Service jobs”.
- Explore Service job payloads in AWS Batch for more complex training job configurations.
- Learn about Submit a service job in AWS Batch and the `SubmitServiceJob` API.
- Review Mapping AWS Batch service job status to SageMaker AI status to understand job state transitions.
- Visit the SageMaker AI Python SDK documentation for more feature-rich ways to create and submit SageMaker Training jobs using Python.
- Explore SageMaker example notebooks for more complex machine learning workflows.

The AWS Batch widget dashboard

Through 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 environments for AWS Batch*.

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

Topics

- How to add the Single job queue widget to the AWS Batch dashboard
- How to add the CloudWatch Container Insights widget to the AWS Batch dashboard
- How to add the Job logs widget to the AWS Batch dashboard

How to add the Single job queue widget to the AWS Batch dashboard

The Single job queue 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, choose 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**.

How to add the CloudWatch Container Insights widget to the AWS Batch dashboard

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

To add this widget, follow these steps.

1. Open the AWS Batch console.
2. From the navigation bar, choose 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**.

How to add the Job logs widget to the AWS Batch dashboard

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

To add this widget, follow these steps.

1. Open the AWS Batch console.
2. From the navigation bar, choose 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**.

Compute environments for AWS Batch

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
- Create a compute environment
- Update a compute environment in AWS Batch
- Compute resource AMIs
- Use Amazon EC2 launch templates with AWS Batch
- Instance Metadata Service (IMDS) configuration
- EC2 configurations
- Instance type allocation strategies for AWS Batch
- Compute resource memory management
- Fargate compute environments
- Amazon EKS compute environments

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.

⚠ **Important**

AWS Batch creates and manages multiple AWS resources on your behalf and within your account, including Amazon EC2 Launch Templates, Amazon EC2 Auto Scaling Groups, Amazon EC2 Spot Fleets, and Amazon ECS Clusters. These managed resources are configured specifically to ensure optimal AWS Batch operation. Manually modifying these Batch-managed resources, unless explicitly stated in AWS Batch documentation, may result in unexpected behavior resulting in `INVALID` Compute Environment, suboptimal instance scaling behavior, delayed workload processing, or unexpected costs. These manual modifications can not be deterministically supported by the AWS Batch service. Always use the supported Batch APIs or the Batch console to manage your 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 in the *Amazon VPC User Guide*. For more information about how to create a VPC, see Create a virtual private cloud .

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.

Note

AWS Batch doesn't automatically upgrade the AMIs in a compute environment after it's created. For example, it doesn't update the AMIs in your compute environment when a newer version of the Amazon ECS optimized AMI is released. 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. There are two ways to use a new AMI for your AWS Batch jobs. The original method is to complete these steps:

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 Update a compute environment in AWS Batch. 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. AWS Batch supports unmanaged compute environments for both Amazon ECS and Amazon EKS, allowing you to maintain control over your infrastructure while leveraging Batch's job scheduling capabilities.

Note

AWS Fargate resources aren't supported in unmanaged compute environments.

Unmanaged Amazon ECS compute environments

For unmanaged Amazon ECS compute environments, 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 Tutorial: Create a compute resource AMI.

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
```

Unmanaged Amazon EKS compute environments

In an unmanaged Amazon EKS compute environment, you manage your own Kubernetes nodes while AWS Batch handles job scheduling and placement. It allows you to directly control your Kubernetes infrastructure for security, compliance, or operational requirements. You are responsible for provisioning and configuring your Amazon EKS nodes, while AWS Batch integrates with your existing Amazon EKS cluster to schedule and run jobs.

For more information, see Tutorial: Create an unmanaged compute environment using Amazon EKS resources.

Create 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:
>
> - 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.

Topics

- Tutorial: Create a managed compute environment using Fargate resources
- Tutorial: Create a managed compute environment using Amazon EC2 resources
- Tutorial: Create an unmanaged compute environment using Amazon EC2 resources
- Tutorial: Create a managed compute environment using Amazon EKS resources
- Tutorial: Create an unmanaged compute environment using Amazon EKS resources
- Resource: Compute environment template
- Instance type compute table

Tutorial: Create a managed compute environment using Fargate resources

Complete the following steps to create a managed compute environment using AWS Fargate resources.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 Using service-linked roles 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**
   >
   > 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

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**.

Tutorial: Create a managed compute environment using Amazon EC2 resources

Complete the following steps to create a managed compute environment using Amazon Elastic Compute Cloud (Amazon EC2) resources.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, choose **Environments**.
4. Choose **Create environment** and then **Compute environment**.

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. 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 Using service-linked roles for AWS Batch.

    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. If you choose to create a new instance profile, the required role (`ecsInstanceRole`) is created for you.

    f. (Optional) Expand **Tags**.

        i. (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**.

        ii. (Optional) For **Tags**, choose **Add tag**. Then, enter a **Key** name and optional **Value**. Choose **Add tag**.

        For more information, see Tag your AWS Batch resources.

    g. Choose **Next**.

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 "Tag 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`.

   AWS Batch can select the instance type for you if you choose one of the following:

   - `optimal` to select instance types (from the `c4`, `m4`, `r4`, `c5`, `m5`, and `r5` instance families) that match the demand of your job queues.
   - `default_x86_64` to choose x86 based instance types (from the m6i, c6i, r6i, and c7i instance families) that matches the resource demands of the job queue.

- `default_arm64` to choose x86 based instance types (from the m6g, c6g, r6g, and c7g instance families) that matches the resource demands of the job queue.

Note

Starting on 11/01/2025 the behavior of `optimal` is going to be changed to match `default_x86_64`. During the change your instance families could be updated to a newer generation. You do not need to perform any actions for the upgrade to happen. For more information about change, see

> **Note**
>
> Starting on 11/01/2025 the behavior of `optimal` is going to be changed to match `default_x86_64`. During the change your instance families could be updated to a newer generation. You do not need to perform any actions for the upgrade to happen.

AWS Batch supported a single option in **instanceTypes** for `optimal` to match the demand of your job queues. We've introduced two new instance type options: `default_x86_64` and `default_arm64`. We will use `default_x86_64` if you make no instance type selection. These new options will automatically select cost-effective instance types across different families and generations based on your job queue requirements, allowing you to get your workloads running quickly.

As sufficient capacity of new instance types become available in an AWS Region, the corresponding default pool will be automatically updated with the new instance type. The existing `optimal` option will continue to be supported and is not being deprecated, as it will be supported by the underlying default pools to provide updated instances going forward. If you are using '`optimal`, no action is needed on your part.

However, please be aware that only `ENABLED` and `VALID` Compute Environments (CEs) will be automatically updated with new instance types. If you have any `DISABLED` or `INVALID` CEs, they will receive updates once they are re-enabled and set to a `VALID` state.

.

Note

- Instance family availability varies by AWS Region. For example, some AWS Regions may not have any fourth generation instance families but have fifth and sixth generation instance families.
- When using `default_x86_64` or `default_arm64` instance bundles, AWS Batch selects instance families based on a balance of cost-effectiveness and performance. While newer generation instances often provide better price-performance, AWS Batch may choose an earlier generation instance family if it provides the optimal combination of availability, cost, and performance for your workload. For example, in an AWS Region where both c6i and c7i instances are available, AWS Batch might select c6i instances if they offer better cost-effectiveness for your specific job requirements. For more information on AWS Batch instance types and AWS Region availability, see Instance type compute table.
- AWS Batch periodically updates your instances in default bundles to newer, more cost-effective options. Updates happen automatically without requiring any action from you. Your workloads continue running during updates with no interruption.

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 p3, p4, p5, p6, g3, g3s, g4, g5, or g6 families.

h. 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 “Instance type allocation strategies”.

i. Expand **Additional configuration**.

   i. (Optional) For **Placement group**, enter a placement group name to group resources in the compute environment.

   ii. (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.

   iii. (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 2023]

AWS Batch supports Amazon Linux 2023.

> **Note**
>
> Amazon Linux 2023 does not support A1 instances.

[Amazon Linux 2023 (GPU)]

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

> **Note**
>
> The AMI that you choose for a compute environment must match the architecture of the instance types that you intend 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*.

j. (Optional) Expand **Launch templates**

   i. For **Default 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 Use Amazon EC2 launch templates with AWS Batch.

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

ii. (Optional) For **Default version**, enter `$Default`, `$Latest`, or a specific version number to use.

> **Note**
>
> Note: If you use either substitution variable ($Default or $Latest), they will apply the current default or latest version number at the time that this configuration is saved. If the default or latest version changes in the future, you must update the information - it won't automatically update.

> **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 “AMI selection during infrastructure updates”.

iii. (Optional) For **Override launch template** choose **Add override launch template**

A. (Optional) For **Launch template**, select an existing Amazon EC2 launch template to use for specific instance types and families.

B. (Optional) For **Default version**, enter a specific version number to use, `$Default`, or `$Latest`.

> **Note**
>
> If you use either the `$Default` or `$Latest` variable, AWS Batch will apply the current information at the time that the compute environment is created. If the default or latest version changes in the future, you must

update the information through UpdateComputeEnvironment or through the AWS Management Console - AWS Batch.

C. (Optional) For **Target instance types**, select the instance type or family that you want to apply the override launch template.

> **Note**
>
> If you specify an override launch template, **Target instance types** is required. For more information, see LaunchTemplateSpecificationOverride.targetInstanceTypes.

> **Note**
>
> If the instance type or family that you want to select doesn't appear in this list, review the selections you made in `Allowed instance types`.

k. Choose **Next**.

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.

> **Note**
>
> AWS Batch on Amazon EC2 supports Local Zones. For more information, see Local Zones in the *Amazon EC2 User Guide* and Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts in the *Amazon Elastic Container Service Developer Guide*.

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.

> **Note**
>
> Note: If you use either substitution variable ($Default or $Latest), they will apply the current default or latest version number at the time that this configuration is saved. If the default or latest version changes in the future, you must update the information - it won't automatically update.

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**.

Tutorial: Create an unmanaged compute environment using Amazon EC2 resources

Complete the following steps to create an unmanaged compute environment using Amazon Elastic Compute Cloud (Amazon EC2) resources.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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. For **Service role**, choose a role that lets the AWS Batch service make calls to the required AWS API operations on your behalf.

   > **Note**
   >
   > You can't use `AWSServiceRoleForBatch` for unmanaged compute environments.

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 Tag 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**.

Tutorial: Create a managed compute environment using Amazon EKS resources

Complete the following steps to create a managed compute environment using Amazon Elastic Kubernetes Service (Amazon EKS) resources.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 (such as `c5.8xlarge`). Metal instance types aren't in the instance families. For example, `c5` doesn't include `c5.metal`.

    AWS Batch can select the instance type for you if you choose one of the following:

- `optimal` to select instance types (from the `c4`, `m4`, `r4`, `c5`, `m5`, and `r5` instance families) that match the demand of your job queues.
- `default_x86_64` to choose x86 based instance types (from the m6i, c6i, r6i, and c7i instance families) that matches the resource demands of the job queue.
- `default_arm64` to choose x86 based instance types (from the m6g, c6g, r6g, and c7g instance families) that matches the resource demands of the job queue.

> **Note**
>
> Starting on 11/01/2025 the behavior of `optimal` is going to be changed to match `default_x86_64`. During the change your instance families could be updated to a newer generation. You do not need to perform any actions for the upgrade to happen. For more information about change, see Optimal instance type configuration to receive automatic instance family updates.

> **Note**
>
> - Instance family availability varies by AWS Region. For example, some AWS Regions may not have any fourth generation instance families but have fifth and sixth generation instance families.
> - When using `default_x86_64` or `default_arm64` instance bundles, AWS Batch selects instance families based on a balance of cost-effectiveness and performance. While newer generation instances often provide better price-performance, AWS Batch may choose an earlier generation instance family if it provides the optimal combination of availability, cost, and performance for your workload. For example, in an AWS Region where both c6i and c7i instances are available, AWS Batch might select c6i instances if they offer better cost-effectiveness for your specific job requirements. For more information on AWS Batch instance types and AWS Region availability, see Instance type compute table.
> - AWS Batch periodically updates your instances in default bundles to newer, more cost-effective options. Updates happen automatically without requiring any action from you. Your workloads continue running during updates with no interruption

> **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 `p3`, `p4`, `p5`, `p6`, `g3`, `g3s`, `g4`, `g5`, or `g6` families.

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**.

    You can use either an Amazon EKS-optimized Amazon Linux AMI or a custom AMI.

    i. To use an Amazon EKS-optimized Amazon Linux AMI:

    A. For **Image type** choose one of the following:

    - 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 (accelerated): 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 2023: AWS Batch supports Amazon Linux 2023 (AL2023).
    - Amazon Linux 2023 (accelerated): GPU instance families and can be used for all non AWS Graviton-based instance types.

        B. For **Kubernetes version** enter in a Kubernetes version number.

    ii. To use a custom AMI:

        A. For **Image type** choose the AMI type that the custom AMI is based off of:

            - 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 (accelerated): 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 2023: AWS Batch supports AL2023.
            - Amazon Linux 2023 (accelerated): GPU instance families and can be used for all non AWS Graviton-based instance types.

        B. For **Image ID override** enter the custom AMI ID.

        C. For **Kubernetes version** enter in a Kubernetes version number.

d. (Optional) For **Launch template**, choose an existing launch template.

e. (Optional) For **Launch template version**, enter **`$Default`**, **`$Latest`**, or a version number.

f. (Optional) For **Override launch template**, to add an override choose **Add override Launch template**:

    i. (Optional) For **Launch template** choose the launch template to add the override to.

    ii. (Optional) For **Launch template version** choose the version number of the launch template, `$Default`, or `$Latest`.

    iii. (Optional) For **Target instances types** choose the instance type or family that this override should be applied to. This can target only instance types and families that are included in **Allowed instance types**.

    iv. (Optional) For **userdataType** choose the EKS node initialization. Only use this field if you have an AMI specified in either the Launch Template or as a Launch Template Override. Choose **EKS_NODEADM** for custom AMIs based on `EKS_AL2023` or `EKS_AL2023_NVIDIA` or **EKS_BOOSTRAP_SH** for `EKS_AL2` and `EKS_AL_NVIDIA`. The default value is **EKS_BOOSTRAP_SH**.

    You would use **userdataType** when you have a mixed environment where you're using both AL2 and AL2023-based custom AMIs in the same compute environment.

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**.

Tutorial: Create an unmanaged compute environment using Amazon EKS resources

Complete the following steps to create an unmanaged compute environment using Amazon Elastic Kubernetes Service (Amazon EKS) resources.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar on top of the page, 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 Kubernetes Service (Amazon EKS)**.
    b. For **Orchestration type**, choose **Unmanaged**.
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 **EKS cluster**, choose an existing Amazon EKS cluster. To create a new EKS cluster, follow the steps on Create an Amazon EKS cluster page.
8. For **Namespace**, enter a Kubernetes namespace to group your AWS Batch processes in the cluster.
9. (Optional) For **Maximum vCPUs**, specify the maximum number of vCPUs available for job scheduling from your provisioned capacity.
10. (Optional) Expand **Tags**. Choose **Add tag** and then enter a key-value pair.
11. Choose **Next page**.
12. For **Review**, review the configuration steps. If you need to make changes, choose **Edit**. When you're finished, choose **Create compute environment**.

Assigning Amazon EKS cluster nodes to unmanaged compute environment

After creating the unmanaged compute environment, you need to label your Amazon EKS nodes with the compute environment UUID.
First, get the compute environment UUID from the `DescribeComputeEnvironments` API result:

```
$ aws batch describe-compute-environments \
    --compute-environments unmanagedEksCE \
    --query "computeEnvironments[].{name: computeEnvironmentName, uuid: uuid}"
```

Get the node information:

```
kubectl get nodes -o name
```

Label the nodes with the AWS Batch compute environment UUID:

```
kubectl label <node-name> batch.amazonaws.com/compute-environment-uuid=uuid
```

Resource: 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*.

Note

You can generate a compute environment template with the following AWS CLI command.

```
$ aws batch create-compute-environment --generate-cli-skeleton
```

```
{
    "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": ""
    }
}
```

Instance type compute table

The following table lists the AWS Region, instance family keyword, and available instance families. AWS Batch will try to allocate an instance from the latest family but because instance family availability varies by AWS Region you may get an earlier instance family generation.

default_x86_64

Region	Instance families
All AWS Regions that support AWS Batch	m6i, c6i, r6i c7i

default_arm64

Region	Instance families
All AWS Regions that support AWS Batch	m6g, c6g, r6g c7g

Optimal

Region	Instance families
ap-northeast-1 ap-northeast-2 ap-south-1 ap-southeast-1 ap-southeast-2 ca-central-1 cn-north-1 cn-northwest-1 eu-central-1 eu-west-1 eu-west-2 sa-east-1 us-east-1 us-east-2 us-gov-west-1 us-west-1 us-west-2	m4, c4, r4
af-south-1 ap-east-1 ap-northeast-3 ap-south-2 ap-southeast-3 ap-southeast-4 ca-west-1 eu-central-2 eu-north-1 eu-south-1	m5, c5, r5

Region	Instance families
eu-south-2 eu-west-3 il-central-1 me-central-1 me-south-1 us-gov-east-1 us-isob-east-1 us-iso-east-1 us-isof-south-1 us-isof-east-1 eu-isoe-west-1 us-northeast-1	
ap-southeast-5 ap-southeast-7 ap-east-2 mx-central-1	m6, c6, r6

Update a compute environment in AWS Batch

AWS Batch provides multiple strategies for updating compute environments, each designed for specific update scenarios and requirements. These approaches use the same underlying update API but represent different prescriptive methods for managing updates effectively. You can manage these updates using the AWS Batch console or the AWS CLI. Understanding these strategies helps you choose the most appropriate method for your needs while minimizing disruption to your workloads.

This topic provides an overview of the available update strategies and guidance on when to use each approach. For detailed procedures, see the individual sections for each update strategy.

⚠ **Important**

AWS Batch creates and manages multiple AWS resources on your behalf and within your account, including Amazon EC2 Launch Templates, Amazon EC2 Auto Scaling Groups, Amazon EC2 Spot Fleets, and Amazon ECS Clusters. These managed resources are configured specifically to ensure optimal AWS Batch operation. Manually modifying these AWS Batch-managed resources, unless explicitly stated in AWS Batch documentation, can result in unexpected behavior, including `INVALID` compute environments, suboptimal instance scaling behavior, delayed workload processing, or unexpected costs. These manual modifications can't be deterministically supported by the AWS Batch service. Always use the supported AWS Batch APIs or the AWS Batch console to manage your compute environments.

Topics

- Compute environment update strategies
- Choosing the right update strategy
- Perform scaling updates
- Perform infrastructure updates
- Perform blue/green updates for compute environments

Compute environment update strategies

When you use scaling or infrastructure updates your compute environment is updated in place. For the blue/green update strategy you are creating a new compute environment (green) and then migrating your workload from the old compute environment (blue) to the new compute environment (green).

AWS Batch provides three different strategies for compute environment updates:

Scaling updates

> Scaling updates adjust the capacity of your compute environment by adding or removing instances without replacing existing instances. This is the fastest update scenario and requires no downtime. Use scaling updates when you need to change capacity settings (vCPUs). These updates typically complete within minutes.

Fargate updates are performed using the same procedures as scaling updates. For more information, see Perform scaling updates.

Infrastructure updates

Infrastructure updates replace instances in your compute environment with new instances that have updated settings. These updates require specific service role and allocation strategy configurations but provide minimal downtime, with running jobs potentially interrupted. Use infrastructure updates when you need to modify instance types, AMI configuration, networking settings, service role, environment state, or other infrastructure components. These updates typically complete in 10-30 minutes depending on job completion.

For more information, see Perform infrastructure updates.

Blue/green updates

Blue/green updates create a new compute environment alongside your existing environment, allowing gradual workload transition with zero downtime. This approach provides the safest update path but requires running two environments temporarily. Use blue/green updates when you need zero downtime, want to test changes before full deployment, require quick rollback capability, or are using unsupported configurations for infrastructure updates. The time to complete is variable and controlled by you.

For more information, see Perform blue/green updates for compute environments.

Choosing the right update strategy

Use this decision guide to select the most appropriate update strategy for your needs:

Choose scaling updates when

Choose the scaling update strategy when you only need to adjust compute capacity (vCPUs). Scaling updates are ideal when you need quick updates with no downtime and no infrastructure configuration changes are needed.

For detailed procedures, see Perform scaling updates.

Choose infrastructure updates when

Choose the infrastructure update strategy when you need to modify instance types, AMI settings, service role, environment state, or networking configuration.

Your environment must use the *AWSServiceRoleForBatch* service-linked role and an allocation strategy of `BEST_FIT_PROGRESSIVE`, `SPOT_CAPACITY_OPTIMIZED`, or `SPOT_PRICE_CAPACITY_OPTIMIZED`. Infrastructure updates work well when some job interruption is acceptable during the update and you want automatic updates to the latest Amazon ECS-optimized AMI.

For detailed procedures, see Perform infrastructure updates.

Choose blue/green updates when

Choose the blue/green update strategy when zero downtime is required for your workloads or you need to test changes before transitioning production workloads. This approach is essential when quick rollback capability is important, your environment uses `BEST_FIT` allocation strategy, or your environment doesn't use the *AWSServiceRoleForBatch* service-linked role. Blue/green updates are also the best choice when you're using custom AMIs that require manual updates or need to make major configuration changes.

For detailed procedures, see Perform blue/green updates for compute environments.

AMI update considerations

AWS Batch can update to the latest Amazon ECS-optimized AMI during infrastructure updates when all of these conditions are met:

> **Note**
>
> After the infrastructure update has completed `updateToLatestImageVersion` is set to `false`. To initiate another update `updateToLatestImageVersion` has to be set to `true`.

- The compute environment uses the *AWSServiceRoleForBatch* service-linked role
- The allocation strategy is set to `BEST_FIT_PROGRESSIVE`, `SPOT_CAPACITY_OPTIMIZED`, or `SPOT_PRICE_CAPACITY_OPTIMIZED`
- No AMI ID is explicitly specified in `imageId`, `imageIdOverride`, or launch template
- The `updateToLatestImageVersion` is set to `true`

AMI updates using blue/green deployment

You must use blue/green deployment to update AMIs in these scenarios:

- When using a specific version of the Amazon ECS-optimized AMI
- When the AMI ID is specified in any of:
  - Launch template (must update the template or remove it)
  - The `imageId` parameter
  - The `imageIdOverride` parameter in EC2 configuration
- When using the `BEST_FIT` allocation strategy (doesn't support infrastructure updates)
- When not using the *AWSServiceRoleForBatch* service-linked role

Perform scaling updates

Scaling updates adjust the capacity of your compute environment by adding or removing instances. This is the fastest update strategy and doesn't require replacing existing instances. Scaling updates work with any service role type and allocation strategy, making them the most flexible update option.

Changes that trigger a scaling update

When you modify only the following settings, AWS Batch performs a scaling update. If you modify any of these settings along with other compute environment settings, AWS Batch performs an infrastructure update instead.

The following settings trigger scaling updates when modified exclusively:

- `desiredvCpus` – Sets the target number of vCPUs for the environment.
- `maxvCpus` – Defines the maximum number of vCPUs that can be launched.
- `minvCpus` – Specifies the minimum number of vCPUs to maintain.

For Fargate compute environments, you can also modify these settings for scaling updates:

- `securityGroupIds` – Security group IDs for the compute environment.
- `subnets` – Subnets for the compute environment.

> **Note**
>
> We recommend not using `desiredvCpus` to initiate a scaling update as AWS Batch will dynamically adjust `desiredvCpus`. Instead you should update `minvCpus`.

When updating `desiredvCpus`, the value must be between `minvCpus` and `maxvCpus`. The new value must be greater than or equal to the current `desiredvCpus`. For more information, see the section called “Error message when you update the `desiredvCpus` setting”.

⚠ **Important**

If you modify any of these scaling settings together with other compute environment settings (such as instance types, AMI IDs, or launch templates), AWS Batch performs an infrastructure update instead of a scaling update. Infrastructure updates take longer and may replace existing instances.

Performing scaling updates using the AWS Management Console

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Environments** and then the **Compute environments** tab.
3. Select the compute environment to update.
4. Choose **Actions** and then **Edit**.
5. Modify one or more of the settings that support scaling updates. For example:
   - For **Minimum vCPUs**, enter the minimum number of vCPUs.
   - For **Desired vCPUs**, enter the desired number of vCPUs.
   - For **Maximum vCPUs**, enter the maximum number of vCPUs.
6. Choose **Save changes**.
7. Monitor the compute environment status. The update should complete quickly since it only involves scaling operations.

Performing scaling updates using the AWS CLI

Use the **update-compute-environment** command to perform scaling updates. The following two examples demonstrate common scaling operations. You can modify one or more of the following settings that support scaling updates

- This example updates the desired, minimum, and maximum vCPUs:

```
aws batch update-compute-environment \
    --compute-environment your-compute-environment-name \
    --compute-resources minvCpus=2,maxvCpus=8
```

Monitoring scaling updates

Monitor your scaling updates using the AWS Batch console to view the compute environment status and check instance count and vCPU metrics. You can also use the AWS CLI with the **describe-compute-environments** command to check status and monitor instance counts and vCPU values.

Perform infrastructure updates

Infrastructure updates replace the instances in your compute environment with new instances that have updated settings. This update strategy takes longer than scaling updates and requires specific service role and allocation strategy settings. Infrastructure updates provide a way to modify fundamental compute environment configurations while maintaining service availability.

> ⚠ **Important**
>
> Infrastructure updates require the *AWSServiceRoleForBatch* service-linked role and an allocation strategy of `BEST_FIT_PROGRESSIVE`, `SPOT_CAPACITY_OPTIMIZED`, or `SPOT_PRICE_CAPACITY_OPTIMIZED`. If your environment doesn't meet these requirements, use blue/green updates instead.

Changes that trigger infrastructure updates

When you modify any of the following settings, AWS Batch performs an infrastructure update. Infrastructure updates also occur when you modify these settings together with scaling update settings.

The following settings trigger infrastructure updates:

Compute configuration

- `allocationStrategy` – Determines how AWS Batch selects instance types.
- `instanceTypes` – Specifies which EC2 instance types to use.

- `bidPercentage` – Maximum percentage of On-Demand price for Spot instances.
- `type` – Compute environment type (EC2 or SPOT).

AMI and launch configuration

- `imageId` – Specific AMI to use for instances.
- `ec2Configuration` – EC2 configuration including `imageIdOverride`.
- `launchTemplate` – EC2 launch template settings.
- `ec2KeyPair` – SSH key pair for instance access.
- `updateToLatestImageVersion` – Automatic AMI updates setting.

Networking and security

- `subnets` – VPC subnets where instances are launched (for EC2 compute environments).
- `securityGroupIds` – Security groups for instances (for EC2 compute environments).
- `placementGroup` – EC2 placement group configuration.

Other settings

- `instanceRole` – IAM role for EC2 instances.
- `tags` – Tags applied to EC2 instances.

> ⚠ **Important**
>
> If you modify any infrastructure update settings together with scaling update settings (such as `desiredvCpus`, `maxvCpus`, or `minvCpus`), AWS Batch performs an infrastructure update. Infrastructure updates take longer than scaling updates.

AMI selection during infrastructure updates

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`.

Job handling during updates

Configure how running jobs are handled during an infrastructure update using the update policy. When you set `terminateJobsOnUpdate=true`, running jobs are terminated immediately, the `jobExecutionTimeoutMinutes` setting is ignored, and the update proceeds as soon as instances can be replaced. When you set `terminateJobsOnUpdate=false`, running jobs continue for the specified timeout period with a default timeout of 30 minutes, and jobs are terminated if they exceed the timeout.

> **Note**
>
> To retry jobs that are terminated during an update, configure a job retry strategy. For more information, see the section called "Automated job retries".

Performing infrastructure updates using the AWS Management Console

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Environments** then the **Compute environments** tab.
3. Select the compute environment to update.
4. Choose **Actions** and then **Edit**.
5. In the **Update behavior** section, configure how running jobs are handled:
   - Choose **Update AMI to latest version** to update the AMI to the latest version.
   - Choose **Terminate jobs immediately on update** to terminate jobs when the update process is run.
   - For **Job execution timeout** enter the number of minutes to wait before starting the update process.
6. Modify one or more of the settings that require an infrastructure updates. For example:
   - **Instance role**
   - **Use EC2 Spot instances**
   - **Allowed instance types**
   - **Placement group**
   - **EC2 key pair**
   - **EC2 configuration**
   - **Launch templates**
   - **Subnets**
   - **Security groups**
7. Choose **Save changes**.

8. Monitor the compute environment status. The environment will show `UPDATING` during the update process.

Performing infrastructure updates using the AWS CLI

Use the **update-compute-environment** command with an change to one or more of the settings that require an infrastructure updates. The following three examples are common infrastructure operations.

- This example updates the instance types and configures the update policy:

```
aws batch update-compute-environment \
    --compute-environment your-compute-environment-name \
    --compute-resources instanceTypes=default_x86_64 \
    --update-policy terminateJobsOnUpdate=false,jobExecutionTimeoutMinutes=30
```

- This example updates the VPC subnets and security groups:

```
aws batch update-compute-environment \
    --compute-environment your-compute-environment-name \
    --compute-resources subnets=subnet-abcd1234,subnet-efgh5678
 securityGroupIds=sg-abcd1234 \
    --update-policy terminateJobsOnUpdate=true
```

- This example enables automatic updates to the latest Amazon ECS optimized AMI:

```
aws batch update-compute-environment \
    --compute-environment your-compute-environment-name \
    --compute-resources updateToLatestImageVersion=true \
    --update-policy terminateJobsOnUpdate=false,jobExecutionTimeoutMinutes=60
```

Monitoring infrastructure updates

Monitor your infrastructure updates using the AWS Batch console to watch the compute environment status change to `UPDATING`, monitor instance replacement progress, and check for any failed updates. The update is successful once the compute environment state is `VAILD`. You can also use CloudWatch to track instance termination events and monitor job states during the update. With the AWS CLI, use the **describe-compute-environments** command to check status and monitor instance lifecycle events.

Perform blue/green updates for compute environments

A blue/green update is an update strategy that reduces downtime and risk by creating a new compute environment (green) alongside your existing compute environment (blue). This approach allows you to gradually transition workloads to the new environment while keeping the existing environment operational. Blue/green updates provide the safest update path and work with any service role type or allocation strategy.

Overview

Blue/green updates offer several advantages that make them ideal for production environments. They provide *zero downtime* by keeping your workloads running continuously during the update process. The approach enables *easy rollback* capabilities, allowing you to quickly revert to the original environment if issues arise. You can implement a *gradual transition* strategy, verifying the new environment's performance before fully switching over your production workloads. This method also provides excellent *risk mitigation* since the original environment remains unchanged and operational until you choose to remove it.

When blue/green updates are required

You must use blue/green updates in the following situations:

- When your compute environment uses `BEST_FIT` allocation strategy (doesn't support infrastructure updates)
- When your compute environment doesn't use the *AWSServiceRoleForBatch* service-linked role
- When you need to transition between different service role types

When blue/green updates are recommended

Blue/green updates are particularly recommended for production environments where zero downtime is critical for your workloads. This approach works well when you need to test new configurations before transitioning production workloads, ensuring that changes meet your performance and reliability requirements. Choose blue/green updates when quick rollback capability is important for your operations, especially if you're updating custom AMIs with significant changes. This method is also ideal when you want to validate performance characteristics and behavior before fully committing to changes, providing confidence in your update process.

Prerequisites

Before performing a blue/green update, ensure you have:

- Appropriate IAM permissions to create and manage compute environments
- Access to view and modify job queue settings
- Job retry strategies configured for your job definitions to handle potential failures during the transition. For more information, see Automated job retries.
- The AMI ID for the new compute environment. This can be either:
  - A recent, approved version of the Amazon ECS optimized AMI (used by default)
  - A custom AMI that meets the Amazon ECS container instance AMI specification. When using a custom AMI, you can specify it in one of these ways:
    - Using the **Image ID override** field in the EC2 configuration
    - Specifying it in a launch template

    For more information about creating custom AMIs, see Tutorial: Create a compute resource AMI.

Before creating the new environment, you need to record the configuration of your existing compute environment. You can do this using either the AWS Management Console or the AWS CLI.

> **Note**
>
> The following procedures detail how to perform a blue/green update that only changes the AMI. You can update other settings for the new environment.

> **Important**
>
> When you remove the old (blue) compute environment, any currently running jobs on those instances will fail because the instances will be terminated. Configure job retry strategies in your job definitions to handle these failures automatically. For more information, see Automated job retries.
> Once you're confident in the new environment:
>
> 1. Edit the job queue to remove the old compute environment.
> 2. Wait for any running jobs in the old environment to complete.

3. Delete the old compute environment.

Performing blue/green updates using the AWS Management Console

1. Clone your current compute environment

    a. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
    b. Select your existing compute environment.
    c. Choose **Actions** and then **Clone**.
    d. For **Name**, enter a unique name for your new compute environment.
    e. Choose **Next**.
    f. In the **Instance configuration** section, update the AMI settings:
        i. Expand **Additional configuration**.
        ii. For **EC2 configuration**, specify the new AMI type in **Image type** and AMI ID in the **Image ID override** field.
    g. Choose **Next**.
    h. For **Network configuration** choose **Next**.
    i. Review the other settings which are automatically copied from your existing environment.
    j. Choose **Create compute environment**.
    k. Wait for the new compute environment status to become `VALID`.
2. Change the job queue order

    a. In the navigation pane, choose **Job queues**.
    b. Select the job queue associated with your existing compute environment.
    c. Choose **Edit**.
    d. Under **Connected Compute environment**, add the new compute environment:
        - Add the new compute environment with a higher order number than the existing environment to transition the workload.
        - Once you verify the new environment is working correctly, you can make it the primary environment by giving it a lower order number.
    e. Choose **Update job queue**.

3. Clean up

   a. Monitor job execution in the new environment to ensure everything is working as expected.

   b. Once you're confident in the new environment:

      1. Edit the job queue to remove the old compute environment.
      2. Wait for any running jobs in the old environment to complete.
      3. Delete the old compute environment.

Performing blue/green updates using the AWS CLI

1. To get the configuration using the AWS CLI, use the following command:

   ```
   aws batch describe-compute-environments \
     --compute-environments your-compute-environment-name
   ```

   Save the output for reference when creating the new environment.

2. Create a new compute environment using the configuration from your existing environment, but with the new AMI. Here's an example command structure:

   Replace the example values with your actual configuration from the previous step:

   ```
   cat <<EOF > ./blue-green-compute-environment.json
   {
     "computeEnvironmentName": "your-new-compute-environment-name",
     "type": "MANAGED",
     "state": "ENABLED",
     "computeResources": {
       "instanceRole": "arn:aws:iam::012345678901:instance-profile/
   ecsInstanceRole",
       "type": "EC2",
       "minvCpus": 2,
       "desiredvCpus": 2,
       "maxvCpus": 256,
       "instanceTypes": [
         "optimal"
       ],
       "allocationStrategy": "BEST_FIT_PROGRESSIVE",
       "ec2Configuration": [
   ```

```
   {
    "imageType": "ECS_AL2023",
    "imageIdOverride": "ami-0abcdef1234567890"
   }
  ],
  "subnets": [,
   "subnet-0abcdef1234567890"
  ],
  "securityGroupIds": [
   "sg-0abcdef1234567890"
  ]
 }
}
EOF
```

```
$ aws batch create-compute-environment --cli-input-json file://./blue-green-compute-environment.json
```

3. Wait for the new environment to become available:

```
aws batch describe-compute-environments \
  --compute-environments your-new-compute-environment-name \
  --query 'computeEnvironments[].status'
```

4. Add the new compute environment to your job queue:

```
aws batch update-job-queue \
  --job-queue your-job-queue \
  --compute-environment-order order=1,computeEnvironment=your-existing-environment \
  order=2,computeEnvironment=your-new-compute-environment-name
```

5. Once verified, update again to make the new environment primary:

```
aws batch update-job-queue \
  --job-queue your-job-queue \
  --compute-environment-order order=1,computeEnvironment=your-new-compute-environment-name
```

After all jobs complete in the old environment, disable and then delete it:

```
aws batch update-compute-environment \
    --compute-environment your-existing-environment \
    --state DISABLED
```

```
aws batch delete-compute-environment \
  --compute-environment your-existing-environment
```

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 Update a compute environment in AWS Batch. 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

- Tutorial: Create a compute resource AMI
- Use a GPU workload AMI
- Amazon Linux deprecation
- Amazon EKS Amazon Linux 2 AMI deprecation
- Amazon ECS Amazon Linux 2 AMI 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*.

Tutorial: Create 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 intend 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*.

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 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 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. Then, configure the AMI to run AWS Batch jobs. For more information, see Use a GPU workload AMI.

> **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 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 and Amazon EC2 Instance Store in the *Amazon EC2 User Guide*.
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 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*.
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:

        ```
        sudo systemctl stop ecs
        ```
        - Amazon ECS-optimized Amazon Linux AMI:

        ```
        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.

```
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* 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 “Tutorial: Create a managed compute environment using Amazon EC2 resources”.

   > **Note**
   >
   > The AMI that you choose for a compute environment must match the architecture of the instance types that you intend 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 Create 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 Job definition examples, Create a single-node job definition , and Tutorial: submit a job.

Use 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 `p3`, `p4`, `p5`, `p6`, `g3`, `g3s`, `g4`, `g5`, or `g6` 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 in *Amazon Elastic Container Service Developer Guide*.

The following examples show how to use the GetParameter command.

AWS CLI

```
$ 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.

```
{
    "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\"}",
        "Version": 9,
        "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",
            "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",
            "LastModifiedDate": 1555434128.712,
            "Value": "amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_name"
        }
    ]
}
```

Python

```
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",
            "LastModifiedDate": 1555434128.712,
            "Value": "amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_name"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
os",
            "LastModifiedDate": 1548368308.143,
            "Value": "Amazon Linux 2",
            "Version": 1,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/os"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
schema_version",
            "LastModifiedDate": 1548368307.914,
            "Value": "1",
            "Version": 1,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/schema_version"
        }
    ]
}
```

Python

```
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.

Amazon EKS Amazon Linux 2 AMI deprecation

AWS will end support for Amazon EKS optimized Amazon Linux 2 AMIs, effective 11/26/25. We recommend migrating AWS Batch Amazon EKS compute environments to Amazon Linux 2023 prior to 11/26/25 to maintain optimal performance and security.

While you can continue using Batch-provided Amazon EKS optimized Amazon Linux 2 AMIs on your Amazon EKS compute environments beyond the 11/26/25 end-of-support date, 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 EKS optimized Amazon Linux 2 AMI after end-of-life.

For more information about the Amazon EKS AL2 end-of-life, see Amazon EKS AMI deprecation FAQs in the *Amazon EKS User Guide*.

For help migrating AWS Batch Amazon EKS compute environments from Amazon Linux 2 to Amazon Linux 2023, see How to upgrade from EKS AL2 to EKS AL2023.

Amazon ECS Amazon Linux 2 AMI deprecation

AWS will end support for Amazon Linux 2. Starting in January 2026, AWS Batch will change the default AMI for new Amazon ECS compute environments from Amazon Linux 2 to Amazon Linux 2023. We recommend migrating AWS Batch Amazon ECS compute environments to Amazon Linux 2023, to maintain optimal performance and security.

For more information about the Amazon Linux 2 end-of-life, see Amazon Linux 2 FAQs.

For information about differences between Amazon Linux 2 and Amazon Linux 2023, see Compare Amazon Linux 2023 and Amazon Linux 2 in the *Amazon Linux 2023 User Guide*.

For information about changes in Amazon Linux 2023 for Amazon ECS-optimized AMI, see Migrating from an Amazon Linux 2 to an Amazon Linux 2023 Amazon ECS-optimized AMI in the *Amazon ECS User Guide*.

For help migrating AWS Batch Amazon ECS compute environments from Amazon Linux 2 to Amazon Linux 2023, see How to migrate from ECS AL2 to ECS AL2023.

Use Amazon EC2 launch templates with AWS Batch

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.

```
{
    "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*.

If you use a launch template to create your compute environment, you can move the following existing compute environment parameters to your launch template:

> **Note**
>
> Suppose that any of these parameters (except the Amazon EC2 tags) are specified both in the launch template and in the compute environment configuration. Then, the compute environment parameters take precedence. Amazon EC2 tags are merged between the launch template and the compute environment configuration. If there's a collision on the tag's key, the value in the compute environment configuration takes precedence.

- 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 Update a compute environment in AWS Batch.

Default and override launch templates

You can define a default launch template for the compute environment and an override launch template for specific instance types and families. This can be useful to you so that the default template is used for the majority of instance types in the compute environments.

The substitution variables `$Default` and `$Latest` can be used instead of naming a specific version. If you do not provide an override launch template, the default launch template is automatically applied.

If you use either the `$Default` or `$Latest` variable, AWS Batch will apply the current information at the time that the compute environment is created. If the default or latest version changes in the future, you must update the information through UpdateComputeEnvironment or through the AWS Management Console - AWS Batch.

To provide additional flexibility, you can define override launch templates are applied to specific compute instance types or families.

> **Note**
>
> You can specify up to ten (10) override launch templates per compute environment.

Use the `targetInstanceTypes` parameter to select the instance type or family that should use this override launch template. The instance type or family must be first identified by the `instanceTypes` parameter.

If you define launch template overrides and decide to remove them later, you can pass an empty array to unset the `overrides` parameter in the `UpdateComputeEnvironment` API operation. You can also choose to not include the `overrides` parameter when submitting the `UpdateComputeEnvironment` API operation. For more information see, `LaunchTemplateSpecification.overrides`

For more information, see `LaunchTemplateSpecificationOverride.targetInstanceTypes` in the AWS Batch API Reference guide.

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.

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.

```
{
    "LaunchTemplateName": "base64-user-data",
    "LaunchTemplateData": {
        "UserData":
 "ewogICAgIkxhdW5jaFRlbXBsYXRlTmFtZSI6ICJpbmNyZWFzZS1jb250YWluZXItdm9sdW..."
    }
}
```

Topics

- Reference: Amazon EC2 launch template examples

Reference: Amazon EC2 launch template examples

The following are example MIME multi-part files that you can use to create your own 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.

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

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo ECS_IMAGE_CLEANUP_INTERVAL=60m >> /etc/ecs/ecs.config
echo ECS_IMAGE_MINIMUM_CLEANUP_AGE=60m >> /etc/ecs/ecs.config

--==MYBOUNDARY==--
```

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*.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- file_system_id_01=fs-0abcdef1234567890
- region=us-east-2
- fsx_directory=/scratch
- amazon-linux-extras install -y lustre2.10
- mkdir -p ${fsx_directory}
- mount -t lustre ${file_system_id_01}.fsx.${region}.amazonaws.com@tcp:fsx
 ${fsx_directory}

--==MYBOUNDARY==--
```

In the volumes and mountPoints members of the container properties the mount points must be mapped into the container.

```
{
    "volumes": [
        {
            "host": {
                "sourcePath": "/scratch"
            },
            "name": "Scratch"
        }
    ],
    "mountPoints": [
        {
            "containerPath": "/scratch",
            "sourceVolume": "Scratch"
        }
    ],
}
```

Instance Metadata Service (IMDS) configuration

The Instance Metadata Service (IMDS) provides metadata about your EC2 instances to applications running on those instances. Use IMDSv2 for all new workloads and migrate existing workloads from IMDSv1 to IMDSv2 for improved security. For more information about IMDS and configuring IMDS, see Use instance metadata to manage your EC2 instance and Configure instance metadata options for new instances in the *Amazon EC2 User Guide*.

Configuration scenarios

Choose the appropriate configuration method based on your compute environment setup:

Default AMI with no launch template

When you use the default AWS Batch AMI and don't specify a launch template, choose one of these options:

1. **Use Amazon Linux 2023 default AMI –** Amazon Linux 2023 requires IMDSv2 by default. When you create your compute environment, select **Amazon Linux 2023** as the image type.
2. **Set account-level IMDSv2 configuration –** Configure your AWS account to require IMDSv2 for all new instances. This setting affects all new instances that you launch in the account. For instructions, see Set IMDSv2 as the default for the account in the *Amazon EC2 User Guide*.

> **Note**
>
> Account-level IMDS configuration can be overridden by launch template or AMI configuration. Launch template settings take precedence over account-level settings.

Custom AMI with no launch template

When you use a custom AMI without a launch template, choose one of these options:

1. **Use Amazon Linux 2023 as base –** Build your custom AMI using Amazon Linux 2023 as the base image. For information about creating custom AMIs for Batch, see Tutorial: Create a compute resource AMI.
2. **Configure IMDSv2 in your custom AMI –** When you create your custom AMI, configure it to require IMDSv2. For instructions, see Configure instance metadata options for custom AMI in the *Amazon EC2 User Guide*.

3. **Set account-level IMDSv2 configuration** – Configure your AWS account to require IMDSv2 for all new instances. This setting affects all new instances that you launch in the account. For instructions, see Set IMDSv2 as the default for the account in the *Amazon EC2 User Guide*.

> **Note**
>
> Account-level IMDS configuration can be overridden by launch template or AMI configuration. Launch template settings take precedence over account-level settings.

Using launch templates

When you use launch templates in your compute environment, add metadata options to your launch template to require IMDSv2. For more information about using launch templates with Batch, see Use Amazon EC2 launch templates with AWS Batch.

```
{
    "LaunchTemplateName": "batch-imdsv2-template",
    "VersionDescription": "IMDSv2 only template for Batch",
    "LaunchTemplateData": {
        "MetadataOptions": {
            "HttpTokens": "required"
        }
    }
}
```

Create the launch template using the AWS CLI:

```
aws ec2 create-launch-template --cli-input-json file://imds-template.json
```

EC2 configurations

AWS Batch uses Amazon ECS optimized AMIs for EC2 and EC2 Spot compute environments. The default is Amazon Linux 2 (`ECS_AL2`). Starting in January 2026, the default will change to AL2023 (`ECS_AL2023`).

AWS will end support for Amazon Linux 2. We recommend migrating AWS Batch Amazon ECS compute environments to Amazon Linux 2023 to maintain optimal performance and security. For more information, see Amazon ECS Amazon Linux 2 AMI deprecation.

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.

For help migrating AWS Batch from the Amazon Linux AMI to Amazon Linux 2023, see How to migrate from ECS AL2 to ECS AL2023

Topics

- How to migrate from ECS AL2 to ECS AL2023

How to migrate from ECS AL2 to ECS AL2023

AL2023 is a Linux-based operating system designed to provide a secure, stable, and high-performance environment for your cloud applications. For more information about the differences between AL2 and AL2023 see Compare Amazon Linux 2023 and Amazon Linux 2 in the *Amazon Linux 2023 User Guide*.

Starting in January 2026, AWS Batch will change the default AMI for new Amazon ECS compute environments from Amazon Linux 2 to Amazon Linux 2023 because AWS will be ending support for Amazon Linux 2. The default AMI is used when you don't specify a value for the imageType.Ec2Configuration field when creating a new compute environment. We recommend migrating AWS Batch Amazon ECS compute environments to Amazon Linux 2023 to maintain optimal performance and security.

Depending on how your compute environment is configured you can use one of the following upgrade paths from AL2 to AL2023.

Upgrade using Ec2Configuration.ImageType

- If you are not using a launch template or launch template overrides then change Ec2Configuration.ImageType to `ECS_AL2023` (or `ECS_AL2023_NVIDIA` when using GPU instances) and then run UpdateComputeEnvironment.
- If you specify an Ec2Configuration.ImageIdOverride then Ec2Configuration.ImageType must match the AMI type specified in Ec2Configuration.ImageIdOverride.

  If you mismatch `ImageIdOverride` and `ImageType` then the compute environment may not function properly.

Upgrade using launch templates

- If you use a launch template that specifies an AMI based on `ECS_AL2023`, ensure your launch template is compatible with Amazon Linux 2023. For information about changes in Amazon Linux 2023 for Amazon ECS-optimized AMI, see Migrating from an Amazon Linux 2 to an Amazon Linux 2023 Amazon ECS-optimized AMI in the *Amazon ECS User Guide*.
- For AL2023 AMIs, verify that any custom user data or initialization scripts are compatible with the AL2023 environment and package management system.

Upgrade using CloudFormation

- If you use CloudFormation to manage your compute environments, update your template to change the `ImageType` property in the `Ec2Configuration` from `ECS_AL2` to `ECS_AL2023` (or `ECS_AL2023_NVIDIA` when using GPU instances):

  ```
  ComputeEnvironment:
    Type: AWS::Batch::ComputeEnvironment
    Properties:
      ComputeResources:
        Ec2Configuration:
          - ImageType: ECS_AL2023
  ```

  Then update your CloudFormation stack to apply the changes.
- If your CloudFormation template specifies a custom AMI using `ImageIdOverride`, ensure the AMI ID corresponds to an AL2023-based AMI and matches the `ImageType` setting.

Migration considerations

When migrating from Amazon Linux 2 to Amazon Linux 2023, consider the following:

- **Package management** – Amazon Linux 2023 uses `dnf` instead of `yum` for package management.
- **System services** – Some system services and their configurations may differ between AL2 and AL2023.
- **Container runtime** – Both AL2 and AL2023 support Docker, but AL2023 may have different default configurations.
- **Security** – AL2023 includes enhanced security features and may require updates to security-related configurations.

- **Instance Metadata Service Version 2 (IMDSv2)** – IMDSv2 is a session-oriented service that requires token-based authentication to access EC2 instance metadata, providing enhanced security. For more information about IMDS see and How Instance Metadata Service Version 2 works in the *Amazon EC2 User Guide*.

For a comprehensive list of changes and migration considerations, see Migrating from an Amazon Linux 2 to an Amazon Linux 2023 Amazon ECS-optimized AMI in the *Amazon ECS User Guide*.

Instance type allocation strategies for AWS Batch

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, 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 Update a compute environment in AWS Batch.

> **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.

> **Note**
>
> For multi-node parallel jobs AWS Batch chooses the optimal instance type available. If the instance type becomes unavailable due to insufficient capacity, other instance types within the family are not launched.

`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.

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 Reserve 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.

Topics

- Reserve system memory
- Tutorial: View compute resource memory
- Memory and vCPU considerations for AWS Batch on Amazon EKS

Reserve 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. We recommend reserving a 5% memory buffer for Amazon ECS container agent and other critical system processes.

Tutorial: View compute resource memory

You can view how much memory a compute resource registers with in the Amazon ECS console or with the DescribeContainerInstances 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

1. Open the console at https://console.aws.amazon.com/ecs/v2.
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 **Total capacity** 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.

```
{
   "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.

```
{
   "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.

```
{
   "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.

```
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.

Example: Node CPU reservation

The CPU reservation value is calculated in millicores using the total number of vCPUs that are available to the instance.

vCPU number	Percentage reserved
1	6%
2	1%

vCPU number	Percentage reserved
3-4	0.5%
4 and above	0.25%

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.

Example: Node memory reservation

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 * 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: (*`instance_capacity_in_MiB`*) - (11 * (*`maximum_number_of_pods`*)) - 255 - (*`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 * 29) -255 -100) = 7074 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.

Fargate compute environments

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 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.

Topics

- 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 Fargate service quotas in the *Amazon Elastic Container Service Developer Guide*.

Job definitions on Fargate

AWS Batch jobs on AWS 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 Use 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 IAM execution role.

Job queues on Fargate

AWS Batch job queues on AWS 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 AWS 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.

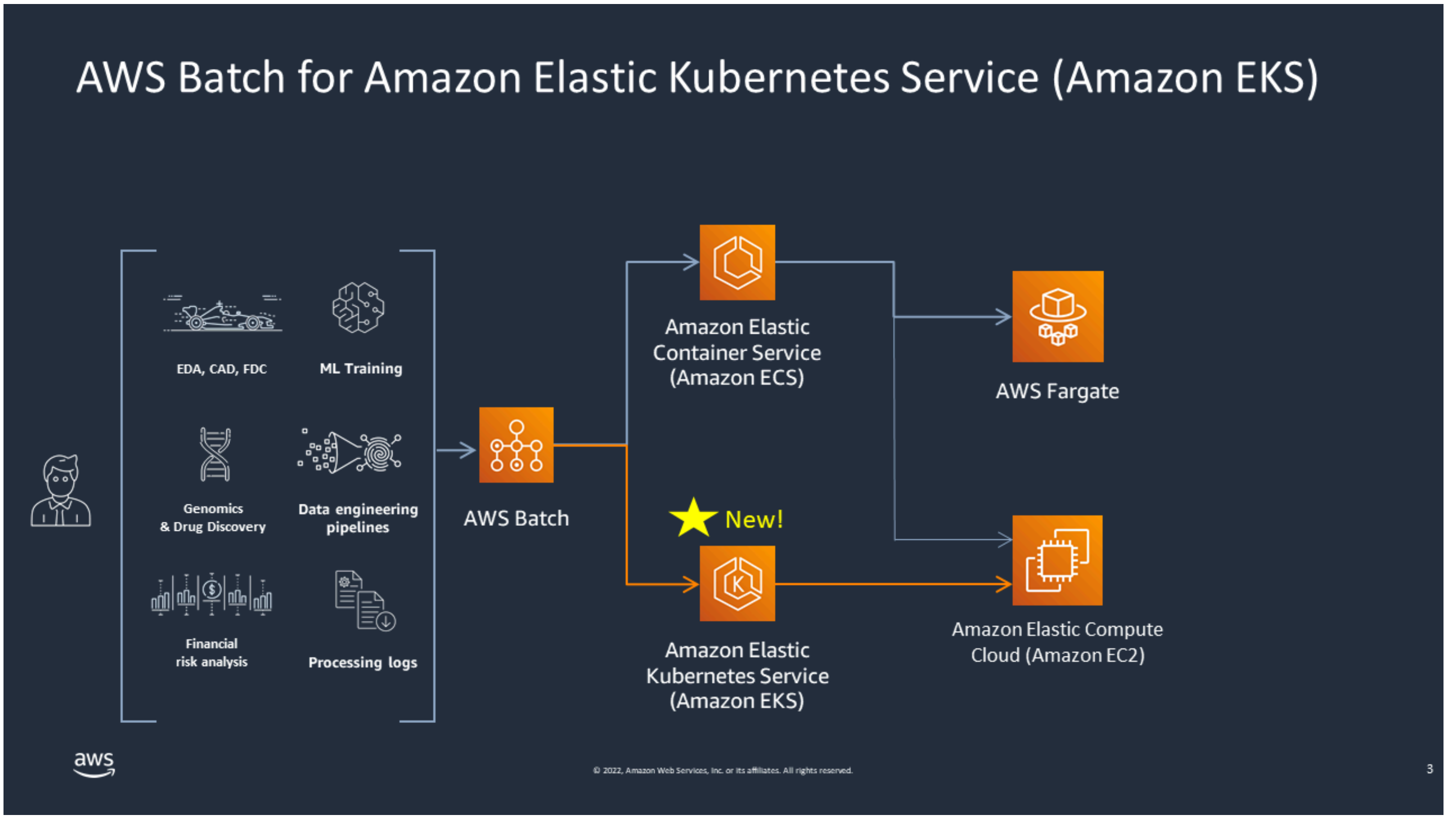
`type`

This must be either `FARGATE` or `FARGATE_SPOT`.

```
"type": "FARGATE_SPOT"
```

Amazon EKS compute environments

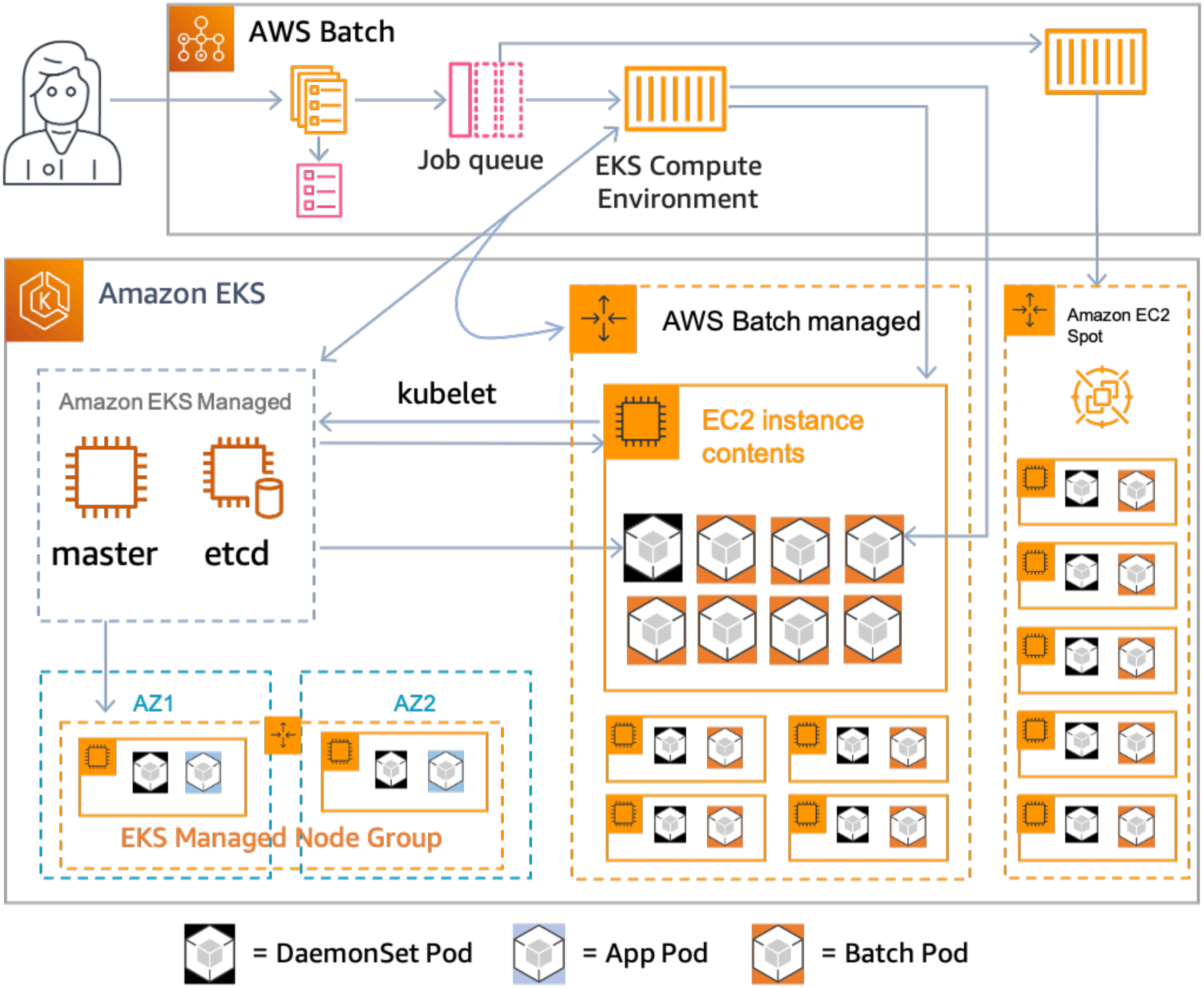
Getting started with AWS Batch on Amazon EKS provides a short guide to creating EKS compute environments. This section provides more details on Amazon EKS compute environments.



AWS Batch for Amazon Elastic Kubernetes Service (Amazon EKS)

EDA, CAD, FDC
ML Training
Genomics & Drug Discovery
Data engineering pipelines
Financial risk analysis
Processing logs
AWS Batch
Amazon Elastic Container Service (Amazon ECS)
New!
Amazon Elastic Kubernetes Service (Amazon EKS)
AWS Fargate
Amazon Elastic Compute Cloud (Amazon EC2)
aws

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.

Amazon EKS

Topics

- Amazon EKS default AMI
- Mixed AMI environments
- Supported Kubernetes versions
- Update the Kubernetes version of the compute environment
- Shared responsibility of the Kubernetes nodes
- Run a DaemonSet on AWS Batch managed nodes
- Customize Amazon EKS launch templates
- How to upgrade from EKS AL2 to EKS AL2023

Amazon EKS default AMI

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*.

> **Important**
>
> Amazon Linux 2023 AMIs are the default on AWS Batch for Amazon EKS.
> AWS will end support for Amazon EKS AL2-optimized and AL2-accelerated AMIs, starting 11/26/25. You can continue using AWS Batch-provided Amazon EKS optimized Amazon Linux 2 AMIs on your Amazon EKS compute environments beyond the 11/26/25 end-of-support date, however these compute environments will no longer receive any new software updates, security patches, or bug fixes from AWS. For more information on upgrading from AL2 to AL2023, see How to upgrade from EKS AL2 to EKS AL2023 in the *AWS Batch User Guide*.

Run the following command to see which AMI type AWS Batch 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 1.32 AMI,
 depending on instance types
    $ aws batch describe-compute-environments --compute-environments My-Eks-CE1 \
        | jq '.computeEnvironments[].computeResources.ec2Configuration'
    [
      {
        "imageType": "EKS_AL2",
        "imageKubernetesVersion": "1.32"
      }
    ]
```

This following example is a GPU instance type.

```
# GPU CE example: indicates Batch has choosen the AL2 x86 EKS Accelerated 1.32 AMI
    $ aws batch describe-compute-environments --compute-environments My-Eks-GPU-CE \
        | jq '.computeEnvironments[].computeResources.ec2Configuration'
    [
      {
        "imageType": "EKS_AL2_NVIDIA",
        "imageKubernetesVersion": "1.32"
      }
    ]
```

Mixed AMI environments

You can use launch template overrides to create compute environments with both Amazon Linux 2 (AL2) and Amazon Linux 2023 (AL2023) AMIs. This is useful for using different AMIs for different architectures or during migration periods when transitioning from AL2 to AL2023.

> **Note**
>
> AWS will end support for Amazon EKS AL2-optimized and AL2-accelerated AMIs, starting 11/26/25. While you can continue using AWS Batch-provided Amazon EKS optimized Amazon Linux 2 AMIs on your Amazon EKS compute environments beyond the 11/26/25 end-of-support date, these compute environments will no longer receive any new software updates, security patches, or bug fixes from AWS. Mixed AMI environments can be useful during the transition period, allowing you to gradually migrate workloads to AL2023 while maintaining compatibility with existing AL2-based workloads.

Example configuration using both AMI types:

```
{
  "computeResources": {
    "launchTemplate": {
      "launchTemplateId": "TemplateId",
      "version": "1",
      "userdataType": "EKS_BOOTSTRAP_SH",
      "overrides": [
        {
          "instanceType": "c5.large",
          "imageId": "ami-al2-custom",
          "userdataType": "EKS_BOOTSTRAP_SH"
        },
        {
          "instanceType": "c6a.large",
          "imageId": "ami-al2023-custom",
          "userdataType": "EKS_NODEADM"
        }
      ]
    },
    "instanceTypes": ["c5.large", "c6a.large"]
  }
}
```

Supported Kubernetes versions

AWS Batch on Amazon EKS currently supports the following Kubernetes versions:

- `1.34`
- `1.33`
- `1.32`
- `1.31`
- `1.30`
- `1.29`

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
```

Update the Kubernetes version of the 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.

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources \
      'ec2Configuration=[{imageType=EKS_AL2,imageKubernetesVersion=1.32}]'
```

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 Run 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.

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources 'updateToLatestImageVersion=true'
```

AWS Batch doesn't automatically upgrade the Kubernetes version. Run the following command to update the Kubernetes version of your computer environment to *1.32*.

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources \
      'ec2Configuration=[{imageType=EKS_AL2,imageKubernetesVersion=1.32}]'
```

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 Update a compute environment in AWS Batch and the infrastructure update policy (UpdatePolicy) set in the UpdateComputeEnvironment API operation.

Run 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`.

```
tolerations:
  - key: "batch.amazonaws.com/batch-node"
    operator: "Exists"
```

Another way to do this is with the following `tolerations`.

```
tolerations:
  - key: "batch.amazonaws.com/batch-node"
    operator: "Exists"
    effect: "NoSchedule"
  - key: "batch.amazonaws.com/batch-node"
    operator: "Exists"
    effect: "NoExecute"
```

Customize Amazon EKS launch templates

AWS Batch on Amazon EKS supports launch templates. There are constraints on what your launch template can do.

> **Important**
>
> - For EKS AL2 AMIs, 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.

- For EKS AL2023, AWS Batch utilizes the NodeConfigSpec from EKS to make instances join the EKS cluster. AWS Batch populates ClusterDetails in NodeConfigSpec for the EKS cluster and you don't need to specify them.

Note

We recommend that you do not set any of the follow NodeConfigSpec settings in the launch template as AWS Batch will override your values. For more information, see Shared responsibility of the Kubernetes nodes.

- `Taints`
- `Cluster Name`
- `apiServerEndpoint`
- `certificatAuthority`
- `CIDR`
- Do not create a labels with the prefix `batch.amazonaws.com/`

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

- Add kubelet extra arguments
- Configure the container runtime
- Mount an Amazon EFS volume
- IPv6 support

Add `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 for EKS AL2 AMIs, *`--node-labels mylabel=helloworld`* is added to the `kubelet` command line.

```
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

      echo AWS_BATCH_KUBELET_EXTRA_ARGS=\"--node-labels mylabel=helloworld\" >> /etc/
aws-batch/batch.config

      --==MYBOUNDARY==--
```

For EKS AL2023 AMIs the file format is YAML. For the list of supported parameters, see NodeConfigSpec in the *Kubernetes documentation*. In the following example for EKS AL2023 AMIs, *`--node-labels mylabel=helloworld`* is added to the `kubelet` command line.

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

--==MYBOUNDARY==
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  kubelet:
    flags:
    - --node-labels=mylabel=helloworld

--==MYBOUNDARY==--
```

Configure 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*.

If you are using an optimized `EKS_AL2023` or `EKS_AL2023_NVIDIA` AMI you do not need to specify the container runtime as only **containerd** is supported.

> **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*.

```
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

      echo CONTAINER_RUNTIME=containerd >> /etc/aws-batch/batch.config

      --==MYBOUNDARY==--
```

Mount 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/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 _netdev,noresvport,tls,iam 0 0"
 >> /etc/fstab
- mount -t efs -o tls ${file_system_id_01}:/ ${efs_directory}

--==MYBOUNDARY==--
```

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.

```
{
    "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"
                        }
                    ]
                }
            ],
            "volumes": [
                {
                    "name": "efs-volume",
                    "hostPath": {
                        "path": "/mnt/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*.

How to upgrade from EKS AL2 to EKS AL2023

The Amazon EKS optimized AMIs are available in two families based on Amazon Linux 2 (AL2) and Amazon Linux 2023 (AL2023). AL2023 is a Linux-based operating system designed to provide a secure, stable, and high-performance environment for your cloud applications. For more information about the differences between AL2 and AL2023 see Upgrade from Amazon Linux 2 to Amazon Linux 2023 in the *Amazon EKS User Guide*.

> ⚠ **Important**
>
> AWS will end support for Amazon EKS AL2-optimized and AL2-accelerated AMIs, starting 11/26/25. We recommend migrating AWS Batch Amazon EKS compute environments to Amazon Linux 2023 prior to 11/26/25 to maintain optimal performance and security. While you can continue using AWS Batch-provided Amazon EKS optimized Amazon Linux 2 AMIs on your Amazon EKS compute environments beyond the 11/26/25 end-of-support date, 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 EKS optimized Amazon Linux 2 AMI after end-of-life.

Depending on how your compute environment is configured you can use one of the following upgrade paths from AL2 to AL2023.

Upgrade using Ec2Configuration.ImageType

- If you are not using a launch template or launch template overrides then change Ec2Configuration.ImageType to `EKS_AL2023` or `EKS_AL2023_NVIDIA` and then run UpdateComputeEnvironment.
- If you specify an Ec2Configuration.ImageIdOverride then Ec2Configuration.ImageType must match the AMI type specified in Ec2Configuration.ImageIdOverride.

  If you mismatch `ImageIdOverride` and `ImageType` then the node won't join the cluster.

Upgrade using launch templates

- If you have any `kubelet` extra arguments defined in a launch template or launch template override, they need to be updated to the new `kubelet` extra arguments format.

  If you mismatch the `kubelet` extra arguments format then the extra arguments aren't applied.
- For AL2023 AMIs, **containerd** is the only supported container runtime. You do not need to specify container runtime for `EKS_AL2023` in the launch template.

  You can't specify a customized container runtime with `EKS_AL2023`.
- If you use a launch template or launch template override that specifies an AMI based on `EKS_AL2023` then you need to set userdataType to `EKS_NODEADM`.

  If you mismatch the `userdataType` and AMI then the node won't join the EKS cluster.

Service environments for AWS Batch

Service environments enable AWS Batch to integrate with SageMaker AI. A service environment contains the SageMaker AI specific configuration parameters required for AWS Batch to submit and manage SageMaker Training jobs while providing AWS Batch's queuing, scheduling, and priority management capabilities.

With service environments, data scientists and ML engineers can submit SageMaker Training jobs with priorities to service job queues. This integration eliminates the need for manual coordination of ML workloads, prevents accidental overspending, and improves resource utilization across your organization's machine learning workflows.

Topics

- What are service environments in AWS Batch
- Service environment states and lifecycle in AWS Batch
- Create a service environment in AWS Batch
- Update a service environment in AWS Batch
- Delete a service environment in AWS Batch

What are service environments in AWS Batch

A service environment is a AWS Batch resource that contains the configuration parameters required to integrate AWS Batch with SageMaker AI. Service environments enable AWS Batch to submit and manage SageMaker Training jobs while providing AWS Batch's queuing, scheduling, and priority management capabilities.

Service environments address common challenges that data science teams face when managing machine learning workloads. Organizations often limit the number of instances available for training models to prevent accidental overspending, meet budget constraints, save costs with reserved instances, or use specific instance types for workloads. However, data scientists may want to run more workloads concurrently than is possible with their allocated instances, requiring manual coordination to decide which workloads run when.

This coordination challenge impacts organizations of all sizes, from teams with just a few data scientists to large-scale operations. As organizations grow, the complexity increases, requiring more time to manage workload coordination and often necessitating infrastructure administrator involvement. These manual efforts waste time and reduce instance efficiency, resulting in real costs for customers.

With service environments, data scientists and ML engineers can submit SageMaker Training jobs with priorities to configurable queues, ensuring workloads run automatically without intervention as soon as resources are available. This integration leverages AWS Batch's extensive queuing and scheduling capabilities, enabling customers to customize their queuing and scheduling policies to match their organization's goals.

How service environments work with other AWS Batch components

Service environments integrate with other AWS Batch components to enable SageMaker Training job queuing:

- **Job queues** - Service environments are associated with job queues to enable the queue to process service jobs for SageMaker Training job
- **Service jobs** - When you submit a service job to a queue associated with a service environment, AWS Batch uses the environment's configuration to submit the corresponding SageMaker Training job
- **Scheduling policies** - Service environments work with AWS Batch scheduling policies to prioritize and manage the execution order of SageMaker Training jobs

This integration allows you to leverage AWS Batch's mature queuing and scheduling capabilities while maintaining the full functionality and flexibility of SageMaker Training jobs.

Best practices for service environments

Service environments provide capabilities for managing SageMaker Training jobs at scale. Following these best practices helps you optimize cost, performance, and operational efficiency while avoiding common configuration issues that can impact your machine learning workflows.

When planning service environment capacity, consider the specific quotas and limits that apply to SageMaker Training job queuing. Each service environment has a maximum capacity limit expressed in number of instances, which directly controls how many SageMaker Training jobs can run concurrently. Understanding these limits helps prevent resource contention and ensures predictable job execution times.

Optimal service environment performance depends on understanding the unique characteristics of SageMaker Training job scheduling. Unlike traditional containerized jobs, service jobs transition through a `SCHEDULED` state while SageMaker AI acquires and provisions the necessary training instances. This means job start times can vary significantly based on instance availability and regional capacity.

> ⚠ **Important**
>
> Service environments have specific quotas that can impact your ability to scale SageMaker Training workloads. You can create up to 50 service environments per account, with each job queue supporting only one associated service environment. Additionally, the Service Request Payload for individual jobs is limited to 10 KiB, and the `SubmitServiceJob` API is limited to 5 transactions per second per account. Understanding these limits during capacity planning prevents unexpected scaling constraints.

Effective monitoring of service environments requires attention to both AWS Batch and SageMaker AI service metrics. Job state transitions provide valuable insights into system performance, particularly the time spent in `SCHEDULED` state, which indicates capacity availability patterns. Service environments maintain their own lifecycle states similar to compute environments, transitioning through `CREATING`, `VALID`, `INVALID`, and `DELETING` states that should be monitored for operational health. Organizations with mature monitoring practices typically track queue depth, job completion rates, and instance utilization patterns to optimize their service environment configurations over time.

Service environment states and lifecycle in AWS Batch

Service environments maintain lifecycle states that indicate their current operational status and readiness to process SageMaker Training jobs. Understanding these states helps you monitor service environment health, troubleshoot configuration issues, and ensure reliable job processing. The state management system follows established patterns from compute environments while accommodating the unique requirements of SageMaker Training job integration.

Service environment states are managed automatically by AWS Batch based on configuration validation, resource availability, and operational health checks. Unlike compute environments that manage physical infrastructure, service environments focus on configuration validation and integration readiness with SageMaker AI services. The state transitions provide visibility into whether your service environment can successfully submit and manage SageMaker Training jobs.

Service environment state definitions

Service environments can be in one of four possible states that indicate their current operational status and readiness to process SageMaker Training jobs. Each state represents a specific phase in the service environment lifecycle, from initial creation through operational readiness to eventual deletion. The following table describes each state and its meaning:

State	Description
`CREATING`	The initial state when you create a service environment. During this state, AWS Batch validates the configuration parameters and establishes integration with SageMaker AI services. The service environment cannot process jobs, and any job queues associated with it will not accept service job submissions. The creation process typically completes within a few seconds for properly configured service environments.
`VALID`	The operational state indicating that the service environment has passed all configuration validation checks and is ready to process SageMaker Training jobs. This state indicates that the service environment configuration is correct, all required permissions are in place, and AWS Batch can successfully submit jobs to SageMaker AI on your behalf. Service environments spend most of their operational lifecycle in this state.
`INVALID`	A state indicating that the service environment has encountered a configuration or permissions issue that prevents it from processing SageMaker Training jobs. Job queues associated with invalid service environments cannot process new service job

State	Description
	submissions until the underlying issues are resolved.
`DELETING`	The state that occurs when you request deletion of a service environment. During this state, AWS Batch ensures that no active SageMaker Training jobs are associated with the environment and performs necessary cleanup operations. Service environments in this state cannot process new job submissions, and the deletion process completes once all associated resources are properly cleaned up.

Service environment state transitions

Service environment state transitions occur automatically based on configuration changes, validation results, and operational health monitoring. The AWS Batch service continuously monitors service environment health and updates states accordingly. Understanding these transitions helps you anticipate when configuration changes will take effect and how to resolve issues that cause invalid states.

After successful creation and validation, service environments transition from `CREATING` to `VALID`. This transition confirms that all configuration parameters are correct, required IAM permissions are properly configured, and the service environment can successfully integrate with SageMaker AI services. Once in the `VALID` state, associated job queues can begin processing service job submissions.

Service environments transition from `VALID` to `INVALID` when configuration validation fails or when dependencies become unavailable. This can occur due to IAM role modifications, capacity limit changes that violate quotas, or external resource modifications that affect the service environment's ability to function. The status reason field provides specific details about what caused the invalid state.

Service environments can transition back to `VALID` from `INVALID` once the underlying issues are resolved. This might involve updating IAM permissions, correcting capacity configurations, or restoring access to required AWS resources. The transition typically occurs automatically once AWS Batch detects that the configuration issues have been addressed.

Create a service environment in AWS Batch

Before you can run SageMaker Training jobs in AWS Batch, you need to create a service environment. You can create a service environment that contains the configuration parameters required for AWS Batch to integrate with SageMaker AI services and submit SageMaker Training jobs on your behalf.

Prerequisites

Before creating a service environment, ensure you have:

- **IAM permissions** – Permissions to create and manage service environments. For more information, see AWS Batch IAM policies, roles, and permissions.

Create a service environment (AWS Console)

Use the AWS Batch console to create a service environment through the web interface.

To create a service environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Environments**.
3. Choose **Create environment**, select **Service environment**.
4. For **Service environment configuration** choose SageMaker AI.
5. For **Name**, enter a unique name for your service environment. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).
6. For **Max number of instances** enter the maximum number of concurrent training instances
7. (Optional) Add tags by choosing **Add tag** and entering key-value pairs.
8. Choose **Next**.
9. Review the details of the new service environment and choose **Create service environment**.

Create a service environment (AWS CLI)

Use the `create-service-environment` command to create a service environment with the AWS CLI.

To create a service environment

1. Create a service environment with the basic required parameters:

```
aws batch create-service-environment \
    --service-environment-name my-sagemaker-service-env \
    --service-environment-type SAGEMAKER_TRAINING \
    --capacity-limits capacityUnit=NUM_INSTANCES,maxCapacity=10
```

2. (Optional) Create a service environment with tags:

```
aws batch create-service-environment \
    --service-environment-name my-sagemaker-service-env \
    --service-environment-type SAGEMAKER_TRAINING \
    --capacity-limits capacityUnit=NUM_INSTANCES,maxCapacity=10 \
    --tags team=data-science,project=ml-training
```

3. Verify the service environment was created successfully:

```
aws batch describe-service-environments \
    --service-environment my-sagemaker-service-env
```

The service environment appears in the Environments list with a `CREATING` state. When creation completes successfully, the state changes to `VALID` and the service environment is ready to have a service job queue added to it so the service environment can start processing jobs.

Update a service environment in AWS Batch

You can update a service environment to modify its capacity limits, change its operational state, or update resource tags. Service environment updates allow you to adjust capacity as your SageMaker Training workload requirements change or modify operational settings without recreating the environment. Before updating a service environment, understand which parameters can be modified and the impact of changes on running jobs.

You can change the Capacity limits, State, or Tags of a service environment.

Update a service environment (AWS Console)

Use the AWS Batch console to update a service environment through the web interface.

To update a service environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Environments**.
3. Choose the **Service environment** tab.
4. Choose the service environment to update.
5. Choose **Actions**, then choose either:
    - **State** - Choose **Enable** or **Disable** to change the state.
    - **Capacity limit** - Modify the **Max number of instances**
6. Choose **Save changes** to apply the changes.

The service environment updates immediately. Check the environment details to confirm the changes were applied successfully. If you disabled the service environment, associated job queues will stop processing new service job submissions until you re-enable it.

Update a service environment (AWS CLI)

Use the `update-service-environment` command to modify a service environment with the AWS CLI.

To update service environment capacity limits

1. Update the capacity limit for a service environment:

```
aws batch update-service-environment \
    --service-environment my-sagemaker-service-env \
    --capacity-limits capacityUnit=NUM_INSTANCES,maxCapacity=20
```

2. Verify the update was applied successfully:

```
aws batch describe-service-environments \
    --service-environments my-sagemaker-service-env
```

To update service environment state

1. Disable a service environment to stop processing new jobs:

```
aws batch update-service-environment \
    --service-environment my-sagemaker-service-env \
    --state DISABLED
```

2. Re-enable a service environment to resume processing:

```
aws batch update-service-environment \
    --service-environment my-sagemaker-service-env \
    --state ENABLED
```

Service environment updates take effect immediately. Monitor the service environment state to ensure updates complete successfully before submitting new jobs.

Delete a service environment in AWS Batch

You can delete a service environment when it's no longer needed for your SageMaker Training jobs. Deleting a service environment removes the configuration and prevents further job submissions. Before deleting a service environment, ensure that no active SageMaker Training jobs depend on it and that no job queues are associated with the service environment.

Important

Service environment deletion is irreversible. Once deleted, you cannot recover the service environment or its configuration. If you need similar functionality in the future, you must create a new service environment with the required settings. Consider disabling the service environment instead of deletion if you may need to reactivate it later.

Note

Deleting all service environments in your account does not automatically remove the service-linked role created for AWS Batch and SageMaker AI integration. The service-linked role remains available for future service environment creation. If you want to remove the service-linked role, you must delete it separately using IAM after ensuring no service environments exist in your account.

Deletion prerequisites

Before you can delete a service environment you must disassociate any service job queue and then disable the service environment.

Before deleting a service environment:

- **Check active jobs** - Ensure no SageMaker Training jobs are currently running through the service environment.
- **Review job queues** - Identify job queues associated with the service environment and either associate the job queue with a different service environment or disable and delete the job queue.

Job queue management: Job queues that were associated with a deleted service environment can still exist but cannot process service jobs. You should either delete unused job queues or associate them with a different service environment before deleting the original service environment.

Delete a service environment (AWS Console)

Use the AWS Batch console to delete a service environment through the web interface.

To delete a service environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Environments**.
3. Choose the **Service environment** tab and then choose a service environment.
4. If the service environment is enabled, choose **Actions** and then **Disable**.
5. Once the service environment is disabled, choose **Actions** and then **Delete**.
6. In the confirmation dialog, choose **Confirm**.

The service environment shows a `DELETING` state while deletion occurs. Once deletion completes, the service environment disappears from the Environments list.

Delete a service environment (AWS CLI)

Use the `delete-service-environment` command to remove a service environment with the AWS CLI.

To delete a service environment

1. Check for associated job queues with the service environment:

   ```
   aws batch describe-job-queues
   ```

   If there are any job queues associated with the service environment you can either disassociate the job queue from the service environment and associate it with a different service environment, or delete the job queue.

2. Disable the service environment:

   ```
   aws batch update-service-environment \
       --service-environment my-sagemaker-service-env \
       --state DISABLED
   ```

3. Delete the service environment:

   ```
   aws batch delete-service-environment \
       --service-environment my-sagemaker-service-env
   ```

4. Monitor the deletion process:

   ```
   aws batch describe-service-environments \
       --service-environment my-sagemaker-service-env
   ```

The service environment transitions to `DELETING` state during the deletion process. Once deletion completes, the service environment is no longer listed in describe operations. Associated job queues remain but cannot process service jobs until associated with a different service environment.

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

- Create a job queue
- View job queue status
- Delete a job queue in AWS Batch
- Fair-share scheduling policies
- Resource-aware scheduling

Create 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.

Topics

- Create an Amazon EC2 job queue
- Create a Fargate job queue
- Create an Amazon EKS job queue
- Create a SageMaker Training job queue in AWS Batch
- Job queue template

Create an Amazon EC2 job queue

Complete the following steps to create a job queue for Amazon Elastic Compute Cloud (Amazon EC2).

To create an Amazon EC2 job queue

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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.

Create a Fargate job queue

Complete the following steps to create a job queue for AWS Fargate.

To create a Fargate job queue

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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.

> **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.

Create an Amazon EKS job queue

Complete the following steps to create a job queue for Amazon Elastic Kubernetes Service (Amazon EKS).

To create an Amazon EKS job queue

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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.

Create a SageMaker Training job queue in AWS Batch

SageMaker Training job queues integrate directly with the SageMaker AI service to provide serverless job scheduling without requiring you to manage underlying compute infrastructure.

Prerequisites

Before creating a SageMaker Training job queue, ensure you have:

- **Service environment** – A service environment that defines capacity limits. For more information, see [Create a service environment in AWS Batch](#).
- **IAM permissions** – Permissions to create and manage AWS Batch job queues and service environments. For more information, see [AWS Batch IAM policies, roles, and permissions](#).

Create a SageMaker Training job queue (AWS Batch console)

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.

2. In the navigation pane, choose **Job queues** and the **Create**.
3. For **Orchestration type**, choose **SageMaker Training**.
4. For **Job queue configuration**:
   a. For **Name**, enter the name of the Job queue.
   b. for **Priority**, enter a value between 0 and 1000. A Job queue with a higher priority is given preference for service environments.
   c. (Optional) For **Scheduling policy Amazon Resource Name (ARN)**, choose an existing scheduling policy.
   d. For **Connected service environments**, select a service environment from the list to associate with the job queue.
5. (Optional) For **Job state limits**:
   a. For **Misconfiguration**, choose `SERVICE_ENVIRONMENT_MAX_RESOURCE` and enter the **Maximum runnable time (seconds)**.
   b. For **Capacity**, choose `INSUFFICIENT_INSTANCE_CAPACITY` and enter the **Maximum runnable time (seconds)**.
6. Choose **Create job queue**

Create a SageMaker Training job queue (AWS CLI)

Use the `create-job-queue` command to create a SageMaker Training job queue.

The following example creates a basic SageMaker Training job queue that uses a service environment:

```
aws batch create-job-queue \
  --job-queue-name my-sm-training-fifo-jq \
  --job-queue-type SAGEMAKER_TRAINING \
  --priority 1 \
  --service-environment-order order=1,serviceEnvironment=ExampleServiceEnvironment
```

Replace *`ExampleServiceEnvironment`* with the name of your service environment.

The command returns output similar to the following:

```
{
  "jobQueueName": "my-sm-training-fifo-jq",
  "jobQueueArn": "arn:aws:batch:region:account:job-queue/my-sm-training-fifo-jq"
}
```

After creating your job queue, verify that it was created successfully and is in a valid state.

Use the `describe-job-queues` command to view details about your job queue:

```
aws batch describe-job-queues --job-queues my-sm-training-fifo-jq
```

The command returns output similar to the following:

```
{
  "jobQueues": [
    {
      "jobQueueName": "my-sm-training-fifo-jq",
      "jobQueueArn": "arn:aws:batch:region:account:job-queue/my-sm-training-fifo-
jq",
      "state": "ENABLED",
      "status": "VALID",
      "statusReason": "JobQueue Healthy",
      "priority": 1,
      "computeEnvironmentOrder": [],
      "serviceEnvironmentOrder": [
        {
          "order": 1,
          "serviceEnvironment": "arn:aws:batch:region:account:service-
environment/ExampleServiceEnvironment"
        }
      ],
      "jobQueueType": "SAGEMAKER_TRAINING",
      "tags": {},
      "jobStateTimeLimitActions": []
    }
  ]
}
```

Ensure that:

- The `state` is `ENABLED`
- The `status` is `VALID`

- The `statusReason` is `JobQueue Healthy`
- The `jobQueueType` is `SAGEMAKER_TRAINING`
- The `serviceEnvironmentOrder` references your service environment

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 in the *AWS Batch API Reference*.

> **Note**
>
> You can generate a job queue template with the following AWS CLI command.
>
> ```
> $ aws batch create-job-queue --generate-cli-skeleton
> ```

```
{
   "computeEnvironmentOrder": [
      {
         "computeEnvironment": "",
         "order": 0
      }
   ],
   "jobQueueName": "",
   "jobStateTimeLimitActions": [
      {
         "state": "RUNNABLE",
         "action": "CANCEL",
   "maxTimeSeconds": 0,
         "reason": ""

      }
   ],
   "priority": 0,
   "schedulingPolicyArn": "",
   "state": "ENABLED",
   "tags": {
```

```
        "KeyName": ""
    }
}
```

View job queue status

After you create a job queue and submit the jobs, it is important to be able to monitor its progress. You can use the **Job details** page to review, manage, and monitor your job queue.

View job queue information

From the AWS Batch console, select **Job queues** in navigation pane and choose your desired job queue to view its details. On this page, you can review and manage your job queue and see additional information about the queue's operations, such as the job queue snapshot, job state limits, environment order, tags, and the job queue's JSON code.

Job queue details

This section provides an overview and maintenance options for the job queue. It is important to note that you can find the Amazon Resource Name (ARN) in this section.

To find this information through the AWS Command Line Interface, use the `DescribeJobQueues` operation along with the job queue name, or the corresponding ARN.

Job queue snapshot

This section provides a static list of the first 100 `RUNNABLE`jobs that are in queue. You can use the search field to narrow the list by searching for information from any column in the results section. The jobs in the snapshot results area are sorted according to the job queue's run strategy. For first-in-first-out (FIFO) job queues, the ordering of the jobs is based on the submission time. For fair-share scheduling job queues, the ordering of the jobs is based on the job priority and share usage.

Because the results are a snapshot of the job queue, the results list doesn't automatically update. To update the list, choose the refresh at the top of the section. Choose the job's name hyperlink to navigate to **Job details** and view the job's status and other related information.

To find this information through the AWS CLI, use the `GetJobQueueSnapshot` operation along with the job queue name or the corresponding ARN.

```
aws batch get-job-queue-snapshot --job-queue my-sm-training-fifo-jq
```

Job state limits

Use this tab to review configuration information about the amount of time that a job can remain in a `RUNNABLE` state before it's canceled.

To find this information through the AWS CLI, use the `DescribeJobQueues` operation along with the job queue name or the corresponding ARN.

Environment order

If your job queue runs in multiple environments, this tab provides their order and an overview.

To find this information through the AWS CLI, use the `DescribeJobQueues` operation along with the job queue name or the corresponding ARN.

Tags

Use this tab to review and manage tags that are associated to this job queue.

JSON

Use this tab to copy the JSON code that's associated with this job queue. You can then reuse the JSON for AWS CloudFormation templates and AWS CLI scripts.

Monitor service jobs

You can monitor the status of service jobs in your job queue using several AWS Batch commands. Service jobs are jobs that run on AWS services such as SageMaker Training, where AWS Batch provides scheduling and queuing capabilities while the target service handles job execution.

List service jobs by status

Use the `ListServiceJobs` operation to view service jobs in your queue filtered by status. Service jobs can have the following statuses:

- `SUBMITTED` - Job has been submitted but not yet processed
- `PENDING` - Job is pending and waiting for resources

- `RUNNABLE` - Job is ready to run and waiting in the queue
- `STARTING` - Job is being started
- `RUNNING` - Job is currently running
- `SCHEDULED` - Job has been submitted to the target service but not yet running
- `SUCCEEDED` - Job completed successfully
- `FAILED` - Job failed to complete

View running jobs in your queue:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status RUNNING
```

View jobs waiting in the queue:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status RUNNABLE
```

View jobs that have been submitted to SageMaker but not yet running:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status SCHEDULED
```

View all succeeded jobs:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status SUCCEEDED
```

View failed jobs for troubleshooting:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status FAILED
```

Filter service jobs

You can filter service jobs by name using pattern matching. If a filter value ends with an asterisk (*), it matches any job name that begins with the string before the '*'.

Find jobs with names starting with "training":

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --filters name=JOB_NAME,values=training*
```

Find jobs with specific names:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --filters name=JOB_NAME,values=my-training-job-1,my-training-job-2
```

Combine status and name filters:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status RUNNING \
  --filters name=JOB_NAME,values=production*
```

Handle large result sets

When you have many service jobs, use pagination to manage the results effectively.

Limit the number of results returned:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --max-results 10
```

Use the next token to get additional results:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --max-results 10 \
  --next-token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Get detailed service job information

Use the `DescribeServiceJob` operation to get comprehensive information about a specific service job, including its current status, service resource identifiers, and detailed attempt information.

View detailed information about a specific job:

```
aws batch describe-service-job \
  --job-id a4d6c728-8ee8-4c65-8e2a-9a5e8f4b7c3d
```

This command returns comprehensive information about the job, including:

- Job ARN and current status
- Service resource identifiers (such as SageMaker Training job ARN)
- Scheduling priority and retry configuration
- Service request payload containing the original service parameters
- Detailed attempt information with start and stop times
- Status messages from the target service

Monitor SageMaker Training jobs

When monitoring SageMaker Training jobs through AWS Batch, you can access both AWS Batch job information and the underlying SageMaker Training job details.

The service resource identifier in the job details contains the SageMaker Training job ARN:

```
{
  "latestAttempt": {
    "serviceResourceId": {
      "name": "TrainingJobArn",
      "value": "arn:aws:sagemaker:us-east-1:123456789012:training-job/my-training-job"
    }
  }
}
```

You can use this ARN to get additional details directly from SageMaker:

```
aws sagemaker describe-training-job \
  --training-job-name my-training-job
```

Monitor job progress by checking both AWS Batch status and SageMaker Training job status. The AWS Batch job status shows the overall job lifecycle, while the SageMaker Training job status provides service-specific details about the training process.

Terminate service jobs

Use the `TerminateServiceJob` operation to stop a running service job.

Terminate a specific service job:

```
aws batch terminate-service-job \
  --job-id a4d6c728-8ee8-4c65-8e2a-9a5e8f4b7c3d \
  --reason "Job terminated by user request"
```

When you terminate a service job, AWS Batch stops the job and notifies the target service. For SageMaker Training jobs, this will stop the training job in SageMaker AI as well.

Delete a job queue in AWS Batch

When you no longer need your job queue, you can disable and delete the job queue.

Delete a job queue (AWS Batch console)

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. In the navigation pane, choose **Job queues** and then choose a job queue.
3. Choose **Actions** and then **Disable**.
4. Once the job queue's state is **Disabled**, choose **Actions** and then **Delete**.
5. In the modal window choose **Delete job queue**.

Delete a job queue (AWS CLI)

1. Disable the job queue to prevent new job submissions:

   ```
   aws batch update-job-queue \
     --job-queue my-sm-training-fifo-jq \
   ```

```
  --state DISABLED
```

2. Wait for any running jobs to complete, then delete the job queue:

```
aws batch delete-job-queue \
  --job-queue my-sm-training-fifo-jq
```

Fair-share scheduling policies

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 Use fair-share scheduling policies to assign share identifiers.

Topics

- Use share identifiers to identify workloads
- Use fair-share scheduling policies to assign share identifiers
- Use fair-share scheduling to help schedule jobs
- Tutorial: Create a scheduling policy
- Reference: Scheduling policy template

Use share identifiers to identify workloads

You can use share identifiers to tag jobs and differentiate between users and workloads. The AWS Batch scheduler tracks usage for each share identifier by using the *`(T * 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 share identifier without overriding it.

> **Note**
>
> Share identifiers are unique within a job queue and are not aggregated across job queues.

You can set fair-share 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 fair-share 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 fair-share scheduling priority.

> **Note**
>
> Attached compute resources are allocated equally among all share identifiers unless explicitly overridden.

Use fair-share scheduling policies to assign share identifiers

You can use scheduling policies to configure how compute resources in a job queue are allocated between users or workloads. Using fair-share scheduling policies, you can assign different share identifiers to workloads or users. AWS Batch assigns each 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 share identifiers that aren't active by specifying a compute reservation. For more information, see SchedulingPolicyDetail.

Use fair-share scheduling to help schedule jobs

Fair-share scheduling provides a set of controls to help schedule jobs.

> **Note**
>
> For more information about scheduling policy parameters, see SchedulingPolicyDetail.

- **Share decay seconds –** The period of time (in seconds) that the AWS Batch scheduler uses to calculate a fair-share percentage for each 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: *`shareDecaySeconds + OrderMinutes`* where *`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 *`(computeReservation/100)^ActiveFairShares`* where `ActiveFairShares` is the number of active 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. For 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.

When the job queue is active and processing jobs, you can review a list of the first 100 `RUNNABLE` jobs through the Job queue snapshot. For more information, see View job queue status.

Tutorial: Create a scheduling policy

Before you can create a job queue with a scheduling policy, you must create a scheduling policy. When you create a fair-share scheduling policy, you associate one or more share identifiers or 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 fair-share scheduling policy's share decay time. A longer share decay time will use considerably more compute resource usage over a longer time when scheduling jobs. This can allow jobs using a share identifier to temporarily use more compute resources than the weight for that share identifier would allow if that share identifier had not recently been using compute resources.
6. (Optional) For **Compute reservation**, enter an integer value for the fair-share scheduling policy's compute reservation. The compute reservation will hold some vCPUs in reserve to be used for share identifiers that are not currently active.

   The reserved ratio is `(`*`computeReservation`*`/100)^`*`ActiveFairShares`* where *ActiveFairShares* is the number of active 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 share identifier, 25% if there are two share identifiers, and 12.5% if there are three share identifiers. A `computeReservation` value of 25 indicates that AWS Batch should reserve 25% of the maximum available VCPU if there is only one share identifier, 6.25% if there are two share identifiers, and 1.56% if there are three share identifiers.
7. In the **Share attributes** section, you can specify the share identifier and weight for each share identifier to associate with the fair-share scheduling policy.

   a. Choose **Add share identifier**.
   b. For **Share identifier**, specify the share identifier. If the string ends with '*', this becomes a share identifier prefix used to match share identifiers for jobs. All of the share identifiers and share identifier prefixes in a scheduling policy must be unique and cannot overlap. For example, you can't have share identifiers prefix 'UserA*' and share identifier 'UserA1' in the same fair-share scheduling policy.

   c. For **Weight factor**, specify the relative weight for the share identifier. The default value is 1.0. A lower value has a higher priority for compute resources. If a share identifier prefix is used, jobs with 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 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 Tag your AWS Batch resources.
9. Choose **Submit** to finish and create your scheduling policy.

Reference: 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*.

Note

You can generate a job queue template with the following AWS CLI command.

```
$ aws batch create-scheduling-policy --generate-cli-skeleton
```

```
{
    "name": "",
    "fairsharePolicy": {
        "shareDecaySeconds": 0,
        "computeReservation": 0,
        "shareDistribution": [
            {
                "shareIdentifier": "",
                "weightFactor": 0.0
            }
        ]
    },
    "tags": {
        "KeyName": ""
    }
```

Reference: Scheduling policy template 206

```
}
```

Resource-aware scheduling

AWS Batch schedules jobs based on the vCPU, GPU and the memory availability in the Compute Environment (CE) associated with the Job Queue (JQ). But sometimes, just the availability of these CE resources doesn't guarantee that the job will run successfully since it might be dependent on other required resources, so these jobs are cancelled or terminated. This results in inefficient use of compute resources. To solve this issue, resource-aware scheduling can check the availability of dependent, non-CE resources before it schedules the job to run on a CE.

AWS Batch resource-aware scheduling lets you schedule jobs based on consumable resources that are needed to run your jobs– 3rd party license tokens, database access bandwidth, the need to throttle calls to a third-party API, and so on. You specify the consumable resources which are needed for a job to run, and Batch takes these resource dependencies into account when it schedules a job. You can avoid making manual interventions to eliminate job failures and long waits caused by a shortage of consumable resources. You can reduce the under-utilization of compute resources by allocating only the jobs that have all the required resources available.

Resource-aware scheduling is available for both FIFO and Fair-share scheduling policies and can be used with all compute platforms supported by Batch including EKS, ECS, and Fargate. It can be used with Array jobs, Multi-node parallel (MNP) jobs, and with regular Batch jobs.

To configure resource-aware scheduling, you first specify all the consumable resources needed to run your jobs, along with the total count available of each resource. Then, for each job that requires a consumable resource, you specify the name and required quantities of each resource needed. Batch keeps track of how many consumable resources are available for the jobs in your job queues and makes sure that a job is scheduled to run only when all the required consumable resources are available for the job to run successfully.

Topics

- Create consumable resources
- Specify the resources needed to run a job
- Check how many resources are in-use and available
- Update the quantity of a resource while it is in use by jobs
- Find the jobs that require a specific consumable resource

- Delete a consumable resource

Create consumable resources

You must first create the consumable resources that represent the non-CE resources that are consumed when a job is running and are only available in limited quantities. Each consumable resource has a:

- resource name (`consumableResourceName`) that must be unique at the account level.
- (optional) resource type (`resourceType`) that indicates whether the resource is available to be re-used after a job completes. This can be one of:
  - `REPLENISHABLE` (default)
  - `NON_REPLENISHABLE`
- total quantity (`totalQuantity`) that specifies the total amount of the consumable resource available.

The maximum number of consumable resources per account is 50k.

Console:

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. Choose **Create consumable resource**.
3. Enter a unique **Resource name**, the **Total resource quantity**, and select whether the **Type of resource** is **Replenishable** or **Non-replenishable**.
4. Choose **Create consumable resource**.

API:

Use the `CreateConsumableResource` API to define the resources you want.

Specify the resources needed to run a job

When you register a job you can specify the name of one or more resources you created (`consumableResource`) and the quantity of that resource each instance of the job requires (`quantity`).

Batch keeps track of the available units of each resource at any given moment. For each job in the job queue, the Batch scheduler ensures that your job runs only when the specified resource dependencies are available.

If a consumable resource for the job is not available when the job reaches the head of the queue, the job will wait in `RUNNABLE` state until all the required resources become available or the job state time limit is reached (see View job queue status). Once Batch has validated that all the resources are available, the job transitions to the `STARTING` state and then to `RUNNING`. Resources are locked once the job moves to `STARTING` and are then unlocked when the job moves to `SUCCEEDED` or `FAILED`.

You can also update the quantity of a resource needed for a specific job when you submit the job.

Console:

To specify resources and their needed quantities when you define a job:

1. Define a job using the job definition wizard from the AWS Batch console (**Job definitions** -> **Create**).
2. In the wizard's Step 4: **Configure containers**, under **Consumable resource**, select the **Name** of a required resource from the list. In the **Requested value** field, enter the quantity of this resource needed by an instance of this job, then choose **Add consumable resource**.
3. Repeat the previous step for all the consumable resources required by the job. You can specify up to 5 resources for each job you define.
4. You'll see a list of the consumable resources you have created after you complete the job definition wizard but before you choose **Create job definition**.

To update needed quantities of resources when you submit a job:

1. In the left navigation pane of the AWS Batch console, choose **Jobs**, then choose **Submit new job**.
2. In the wizard's Step 2: **Configure overrides**, under **Consumable resource overrides**, enter a new **Requested value** for any consumable resource whose needed quantity you want to override for the job.
3. After you have completed all the overrides you want to make for this job, choose **Next** to continue to **Review and submit**.

API:

When you register a job with the `RegisterJobDefinition` API, use the `consumableResourceList` in the `consumableResourceProperties` portion of the request to specify the consumable resources required to run an instance of the job, and the quantity of each.

When you submit a job with the `SubmitJob` API you can override the list of consumable resources and the quantity of each using the `consumableResourcePropertiesOverride` portion of the request. Note that this only overrides the quantity of the resource needed by each instance of the job, not the total quantity available.

Check how many resources are in-use and available

Batch lets you query the number of available resources (`availableQuantity`), the number of resources in use (`inUseQuantity`), and the total resources (`totalQuantity`) at a given moment.

Once a job goes to the `STARTING` state, the consumed resources will be subtracted from the available quantity of that resource. If the resource is `REPLENISHABLE`, the number of consumed resources will be added back to the available quantity as soon as the job has moved to either the SUCCEEDED or FAILED state, and the total quantity will remain the same. If the resource is `NON_REPLENISHABLE`, the number of consumed resources is subtracted from both the total and available quantities and won't be added back whether the job moves to the `SUCCEEDED` or `FAILED` state.

> **Note**
>
> This information may lag by up to 30 seconds.

Console:

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. Select either the **Replenishable** or **Non-replenishable** tab to view the resources of that type that you have created.
3. For each **Replenishable** resource, the console displays the **Name**, the **Total** quantity of the resource, the number currently **In-use** and how many are still **Available**, along with a calculation of the **Utilization** (the number of resources in-use divided by the total quantity of that resource).

For each **Non-replenishable** resource, the console displays the **Name**, the number currently **In-use** and how many are still **Available**.

You can also view current information about consumable resources from a job detail page in the console.

1. In the left navigation panel of the AWS Batch console, choose **Jobs**, then select a job's name to open the details page for that job.
2. Information on both **Replenishable resources** and **Non-replenishable resources** are available to view if the job requires them. For both types, the console displays the resource's **Name**, the **Requested** quantity for the job, how many are still **Available**, the number currently **In-use**, the **Total** quantity of the resource, along with a calculation of the **Current utilization** (the number of resources in-use by the job divided by the total quantity of that resource).

API:

Use the `DescribeConsumableResource` API which returns the following information:

```
{
   "availableQuantity": number,
   "consumableResourceArn": "string",
   "consumableResourceName": "string",
   "createdAt": number,
   "inUseQuantity": number,
   "resourceType": "string",
   "tags": {
      "string" : "string"
   },
   "totalQuantity": number
}
```

The `ListConsumableResources` API also reports the number of resources in use (`inUseQuantity`) and the total number of resources currently available (`totalQuantity`) as part of its listing of all the consumable resources you have created in your account. This API also allows you filter the consumable resource list query based on the consumable resource name.

Update the quantity of a resource while it is in use by jobs

You can reset the total quantity of a resource to a new value, add to the total quantity or subtract from it.

If the new total quantity you specify is greater than before, Batch schedules more jobs accordingly. If the new total quantity is less than before and there are no units of this resource in use, Batch just reduces the total (or available) quantity. If there are units in use, Batch reduces the available quantity immediately and, as jobs finish, Batch reduces the total (available) quantity so that it eventually arrives at the new number.

Console:

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. Select either the **Replenishable** or **Non-replenishable** tab to view the resources of that type that you have created.
3. For **Replenishable** resources:

    1. Choose the resource you want to update, then select **Actions** and choose **Set resources**, **Add resources**, or **Remove resources**.
    2. A pop-up window appears in which you can **Set total value**, **Add resources**, or **Remove resources** depending on which action you chose in the previous step. Enter the quantity you want to set as the new total value, you want to add to the total quantity or that you want to subtract from the total quantity, then select **Ok**.

    For **Non-replenishable** resources:

    1. Choose the resource you want to update, then select **Actions** and choose **Set resources**, **Add resources**, or **Remove resources**.
    2. A pop-up window appears in which you can **Set available value**, **Add resources**, or **Remove resources** depending on which action you chose in the previous step. Enter the quantity you want to set as the new available value, you want to add to the available quantity or that you want to subtract from the available quantity, then select **Ok**.

API:

Use the `UpdateConsumableResource` API to set a new total quantity for the resource, or to increase or reduce the total quantity.

Find the jobs that require a specific consumable resource

Batch lets you retrieve a list of jobs that require a specific consumable resource.

Console:

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. In the list, select the name of the consumable resource. The details page for that resource opens.
3. Under **Search jobs**, enter any filters you want to apply to the list of jobs. You can filter by the **Job name** ('equals', 'starts with'), the **Date range** (when the job was created), and by **Additional criteria** ('job queue', 'job definition', 'shared job identifier'). For each type of filter you want to apply, select from the available options in the drop-down list and enter any additional information requested.

   Choose **Search**.
4. A (filtered) list of jobs that require the consumable resource is displayed, including the job's name, status, number of requested units of the consumable resource, other needed consumable resources, and so on. Using this list, you can select one or more jobs to **Cancel** or **Terminate**. You can also select a job's name to open that job's detail page.
5. Under **Search jobs** you can now **Refresh results** or **Clear search** and start over.

API:

You can get a list of jobs that use a specific consumable resource with the `ListJobsByConsumableResource` API. This API also lets you filter the job list query using the job status or the job name.

Delete a consumable resource

You can delete a consumable resource at any time, even when the jobs that require the resource are still running. Once a consumable resource is deleted, there may be a gap between the time the delete command is received and the job scheduler honors the delete, so it's possible jobs that consume the resource may be scheduled right after the delete call. If the deleted consumable resource has resource type (`resourceType`) `REPLENISHABLE`, this will be ignored when the jobs complete. If you delete a consumable resource and re-create it with the same name, it is considered to be the same resource and it can be used by `RUNNABLE` jobs.

Console:

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. Select either the **Replenishable** or **Non-replenishable** tab to view the resources of that type that you have created.
3. Select each resource you want to delete, then choose **Delete**. A pop-up window **Delete consumable resource** appears. To confirm the deletion, choose **Delete**.

You can also delete a consumable resource from its detail page in the console.

1. In the left navigation panel of the AWS Batch console, choose **Consumable resources**.
2. Select either the **Replenishable** or **Non-replenishable** tab to view the resources of that type that you have created.
3. Choose the name of the resource you want to delete. The details page of the consumable resource appears. Choose **Delete**. A pop-up window **Delete consumable resource** appears. To confirm the deletion, choose **Delete**.

API:

Use the `DeleteConsumableResource` API to delete a consumable resource.

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.

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.

Contents

- Create a single-node job definition
- Create a multi-node parallel job definition
- Job definition template that uses ContainerProperties
- Create job definitions using EcsProperties
- Use the awslogs log driver
- Specify sensitive data
- Private registry authentication for jobs
- Amazon EFS volumes
- Job definition examples

Create 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 (also known as *gang scheduling*).

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

Topics

- Create a single-node job definition on Amazon EC2 resources
- Create a single-node job definition on Fargate resources
- Create a single-node job definition on Amazon EKS resources
- Create a single-node job definition with multiple containers on Amazon EC2 resources

Create a single-node job definition on Amazon EC2 resources

Complete the following steps to create a single-node job definition on Amazon Elastic Compute Cloud (Amazon EC2) resources.

To create a new job definition on Amazon EC2 resources:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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`).

- 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**, enter the commands into the field as their **JSON** string array equivalent.

   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 default values for parameter substitution 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. 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*.

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.

    > **Note**
    >
    > To maximize your resource utilization, prioritize memory for jobs of a specific instance type. For more information, see Compute resource memory management.

    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 LogConfiguration:secretOptions.
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 LinuxParameters:swappiness.

    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. In the **Task properties** section:

    a. For **Execution role - conditional**, choose a role to allow Amazon ECS agents to make AWS API calls on your behalf. For more information on creating an **Execution role**, see Tutorial: Create the IAM execution role.

   b. Choose **Enable ECS execute command**, to enable access to the Amazon ECS container shell directly and bypass the host OS. You must choose a **Task role**.

      > ⚠ **Important**
      >
      > The **ECS execute** command requires that file system be writable.

   c. For **Task role**, choose an Amazon ECS Identity and Access Management (IAM) role to allow the container to make AWS API calls on your behalf. For more information see, Amazon ECS task IAM role in the *Amazon Elastic Container Service Developer Guide*.

22. (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 LogConfiguration:logDriver.

      > ⓘ **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.

      > ⓘ **Tip**
      >
      > For more information, see LogConfiguration:secretOptions.

23. Choose **Next page**.

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

Create a single-node job definition on Fargate resources

Complete the following steps to create a single-node job definition on AWS Fargate resources.

To create a new job definition on Fargate resources:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 Fargate compute environments.
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 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 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**, enter the commands into the field as their JSON string array equivalent.

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 default values for parameter substitution and placeholders in your command. For more information, see Parameters.

c. (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**

d. 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.

    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. (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 LogConfiguration:secretOptions.
    g. 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 **Task properties** section:
        i. For **Execution role - conditional**, choose a role to allow Amazon ECS agents to make AWS API calls on your behalf. For more information on creating an **Execution role**, see Tutorial: Create the IAM execution role.
        ii. Choose **Enable ECS execute command**, to enable access to the Amazon ECS container shell directly and bypass the host OS. You must choose a **Task role**.

> **Important**
>
> The **ECS execute** command requires that file system be writable.

iii. For **Task role**, choose an Amazon ECS Identity and Access Management (IAM) role to allow the container to make AWS API calls on your behalf. For more information see, Amazon ECS task IAM role in the *Amazon Elastic Container Service Developer Guide*.

h. 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 LogConfiguration:logDriver.

> **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 LogConfiguration:secretOptions.

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**.

Create a single-node job definition on Amazon EKS resources

Complete the following steps to create a single-node job definition on Amazon Elastic Kubernetes Service (Amazon EKS).

To create a new job definition on Amazon EKS resources:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.

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 **Volumes**, select **Add volume**, then:

   i. Add a **Name** for your volume.

   ii. (Optional) Add the **Host path** for the directory on the host.

   iii. (Optional) Add a **Medium** and a **Size limit** to configure a Kubernetes emptyDir.

   iv. (Optional) Provide a **Secret name** for the pod and whether the secret is **Optional**.

   v. (Optional) Define a **Claim name** to attach a Kubernetes Persistent Volume Claim to the pod, and whether it is **Read only**.

e. (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/`.

f. (Optional) For **Pod annotations**, choose **Add annotations**, then enter a name-value pair.

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

g. Choose **Next page**.

h. 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**

> **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 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.

i. (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**

j. 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.

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

l. (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. Enter a **SubPath** to specify a sub-path inside the referenced volume instead of its root.

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

    iv. Choose **Add volume mount**.

m. (Optional) For **Run as user**, enter a user ID to run the container process.

> **Note**
>
> The user ID must exist in the image for the container to run.

n. (Optional) For **Run as group**, enter a group ID to run the container process runtime.

> **Note**
>
> The group ID must exist in the image for the container to run.

o. (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**](#).

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

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

> **Note**
>
> If **Run as non-root** is turned on, the kubelet validates the image at runtime to verify that image doesn't run as UID 0.

r. 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**.

Create a single-node job definition with multiple containers on Amazon EC2 resources

Complete the following steps to create a single-node job definition with multiple containers on Amazon Elastic Compute Cloud (Amazon EC2) resources.

To create a new job definition on Amazon EC2 resources:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 **Job definition structure**, turn off **Use legacy containerProperties structure** processing.

7. For **EC2 platform configuration**, turn off **Enable multi-node parallel** processing.
8. Choose **Next**.
9. In **General configuration** section, enter the following:

   a. 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 (_).

   b. For **Execution timeout - *optional***, 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.

   c. Turn on **Scheduling priority - *optional***. Enter a scheduling priority value between 0 and 100. Higher values are given higher priority.

   d. Expand **Tags - *optional*** and then choose **Add tag** to add tags to the resource. Enter a key and optional value, then choose **Add tag**.

   e. Turn on **Propagate tags** to propagate tags from the job and job definition to the Amazon ECS task.

10. In **Retry strategy - *optional*** section, enter the following:

    a. 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.

    b. 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**.

11. In **Task properties** section, enter the following:

   a. For **Execution role - *conditional***, choose a role to allow Amazon ECS agents to make AWS API calls on your behalf. For more information on creating an **Execution role**, see Tutorial: Create the IAM execution role.
   b. Choose **Enable ECS execute command**, to enable access to the Amazon ECS container shell directly and bypass the host OS. You must choose a **Task role**.

      > ⚠ **Important**
      >
      > The **ECS execute** command requires that file system be writable.

   c. For **Task role**, choose an Amazon ECS Identity and Access Management (IAM) role to allow the container to make AWS API calls on your behalf. For more information see, Amazon ECS task IAM role in the *Amazon Elastic Container Service Developer Guide*.
   d. For **IPC mode** choose `host`, `task`, or `none`. If `host` is specified, then all the containers that are within the tasks that specified the host IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If task is specified, all the containers that are within the specified task share the same IPC resources. If none is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance.
   e. For **PID mode** choose `host` or `task`. For example, monitoring sidecars might need `pidMode` to access information about other containers running in the same task. If `host` is specified, all containers within the tasks that specified the host PID mode on the same container instance share the same process namespace with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same process namespace. If no value is specified, the default is a private namespace for each container.

12. In the **Consumable resource** section, enter the following:
   a. Enter a unique **Name** and the **Requested value**.
   b. You can add more consumable resources by choosing **Add consumable resource**.
13. In the **Storage** section, enter the following:
   a. Enter a **Name** and **Source path** for the volume and then choose **Add volume**. You can also choose to turn on Enable EFS.

   b. You can add more Volumes by choosing **Add volume**.

14. For **Parameters**, choose **Add parameters** to add parameter substitution placeholders as **Key** and optional **Value** pairs.
15. Choose **Next page**.
16. In the **Container configuration** section:

    a. For **Name**, enter a name for the container.

    b. For **Essential container**, enable if the container is essential.

    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 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`).
    - 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 **Resource requirements** configure each of the following:

   i. For **vCPUs**, choose the number of CPUs for the container.
   ii. For **Memory**, choose the amount of memory for the container.
   iii. For **GPU - *optional***, choose the number of GPUs for the container.

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

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

g. Turn on **Privileged** to give the job container elevated permissions on the host instance, similar to the root user.

h. For **Command**, enter the commands into the field as their **JSON** string array equivalent.

   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 default values for parameter substitution and placeholders in your command. For more information, see Parameters.

i. For **Repository credentials - *optional*** enter the ARN of the secret containing your credentials.

j. For **Environment variables - *optional***, choose **Add environment variables** to add environment variables to pass to the container.

k. In the **Linux parameters - *optional*** section:

   i. Turn on **Enable init process** to run an init process inside the container.
   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.
   v. 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**.

vi. For **Tmpfs**, choose **Add tmpfs** to add a `tmpfs` mount.

l.

> **Note**
>
> Firelens logging has to be done in a dedicated container. To configure Firelens logging:
>
> - In every container, except your dedicated firelens container, set the **Logging driver** to `awsfirelens`
> - In your Firelens container set the **Firelens Configuration - optional** and the **Logging configuration - *optional*** to the logging destination

In the **Firelens Configuration - optional** section:

> **Important**
>
> AWS Batch enforces `host` network mode on non-MNP, non-FARGATE Amazon ECS jobs. Root user is required for Amazon ECS Firelens. When running tasks that use the `host` network mode, Amazon ECS advises against running containers using the root user (UID 0) for better security. Therefore, all non-MNP, non-FARGATE ECS jobs with Firelens logging will not meet security best practice.

i. For **Type**, choose either `fluentd` or `fluentbit`.

ii. For **Options**, enter the name/value pair of the option. You can add more **Options** using **Added option**.

m. In the **Logging configuration - *optional*** section:

        i. For **Log driver**, choose the log driver to use. For more information about the available log drivers, see [LogConfiguration:logDriver](#).

            > **Note**
            >
            > By default, the `awslogs` log driver is used.

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

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

            > **Tip**
            >
            > For more information, see [LogConfiguration:secretOptions](#).

    n. For **Mount points - *optional***, choose **Add mount points** to add mount points for data volumes. You must specify the source volume and container path.

    o. For **Secrets - *optional***, choose **Add secret** to add a secret. Then, enter a name-value pair, and choose **Add secret**.

        > **Tip**
        >
        > For more information, see [LogConfiguration:secretOptions](#).

    p. For **Ulimits - *optional***, choose **Add ulimit** to add a `ulimits` value for the container. Enter **Name**, **Soft limit**, and **Hard limit** values, and then choose **Add ulimit**.

    q. For **Dependencies - *optional***, choose **Add container dependencies**. Choose the name of the container and it's state to determine when this container starts.

17. If you only have one container configured then you must choose **Add container** and complete configuring the new container. Otherwise, choose **Next** to review.

Create 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 (also known as *gang scheduling*). For more information, see Multi-node parallel jobs.

> **Note**
>
> AWS Fargate doesn't support multi-node parallel jobs.

Contents

- Tutorial: Create a multi-node parallel job definition on Amazon EC2 resources

Tutorial: Create a multi-node parallel job definition on Amazon EC2 resources

To create a multi-node parallel job definition on Amazon Elastic Compute Cloud (Amazon EC2) resources.

> **Note**
>
> To create a *single-node* job definition, see Create a single-node job definition on Amazon EC2 resources.

To create a multi-node parallel job definition on Amazon EC2 resources:

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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, `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`).

d. For **Command**, enter the commands into the field as their **JSON** string array equivalent.

   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 default values for parameter substitution and placeholders in your command. For more information, see Parameters.

e. 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.

f. 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.

   > **Note**
   >
   > To maximize your resource utilization, you can provide your jobs as much memory as possible for a particular instance type. For more information, see Compute resource memory management.

g. (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.

h. (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.

> **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*.

i. (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 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 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 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*.

    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 in the *Amazon Elastic Container Service Developer Guide*.

20. In the **Security Configuration** section:
    1. (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**.
    2. (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**.
    3. (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 LogConfiguration:secretOptions.
21. In the **Linux configuration** section:
    1. Turn on **Enable read only filesystem** to remove write access to the volume.
    2. (Optional) Turn on **Enable init process** to run an `init` process inside the container. This process forwards signals and reaps processes.
    3. For **Shared memory size**, enter the size (in MiB) of the `/dev/shm` volume.
    4. For **Max swap size**, enter the total amount of swap memory (in MiB) that the container can use.
    5. 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 LinuxParameters:swappiness.
    6. (Optional) For **Devices**, choose **Add device** to add a device:
        1. 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.
        2. For **Host path**, specify the path of a device in the host instance.
        3. 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. In the **Task properties** section:
    a. For **Execution role - conditional**, choose a role to allow Amazon ECS agents to make AWS API calls on your behalf. For more information on creating an **Execution role**, see Tutorial: Create the IAM execution role.
    b.

    > ⚠ **Important**
    >
    > To use **ECS execute command** your compute environment must meet the compute environment considerations for multi node parallel jobs.

    Choose **Enable ECS execute command**, to enable access to the Amazon ECS container shell directly and bypass the host OS. You must choose a **Task role**.

    > ⚠ **Important**
    >
    > The **ECS execute** command requires that file system be writable.

    c. For **Task role**, choose an Amazon ECS Identity and Access Management (IAM) role to allow the container to make AWS API calls on your behalf. For more information see, Amazon ECS task IAM role in the *Amazon Elastic Container Service Developer Guide*.
27. (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 LogConfiguration:logDriver.

    > ⓘ **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.

   > **Tip**
   >
   > For more information, see LogConfiguration:secretOptions.

28. Choose **Next page**.

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

Job definition template that uses 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 JobDefinition.

> **Note**
>
> You can generate a single-container job definition template with the following AWS CLI command:
>
> ```
> $ aws batch register-job-definition --generate-cli-skeleton
> ```

```
{
    "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": ""
                            },
                            "name": "",
                            "efsVolumeConfiguration": {
                                "fileSystemId": "",
                                "rootDirectory": "",
                                "transitEncryption": "DISABLED",
                                "transitEncryptionPort": 0,
                                "authorizationConfig": {
                                    "accessPointId": "",
                                    "iam": "ENABLED"
                                }
                            }
                        }
                    ],
                    "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": "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
                        }
                    }
                ]
            }
        }
}
```

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
- Type
- Parameters
- Container properties
- Amazon EKS properties
- Platform capabilities
- Propagate tags
- Node properties
- Retry strategy
- Scheduling priority
- Tags
- Timeout

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 Create 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

`environment`

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 IAM execution 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.

Type: String

Default: `LATEST`

Required: No

`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**.

> **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`).

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"
            ]
        }
    ],
    "maxSwap": integer,
    "swappiness": integer
}
```

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.

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

Type: Integer

Required: No

`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 in the *Amazon EC2 User Guide* or How do I allocate memory to work as swap space in an Amazon EC2 instance by using a swap file?.

- 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.

Type: Integer

Required: No

`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.

Required: No

`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`" | "`dirsync`" | "`remount`" | "`mand`" | "`nomand`" | "`atime`" | "`noatime`" | "`diratime`" | "`nodiratime`" | "`bind`" | "`rbind`" | "`unbindable`" | "`runbindable`" | "`private`" | "`rprivate`" | "`shared`" | "`rshared`" | "`slave`" | "`rslave`" | "`relatime`" | "`norelatime`" | "`strictatime`" | "`nostrictatime`" | "`mode`" | "`uid`" | "`gid`" | "`nr_inodes`" | "`nr_blocks`" | "`mpol`"

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.

> **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 in the *Amazon Elastic Container Service Developer Guide*.

`awslogs`

Specifies the Amazon CloudWatch Logs logging driver. For more information, see Use the awslogs log driver and Amazon CloudWatch Logs logging driver in the Docker documentation.

`fluentd`

Specifies the Fluentd logging driver. For more information including usage and options, see Fluentd logging driver 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 in the Docker documentation.

`journald`

Specifies the journald logging driver. For more information including usage and options, see Journald logging driver in the Docker documentation.

`json-file`

Specifies the JSON file logging driver. For more information including usage and options, see JSON File logging driver in the Docker documentation.

`splunk`

Specifies the Splunk logging driver. For more information including usage and options, see Splunk logging driver in the Docker documentation.

`syslog`

Specifies the syslog logging driver. For more information including usage and options, see Syslog logging driver 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 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 Specify sensitive data.

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

`type`

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**
>
> 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.

`VCPU`	`MEMORY`
0.25 vCPU	512, 1024, and 2048 MiB
0.5 vCPU	1024-4096 MiB in 1024 MiB increments
1 vCPU	2048-8192 MiB in 1024 MiB increments
2 vCPU	4096-16384 MiB in 1024 MiB increments
4 vCPU	8192-30720 MiB in 1024 MiB increments
8 vCPU	16384-61440 MiB in 4096 MiB increments

VCPU	MEMORY
16 vCPU	32768-122880 MiB in 8192 MiB increments

type="VCPU"

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 Specify 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",
        "iam": "string"
      },
      "fileSystemId": "string",
      "rootDirectory": "string",
      "transitEncryption": "string",
      "transitEncryptionPort": number
    }
  }
]
```

`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 Use 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`

Required: No

`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 `ENTRYPOINT` 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 ENTRYPOINT 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

Required: No

`name`

The name the volume mount. This must match the name of one of the volumes in the pod.

Type: String

Required: No

`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

Required: No

`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

Required: No

`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

emptyDir

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

`nodeProperties`

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

`container`

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 Tag your AWS Batch resources.

Type: String to string map

Required: No

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

Create 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:

```
{
    "containerProperties": {
      ...
      "image": "my_ecr_image1",
      ...
   },
...
}
```

A job definition that uses `EcsProperties` on Amazon ECS and Fargate has the following structure:

```
{
  "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 `ContainerProperties` 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 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 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 values is `false`. Setting this to `true` to enable containers see and signal processes in other containers that 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 request syntax for eksProperties.

Reference: 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 ECS on Amazon EC2

The following is an example of a AWS Batch job for Amazon Elastic Container Service on Amazon Elastic Compute Cloud:

```
{
    "jobDefinitionName": "multicontainer-ecs-ec2",
    "type": "container",
    "ecsProperties": {
        "taskProperties": [
          {
            "containers": [
              {
                "name": "c1",
                "essential": false,
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "2"
                  },
                  {
                    "type": "MEMORY",
                    "value": "4096"
                  }
                ]
              },
              {
                "name": "c2",
                "essential": false,
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "2"
                  },
                  {
                    "type": "MEMORY",
                    "value": "4096"
                  }
                ]
              },
              {
                "name": "c3",
                "essential": true,
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "firelensConfiguration": {
                  "type": "fluentbit",
                  "options": {
                    "enable-ecs-log-metadata": "true"
                  }
                 },
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "6"
                  },
                  {
                    "type": "MEMORY",
                    "value": "12288"
                  }
                ]
              }
            ]
          }
        ]
  }
```

```
}
```

AWS Batch job for Amazon ECS on Fargate

The following is an example of a AWS Batch job for Amazon Elastic Container Service 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"
                  }
                ]
              }
            ],
            "executionRoleArn": "arn:aws:iam::1112223333:role/ecsTaskExecutionRole"
          }
        ]
  }
}
```

AWS Batch job for Amazon EKS

The following is an example of a 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"
            }
          }
        }
      ],
      "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"
            }
          }
          }
        ]
      }
    }
}
```

MNP AWS Batch job with multiple containers per node

The following is an example of a multi-node parallel (MNP) AWS Batch job with multiple containers per node:

```
{
  "jobDefinitionName": "multicontainer-mnp",
  "type": "multinode",
  "nodeProperties": {
    "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"
                    }
                  ]
                }
              ]
            }
          ]
        }
      }
    ]
  }
}
```

Use 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*.

awslogs log driver options in the AWS Batch JobDefiniton data type

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

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

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`.

Specify a log configuration in your job definition

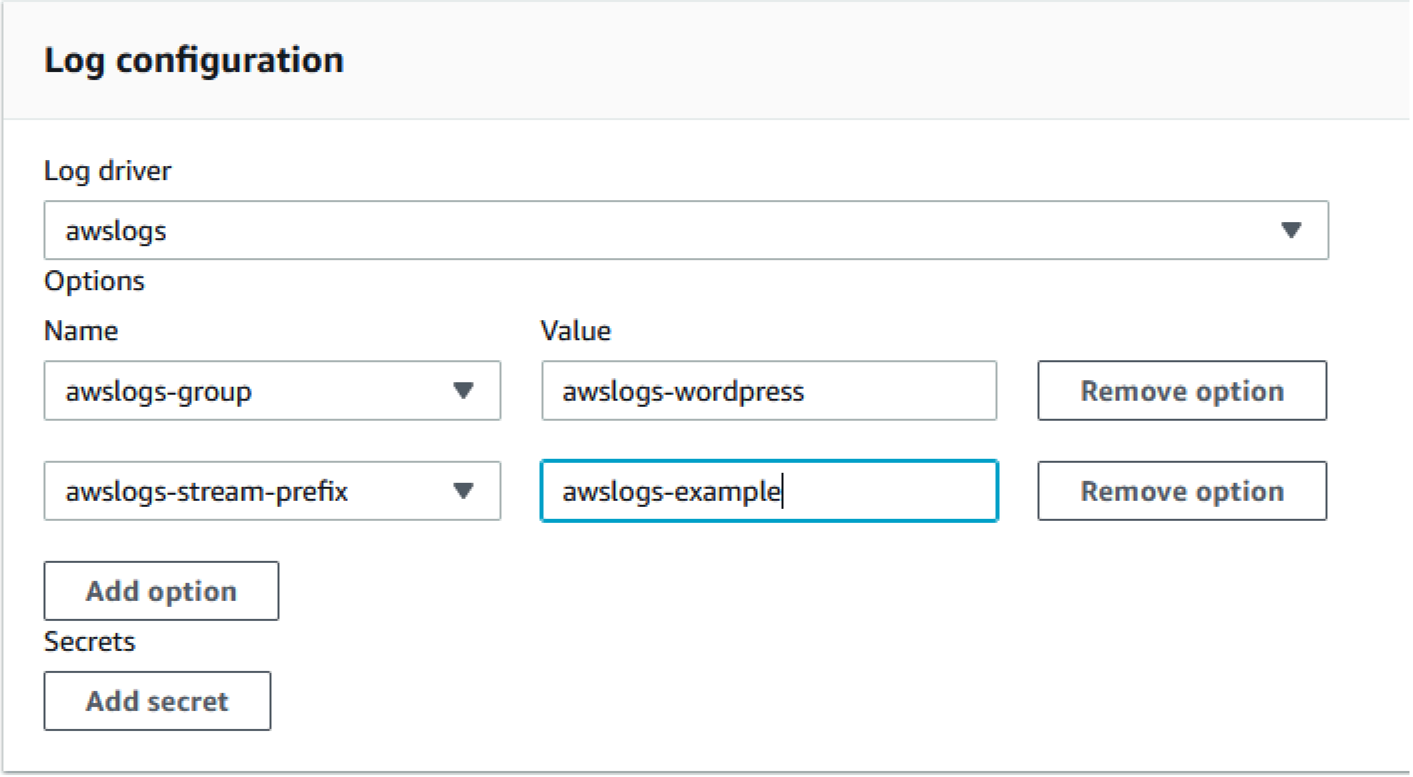
By default, AWS Batch enables the `awslogs` log driver. This section describes how to customize the `awslogs` log configuration for a job. For more information, see Create a single-node job definition .

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.

```
"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

Log driver

awslogs

Options

Name	Value
awslogs-group	awslogs-wordpress
awslogs-stream-prefix	awslogs-example

Secrets

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 [Tutorial: submit a job](#).

Specify 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

- Specify sensitive data with Secrets Manager
- Specify sensitive data with Systems Manager Parameter Store

Specify sensitive data with 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 when you specify 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 IAM execution 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-2:777777777777:secret:<secret_name>",
                "arn:aws:kms:us-east-2:777777777777:key/<key_id>"
            ]
        }
    ]
}
```

Injecting sensitive data as an environment variable

You can specify the following Within your job definition:

- 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.

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-
stage:version-id
```

The following section describes the additional parameters. These parameters are optional. However, if you don't use them, you must include the colons `:` to use the default values. Examples are provided below for more context.

`json-key`

Specifies the name of the key in a key-value pair with the value that you want to set as the environment variable value. Only values in JSON format are supported. If you 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 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 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.

```
{
  "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.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "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.

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
    }]
  }]
}
```

Example referencing a specific secret version

The following shows an example output from a [>describe-secret](#) command that displays the unencrypted contents of a secret along with the metadata for all versions of the secret.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981dd39ab30c": [
            "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:appauthexample-
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.

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf::9d4cb84b-ad69-40c0-a0ab-cead36b967e8"
    }]
  }]
}
```

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",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1:AWSPREVIOUS:"
    }]
  }]
}
```

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-cead36b967e8"
    }]
  }]
}
```

Inject sensitive data in a log configuration

When you specify a `logConfiguration` within your job definition, 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"
```

```
      }]
    }]
  }]
}
```

Create 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 in the *AWS Secrets Manager User Guide*.

To create a basic secret

Use Secrets Manager to create a secret for your sensitive data.

1. Open the 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. 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.

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

Specify sensitive data with 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 when you specify sensitive data using Systems Manager Parameter Store
- Required IAM permissions for AWS Batch secrets
- Inject sensitive data as an environment variable
- Inject sensitive data in a log configuration
- Create an AWS Systems Manager Parameter Store parameter

Considerations when you specify 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 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 IAM execution role.

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 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:us-east-2:999999999999:parameter/<parameter_name>",
                "arn:aws:secretsmanager:us-
east-2:999999999999:secret:<secret_name>",
                "arn:aws:kms:us-east-2:999999999999:key/<key_id>"
            ]
        }
    ]
}
```

Inject 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.

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

Inject sensitive data in a log configuration

Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

> **Important**
>
> If the Systems Manager Parameter Store parameter exists in the same Region as the task you'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.

```
{
  "containerProperties": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
      }]
    }]
  }]
}
```

Create 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

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
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 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:

  ```
  arn:aws:kms:us-east-2:123456789012:alias/MyAliasName
  ```

  The following is an alias name example:

  ```
  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 IAM execution 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:secretsmanager:us-
east-1:123456789012:secret:secret_name",
                "arn:aws:kms:us-east-1:123456789012:key/key_id"
            ]
        }
    ]
}
```

Tutorial: Create a secret for private registry authentication

Complete the following steps to create a secret for your private registry credentials with AWS Secrets Manager.

Create a basic secret

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

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

5. Choose **Next**.

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

   The secret name must be ASCII letters, digits, or any of the following characters: `/_+=.@-`.
7. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at **Disable automatic rotation** and choose **Next**.

   For instructions on how to configure rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets.
8. Review your settings, and then choose **Store secret** to save everything that you entered as a new secret in Secrets Manager.

Register a 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*.

Use 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*.

Specify 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:

```
{
    "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

Required: No

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

Job definition examples

The job definition examples in the following topics illustrate how to use common patterns such as environment variables, parameter substitution, and volume mounts.

Contents

- Environment variables
- Parameter substitution
- Test GPU functionality
- Multi-node parallel job

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.

```
{
    "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://amzn-s3-demo-source-bucket/myjob.sh"
            },
            {
                "name": "BATCH_FILE_TYPE",
                "value": "script"
            }
        ],
        "user": "nobody"
    }
}
```

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.

```
{
    "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 Use a GPU workload AMI is configured properly. This example job definition runs the TensorFlow deep MNIST classifier example from GitHub.

```
{
    "containerProperties": {
        "image": "tensorflow/tensorflow:1.8.0-devel-gpu",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "32000"
            },
            {
                "type": "VCPU",
                "value": "8"
            }
        ],
        "command": [
            "sh",
            "-c",
            "cd /tensorflow/tensorflow/examples/tutorials/mnist; python mnist_deep.py"
        ]
    },
    "type": "container",
    "jobDefinitionName": "tensorflow_mnist_deep"
}
```

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",
  "jobDefinitionArn": "arn:aws:batch:us-east-2:123456789012:job-definition/gromacs-
jobdef:1",
  "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"
        }
      }
    ]
  }
}
```

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 JobDefinition.

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

Topics

- Tutorial: submit a job
- Service jobs in AWS Batch
- Job states
- AWS Batch job environment variables
- Automated job retries
- Job dependencies
- Job timeouts
- Amazon EKS jobs
- Multi-node parallel jobs
- Multi-node parallel jobs on Amazon EKS
- Array jobs
- Run GPU jobs
- View AWS Batch jobs in a job queue
- Search AWS Batch for jobs in a job queue
- Networking modes for AWS Batch jobs
- View AWS Batch job logs in CloudWatch Logs
- Review AWS Batch job information

Tutorial: submit 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 Create a single-node job definition .
7. For **Job queue**, choose an existing job queue. For more information, see Create 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**, enter the commands into the field as their **JSON** string array equivalent.

    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**
    >
    > 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**.

Service jobs in AWS Batch

AWS Batch service jobs enable you to submit requests to AWS services through AWS Batch job queues. Currently, AWS Batch supports SageMaker Training jobs as service jobs. Unlike containerized jobs where AWS Batch manages the underlying container execution, service jobs allow AWS Batch to provide job scheduling and queuing capabilities while the target AWS service (such as SageMaker AI) handles the actual job execution.

AWS Batch for SageMaker Training jobs allows data scientists to submit training jobs with priorities to configurable queues, ensuring workloads run without intervention as soon as resources are available. This capability addresses common challenges such as resource coordination, preventing accidental overspending, meeting budget constraints, optimizing costs with reserved instances, and eliminating the need for manual coordination between team members.

Service jobs differ from containerized jobs in several key ways:

- **Job submission**: Service jobs must be submitted using the SubmitServiceJob API. Service jobs cannot be submitted through the AWS Batch console.
- **Job execution**: AWS Batch schedules and queues service jobs, but the target AWS service runs the actual job workload.
- **Resource identifiers**: Service jobs use ARNs that contain "service-job" instead of "job" to distinguish them from containerized jobs.

To get started with AWS Batch service jobs for SageMaker Training, see the section called “Getting started with AWS Batch on SageMaker AI”.

Topics

- Service job payloads in AWS Batch
- Submit a service job in AWS Batch
- Mapping AWS Batch service job status to SageMaker AI status
- Service job retry strategies in AWS Batch
- Monitor service jobs in an AWS Batch queue

Service job payloads in AWS Batch

When you submit service jobs using SubmitServiceJob, you provide two key parameters that define the job: `serviceJobType`, and `serviceRequestPayload`.

- The `serviceJobType` specifies which AWS service will execute the job. For SageMaker Training jobs, this value is `SAGEMAKER_TRAINING`.
- The `serviceRequestPayload` is a JSON-encoded string that contains the complete request that would normally be sent directly to the target service. For SageMaker Training jobs, this payload contains the same parameters you would use with the SageMaker AI CreateTrainingJob API.

For a complete list of all available parameters and their descriptions, see the SageMaker AI CreateTrainingJob API reference. All parameters supported by `CreateTrainingJob` can be included in your service job payload.

For examples of more training job configurations, see APIs, CLI, and SDKs in the SageMaker AI Developer Guide.

We recommend using the PySDK for service job creation because PySDK has helper classes and utilities. For an example of using PySDK, see SageMaker AI examples on GitHub.

Example service job payload

The following example shows a simple service job payload for a SageMaker Training job that runs a "hello world" training script:

This payload would be passed as a JSON string to the `serviceRequestPayload` parameter when calling `SubmitServiceJob`.

```
{
  "TrainingJobName": "my-simple-training-job",
  "RoleArn": "arn:aws:iam::123456789012:role/SageMakerExecutionRole",
  "AlgorithmSpecification": {
    "TrainingInputMode": "File",
    "TrainingImage": "763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-
training:2.0.0-cpu-py310",
    "ContainerEntrypoint": [
      "echo",
      "hello world"
    ]
  },
  "ResourceConfig": {
    "InstanceType": "ml.c5.xlarge",
    "InstanceCount": 1,
    "VolumeSizeInGB": 1
  },
  "OutputDataConfig": {
    "S3OutputPath": "s3://your-output-bucket/output"
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": 30
  }
}
```

Submit a service job in AWS Batch

To submit service jobs to AWS Batch, you use the SubmitServiceJob API. You can submit jobs using the AWS CLI or SDK.

If you don't already have an execution role then you must create one before you can submit your service job. To create the SageMaker AI execution role, see How to use SageMaker AI execution roles in the *SageMaker AI Developer guide*.

Service job submission workflow

When you submit a service job, AWS Batch follows this workflow:

1. AWS Batch receives your `SubmitServiceJob` request and validates the AWS Batch-specific parameters. The `serviceRequestPayload` is passed through without validation.
2. The job enters the `SUBMITTED` state and is placed in the specified job queue
3. AWS Batch evaluates if there is available capacity in the service environment for `RUNNABLE` jobs at the front of the queue
4. If capacity is available, the job moves to `SCHEDULED` and the job has been passed to SageMaker AI
5. When capacity has been acquired and SageMaker AI has downloaded the service job data, the service job will start initialization and the job is changed to `STARTING`.
6. When SageMaker AI starts to execute the job its status is changed to `RUNNING`.
7. While SageMaker AI executes the job, AWS Batch monitors its progress and maps service states to AWS Batch job states. For details about how service job states are mapped, see ???
8. When the service job is completed it moves to `SUCCEEDED` and any output is ready to be downloaded.

Prerequisites

Before submitting a service job, ensure you have:

- **Service environment** – A service environment that defines capacity limits. For more information, see Create a service environment in AWS Batch.
- **SageMaker job queue** – A SageMaker job queue to provide job scheduling. For more information, see Create a SageMaker Training job queue in AWS Batch.
- **IAM permissions** – Permissions to create and manage AWS Batch job queues and service environments. For more information, see AWS Batch IAM policies, roles, and permissions.

Submit a service job with the AWS CLI

The following shows how to submit a service job using the AWS CLI:

```
aws batch submit-service-job \
    --job-name "my-sagemaker-training-job" \
    --job-queue "my-sagemaker-job-queue" \
    --service-job-type "SAGEMAKER_TRAINING" \
    --service-request-payload '{\"TrainingJobName\": \"sagemaker-training-job-
example\", \"AlgorithmSpecification\": {\"TrainingImage\": \"123456789012.dkr.ecr.us-
east-1.amazonaws.com/pytorch-inference:1.8.0-cpu-py3\", \"TrainingInputMode
\": \"File\", \"ContainerEntrypoint\":  [\"sleep\", \"1\"]}, \"RoleArn\":
\"arn:aws:iam::123456789012:role/SageMakerExecutionRole\", \"OutputDataConfig\":
 {\"S3OutputPath\": \"s3://example-bucket/model-output/\"}, \"ResourceConfig\":
 {\"InstanceType\": \"ml.m5.large\", \"InstanceCount\": 1, \"VolumeSizeInGB\": 1}}'
    --client-token "unique-token-12345"
```

For more information about the `serviceRequestPayload` parameters, see the section called “Service job payloads”.

Mapping AWS Batch service job status to SageMaker AI status

When you submit jobs to a SageMaker job queue using SubmitServiceJob, AWS Batch manages the job lifecycle and maps AWS Batch job states to equivalent SageMaker Training job states. Service jobs, such as SageMaker Training jobs, follow a different state lifecycle than traditional container jobs. While service jobs share most states with container jobs, they introduce the `SCHEDULED` state and exhibit different retry behaviors, particularly for handling insufficient capacity errors from the target service.

The following table shows the AWS Batch job state and the corresponding SageMaker Status/SecondaryStatus:

Batch Status	SageMaker AI Primary Status	SageMaker AI Secondary Status	Description
`SUBMITTED`	N/A	N/A	Job submitted to queue, waiting for scheduler evaluation.

Batch Status	SageMaker AI Primary Status	SageMaker AI Secondary Status	Description
`RUNNABLE`	`N/A`	`N/A`	Job is queued and ready for scheduling. Jobs in this state are started as soon as sufficient resources are available in the service environment. Jobs can remain in this state indefinitely when sufficient resources are unavailable.
`SCHEDULED`	`InProgress`	`Pending`	Service job successfully submitted to SageMaker AI
`STARTING`	`InProgress`	`Downloading`	SageMaker Training job downloading data and images. Training job capacity has been acquired and job initialization begins.
`RUNNING`	`InProgress`	`Training`	SageMaker Training job executing algorithm
`RUNNING`	`InProgress`	`Uploading`	SageMaker Training job uploading output artifacts after training completion
`SUCCEEDED`	`Completed`	`Completed`	SageMaker Training job completed successfully. Output artifacts finished uploading.
`FAILED`	`Failed`	`Failed`	SageMaker Training job encountered an unrecoverable error.
`FAILED`	`Stopped`	`Stopped`	SageMaker Training job was manually stopped using `StopTrainingJob` .

Service job retry strategies in AWS Batch

Service job retry strategies allow AWS Batch to automatically retry failed service jobs under specific conditions.

Service jobs may require multiple attempts for several reasons:

- **Temporary service issues**: Internal service errors, throttling, or temporary outages can cause jobs to fail during submission or execution.
- **Training initialization failures**: Issues during job startup, such as image pulling problems or initialization errors, may be resolved on retry.

By configuring appropriate retry strategies, you can improve job success rates and reduce the need for manual intervention, especially for long-running training workloads.

> **Note**
>
> Service jobs automatically retry certain types of failures, such as insufficient capacity errors, without consuming your configured retry attempts. Your retry strategy primarily handles other types of failures such as algorithm errors or service issues.

Configuring retry strategies

Service job retry strategies are configured using ServiceJobRetryStrategy, which supports both simple retry counts and conditional retry logic.

Retry configuration

The simplest retry strategy specifies the number of retry attempts that should be made if a service job fails:

```
{
  "retryStrategy": {
    "attempts": 3
  }
}
```

This configuration allows the service job to be retried up to 3 times if it fails.

Important

The `attempts` value represents the total number of times the job can be placed in the `RUNNABLE` state, including the initial attempt. A value of 3 means the job will be attempted once initially, then retried up to 2 additional times if it fails.

Retry configuration with evaluateOnExit

You can use the `evaluateOnExit` parameter to specify conditions under which jobs should be retried or allowed to fail. This is useful for when different types of failures require different handling.

The `evaluateOnExit` array can contain up to 5 retry strategies, each specifying an action (`RETRY` or `EXIT`) and conditions based on status reasons:

```
{
  "retryStrategy": {
    "attempts": 5,
    "evaluateOnExit": [
      {
        "action": "RETRY",
        "onStatusReason": "Received status from SageMaker: InternalServerError*"
      },
      {
        "action": "EXIT",
        "onStatusReason": "Received status from SageMaker: ValidationException*"
      },
      {
        "action": "EXIT",
        "onStatusReason": "*"
      }
    ]
  }
}
```

This configuration:

- Retries jobs that fail due to SageMaker AI internal server errors
- Immediately fails jobs that encounter validation exceptions (client errors that won't be resolved by retry)

- Includes a catch-all rule to exit for any other failure types

Status reason pattern matching

The `onStatusReason` parameter supports pattern matching with up to 512 characters. Patterns can use wildcards (*) and match against status reasons returned by SageMaker AI.

For service jobs, status messages from SageMaker AI are prefixed with "Received status from SageMaker: " to distinguish them from AWS Batch-generated messages. Common patterns include:

- `Received status from SageMaker: InternalServerError*` - Match internal service errors
- `Received status from SageMaker: ValidationException*` - Match client validation errors
- `Received status from SageMaker: ResourceLimitExceeded*` - Match resource limit errors
- `*CapacityError*` - Match capacity-related failures

> **Tip**
>
> Use specific pattern matching to handle different error types appropriately. For example, retry internal server errors but immediately fail on validation errors that indicate problems with job parameters.

Monitor service jobs in an AWS Batch queue

You can monitor the status of jobs in your SageMaker Training job queue using `list-service-jobs`, and `get-job-queue-snapshot`.

View running jobs in your queue:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status RUNNING
```

View jobs waiting in the queue:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status RUNNABLE
```

View jobs that have been submitted to SageMaker but not yet running:

```
aws batch list-service-jobs \
  --job-queue my-sm-training-fifo-jq \
  --job-status SCHEDULED
```

Get a snapshot of jobs at the front of your queue:

```
aws batch get-job-queue-snapshot --job-queue my-sm-training-fifo-jq
```

This command shows the order of upcoming service jobs in your queue.

Get detailed information about a specific service job:

```
aws batch describe-service-job --jobId job-id
```

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`.

Note

Array job parents are updated to `PENDING` when any child job is updated to `RUNNABLE` and remain in `PENDING` status while child jobs are running. To view these jobs, filter by `PENDING` status until all child jobs reach a terminal state.

`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.

Note

If your jobs don't progress to `STARTING`, see Jobs stuck in a RUNNABLE status in the troubleshooting section.

`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`.

Image pull duration, Amazon EKS initContainer completion duration, and Amazon ECS containerDependency resolution duration occur in the STARTING state. The amount of time it takes to pull an image for your job is equivalent to the amount of time your job will be in the STARTING state.

For example, if it takes three minutes to pull the image for your job, your job will be in the STARTING state for three minutes. If initContainers takes a total of ten minutes to complete, then your Amazon EKS job will be in STARTING for ten minutes. If you have Amazon ECS containerDependencies sets in your Amazon ECS job, the job will be in STARTING until all container dependencies (their runtime) are resolved. STARTING is not included in timeouts; duration starts at RUNNING. For more information, see Job states.

`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.

> **Note**
>
> Logs for `RUNNING` 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 might 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*.

`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 expires. 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 Use 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": [
    {
        "action": "RETRY",
        "onReason": "AGENT"
    },
    {
        "action": "RETRY",
        "onStatusReason": "Task failed to start"
    },
    {
        "action": "EXIT",
        "onReason": "*"
    }
]
```

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, Create a single-node job definition , Tutorial: submit 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 Tutorial: submit a job.

Resource-aware scheduling lets you schedule jobs based on consumable resources that are needed to run your jobs. You specify the consumable resources which are needed for a job to run, and Batch takes these resource dependencies into account when it schedules a job. You can reduce the underutilization of compute resources by allocating only the jobs that have all the required resources available. Resource-aware scheduling is available for both FIFO and fair-share scheduling policies and can be used with all compute platforms supported by Batch including EKS, ECS, and Fargate. It can be used with Array jobs, Multi-node parallel (MNP) jobs, and with regular Batch jobs.

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 Tutorial: submit 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.

Topics

- Tutorial: Map a running job to a pod and a node
- Tutorial: Map a running pod back to its job

Tutorial: 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.

```
$ 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"
        }
       }
      }
     ],
     "podName": "aws-batch.b0aca953-ba8f-3791-83e2-ed13af39428c"
    }
   }
  }
 ]
}
```

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.

Tutorial: 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-d83fbba0e9f0
               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:
      AWS_BATCH_JOB_KUBERNETES_NODE_UID:   a4be5c1d-9881-4524-b967-587789094647
      AWS_BATCH_JOB_ID:                    f980f2cf-6309-4c77-a2b2-d83fbba0e9f0
      AWS_BATCH_JQ_NAME:                    My-Eks-JQ1
      AWS_BATCH_JOB_ATTEMPT:                1
      AWS_BATCH_CE_NAME:                    My-Eks-CE1

...
```

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
- Use fair-share scheduling to help schedule jobs

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
- Run GPU jobs
- Jobs stuck in a RUNNABLE status

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 (also known as *gang scheduling*), 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, inter-node 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.

Topics

- Environment variables
- Node groups
- Job lifecycle for MNP jobs
- Compute environment considerations for MNP with AWS Batch

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 for MNP jobs

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 for MNP with AWS Batch

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 in the *Amazon EC2 User Guide*.
- 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.
- Your compute environment might have no more than five security groups associated with it. The elastic network interfaces that are created and attached to an MNP task will use the security groups specified in your compute environment, If you don't specify a security group, the default security group for the VPC is used.
- 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 in the *Amazon VPC User Guide*. Inter-node 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 Create a virtual private cloud .
- 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.

Multi-node parallel jobs on Amazon EKS

You can use AWS Batch on Amazon Elastic Kubernetes Service to run multi-node parallel (MNP) jobs (also known as *gang scheduling*) on your managed Kubernetes clusters. This option is commonly used for large, tightly-coupled, high-performance jobs that can't be run on a single Amazon Elastic Compute Cloud instance. For more information, see Multi-node parallel jobs.

You can use this feature to run Amazon EKS managed Kubernetes-specific high-performance computing applications, large language model training, and other Artificial Intelligence (AI)/ Machine Learning (ML) jobs.

Topics

- Running MNP jobs
- Create an Amazon EKS MNP job definition
- Submit an Amazon EKS MNP job
- Override an Amazon EKS MNP job definition

Running MNP jobs

AWS Batch supports MNP jobs on Amazon Elastic Container Service and Amazon EKS using Amazon EC2. The following provides more specifics about the instance and container parameters for the feature.

Instance quotas for MNP on Amazon EKS

- Up to 1000 instances can be used for a single MNP job.
- Up to 5000 instances can join a single Amazon EKS cluster.
- Up to 5 compute environments can be clustered and attached to a job-queue.

For example, you can scale up to 5 clustered compute environments in a job queue and 1000 instances in each compute environment.

In addition to the instance parameters, it's important to note that you can't use Fargate for MNP jobs through either service.

You can use only one instance type in each MNP job. You can change the instance type by updating the compute environment, or when you define a new compute environment. You can also specify the instance type, and provide vCPU and memory requirements when creating the job definition.

Container quotas for MNP on Amazon EKS

- A multi-node parallel job supports one pod per node.
- Up to 10 containers (or 10 init containers. For more information see Init Containers in the *Kubernetes documentation*.) in each pod.
- Up to 5 node ranges in each MNP job.
- Up 10 distinct container images in each node range.

For example, you can run up to a maximum of 10,000 containers in a single MNP job that contains 5 node ranges and a total of 50 unique images.

Running MNP jobs in a private Amazon VPC and an Amazon EKS cluster

MNP jobs can run on any Amazon EKS cluster whether it has public Internet or not. When using an Amazon EKS cluster with only private network access be sure that AWS Batch can access the Amazon EKS control plane and the managed Kubernetes API server. You can grant the necessary access through Amazon Virtual Private Cloud endpoints. For more information, see Configure an endpoint service.

Amazon EKS cluster Pods can’t download an image from a public source since the private VPC doesn’t have Internet access. Your Amazon EKS cluster must pull images from a container registry that's within your Amazon VPC. You can create an Amazon Elastic Container Registry (Amazon ECR) in your Amazon VPC and copy container images to it for your nodes access.

You can also create a pull through cache rule with Amazon ECR. Once a pull through cache rule is created for an external public registry, you can simply pull an image from that external public registry using your Amazon ECR private registry 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 will update your private registry up to one time every 24 hours. For more information, see Creating a pull through cache rule in Amazon ECR.

Error notification

If your MNP jobs are blocked, you can receive notifications through the AWS Management Console and Amazon EventBridge. For example, if an MNP job is stuck at the head of the queue, you can be notified about the issue along with information about what caused it so that you can take prompt action to unblock your job queue. Optionally, you can auto-terminate the MNP job if no action is taken within a distinct amount of time, which can be defined in the job queue template. For more information, see Job queue blocked events

Create an Amazon EKS MNP job definition

To define and run MNP jobs on Amazon EKS, there are new parameters within the `RegisterJobDefinition` and `SubmitJob` API operations.

- Use `eksProperties` under the `nodeProperties` section to define your MNP job definition.
- Use `eksPropertiesOverride` under the `nodePropertyOverrides` section to override the parameters defined in the job definition when submitting an MNP job.

These actions can be defined through API operations and the AWS Management Console.

Reference: Register the Amazon EKS MNP job definition request payload

The following example illustrates how you can register an Amazon EKS MNP job definition with two nodes.

```
{
  "jobDefinitionName": "MyEksMnpJobDefinition",
  "type": "multinode",
  "nodeProperties": {
    "numNodes": 2,
    "mainNode": 0,
    "nodeRangeProperties": [
      {
        "targetNodes" : "0:",
        "eksProperties": {
          "podProperties": {
            "containers": [
              {
                "name": "test-eks-container-1",
                "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                "command": [
                  "sleep",
                  "60"
                ],
                "resources": {
                  "limits": {
                    "cpu": "1",
                    "memory": "1024Mi"
                  }
                },
                "securityContext":{
                  "runAsUser":1000,
                  "runAsGroup":3000,
                  "privileged":true,
                  "readOnlyRootFilesystem":true,
                  "runAsNonRoot":true
               }
              }
            ],
            "initContainers": [
               {
                  "name":"init-ekscontainer",
                  "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                  "command": [
                     "echo",
                     "helloWorld"
                   ],
                   "resources": {
                     "limits": {
                       "cpu": "1",
                       "memory": "1024Mi"
                     }
                  }
               }
            ],
            "metadata": {
               "labels": {
                  "environment" : "test"
               }
            }
          }
        }
      }
    ]
  }
```

```
}
```

To register the job definition using the AWS CLI, copy the definition to a local file named *MyEksMnpJobDefinition.json* and run the following command.

```
aws batch register-job-definition --cli-input-json file://MyEksMnpJobDefinition.json
```

You will receive the following JSON response.

```
{
    "jobDefinitionName": "MyEksMnpJobDefinition",
    "jobDefinitionArn": "arn:aws:batch:us-east-1:0123456789:job-definition/
MyEksMnpJobDefinition:1",
    "revision": 1
}
```

Submit an Amazon EKS MNP job

To submit a job using the registered job definition, enter the following command. Replace the value of <EKS_JOB_QUEUE_NAME> with the name or ARN of a pre-existing job queue associated with an Amazon EKS compute environment.

```
aws batch submit-job --job-queue <EKS_JOB_QUEUE_NAME> \
    --job-definition MyEksMnpJobDefinition \
    --job-name myFirstEksMnpJob
```

You will receive the following JSON response.

```
{
    "jobArn": "arn:aws:batch:region:account:job/9b979cce-9da0-446d-90e2-ffa16d52af68",
    "jobName": "myFirstEksMnpJob",
    "jobId": "<JOB_ID>"
}
```

You can check the status of the job using the returned jobId with the following command.

```
aws batch describe-jobs --jobs <JOB_ID>
```

Override an Amazon EKS MNP job definition

Optionally, you can override the job definition details (such as changing the MNP job size or child job details). The following provides an example JSON request payload to submit a five node MNP job, and changes to the `test-eks-container-1` container's command.

```
{
  "numNodes": 5,
  "nodePropertyOverrides": [
    {
      "targetNodes": "0:",
      "eksPropertiesOverride": {
        "podProperties": {
          "containers": [
            {
              "name": "test-eks-container-1",
              "command": [
                "sleep",
                "150"
              ]
            }
          ]
        }
      }
    }
  ]
}
```

To submit a job with these overrides, save the example to a local file, *eks-mnp-job-nodeoverride.json*, and use the AWS CLI to submit the job with the overrides.

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.

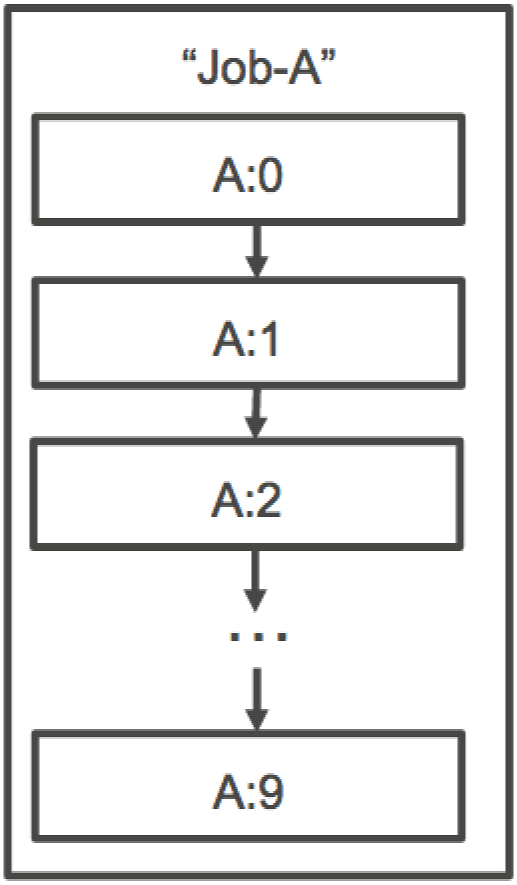
AWS Batch array jobs are submitted just like regular jobs. However, you specify an array size (between 2 and 10,000) to define how many child jobs should run in the array. If you submit a job with an array size of 1000, a single job runs and spawns 1000 child jobs. The array job is a reference or pointer 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.

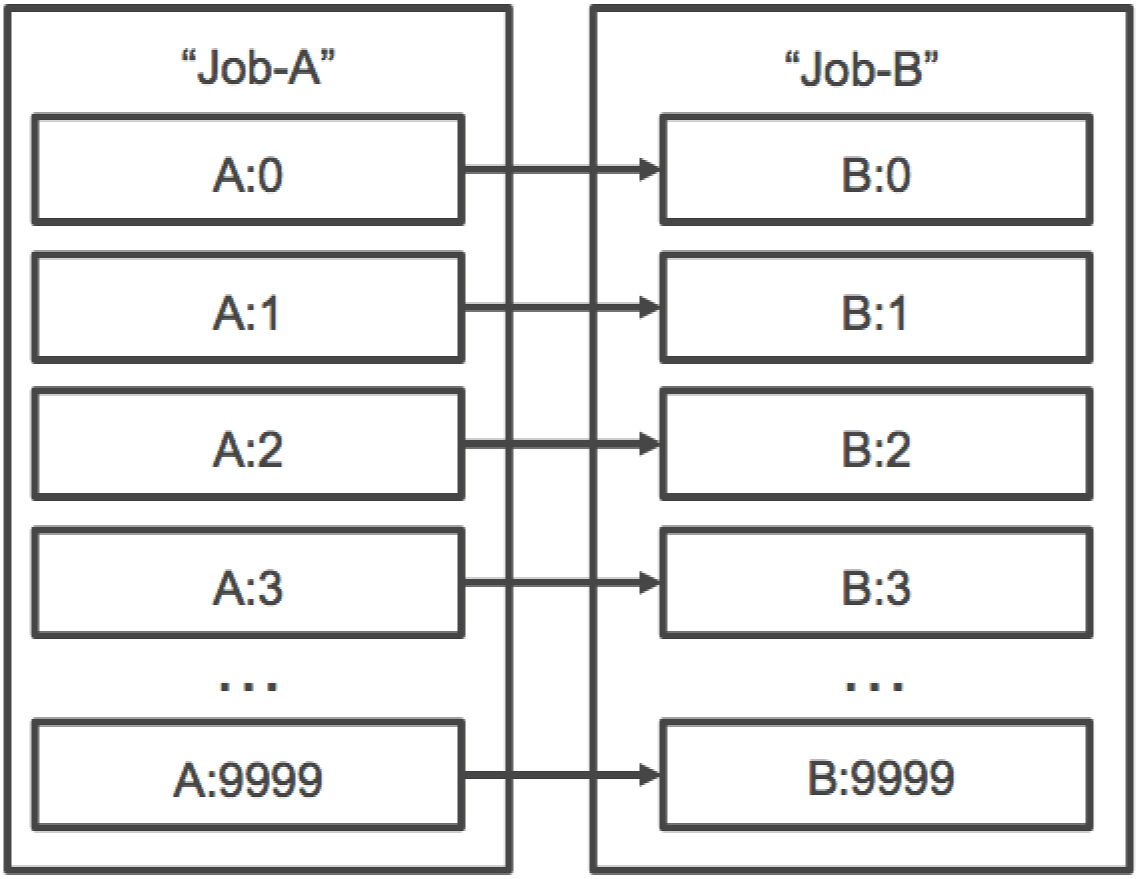
The parent array job can enter a `SUBMITTED`, `PENDING`, `FAILED`, or `SUCCEEDED` status. An array parent job is updated to `PENDING` when any child job is updated to `RUNNABLE`. For more information about job dependencies, see Job dependencies.

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 Use 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 the child jobs are cancelled or terminated with it. You can cancel or terminate individual child jobs (which moves them to a `FAILED` status) without affecting the other child jobs. However, if a child array job fails (on its own, or by manually cancelling or terminating the job), the parent job also fails. In this scenario, the parent job transitions to `FAILED` when all child jobs complete.

For more information about searching and filtering array jobs, see Search for jobs in a job queue.

Topics

- Example of an array job workflow
- Use the array job index to control job differentiation

Example of an 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 JSON syntax is as follows.

```
{
    "jobName": "JobA",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobA-list-and-validate:1"
}
```

- `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.

```
{
    "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.

```
{
    "jobName": "JobC",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobC-Memory-Intensive-Processing:1",
    "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.

```
{
    "jobName": "JobD",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobD-Sequential-Validation:1",
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "32768"
            },
            {
                "type": "VCPU",
                "value": "1"
            }
        ]
   }
    "arrayProperties": {
        "size": 10
    },
    "dependsOn": [
        {
            "jobId": "JobC_job_ID"
        },
        {
            "type": "SEQUENTIAL"
        },

    ]
}
```

- `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.

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

Use 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.

Topics

- Prerequisites
- Build a container image
- Push your image to Amazon ECR
- Create and register a job definition
- Submit an AWS Batch array job
- View your array job logs

Prerequisites

This tutorial workflow has the following prerequisites:

- An AWS Batch compute environment. For more information, see Create a compute environment.
- An AWS Batch job queue and associated compute environment. For more information, see Create 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.

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
yellow
green
blue
indigo
violet
```

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

> **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.

```
#!/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.
```

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.

```
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.

```
$ 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.

```
$ 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.
```

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
```

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 Create 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.

```
{
  "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
```

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.

> **Note**
>
> This example uses the job queue mentioned in the the section called "Prerequisites" section.

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

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
```

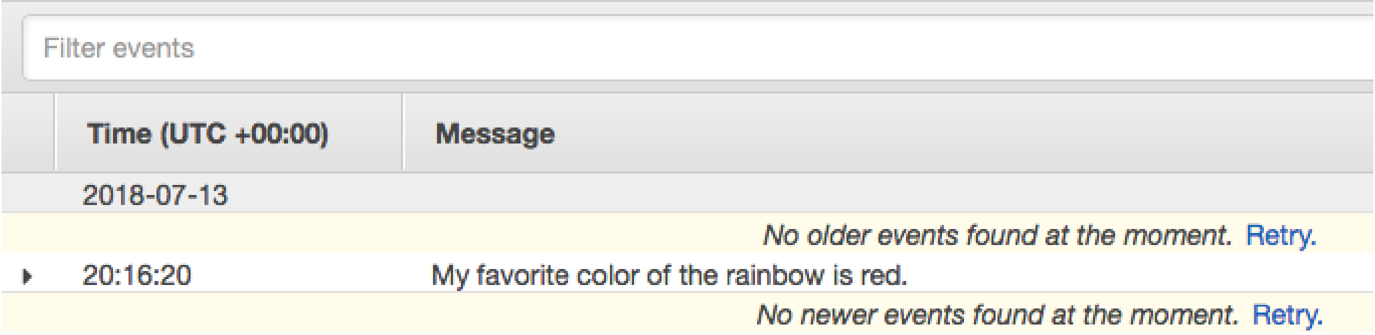
3. Describe the job's status and wait for the job to move to SUCCEEDED.

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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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**.



Time (UTC +00:00)	Message
2018-07-13	
	No older events found at the moment. Retry.
20:16:20	My favorite color of the rainbow is red.
	No newer events found at the moment. Retry.

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

Run 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 G6 Instances , Amazon EC2 P2 Instances, Amazon EC2 P3 Instances, Amazon EC2 P4d Instances, Amazon EC2 P5 Instances, Amazon EC2 P6 Instances, Amazon EC2 Trn1 Instances, Amazon EC2 Trn2 Instances, Amazon EC2 Inf1 Instances, Amazon EC2 Inf2 Instances, Amazon EC2 Dl1 Instances, and Amazon EC2 Dl2 Instances.

Instance type	GPUs	GPU memory	vCPUs	Memory	Network bandwidth
g3s.xlarge	1	8 GiB	4	30.5 GiB	10 Gbps
g3.4xlarge	1	8 GiB	16	122 GiB	Up to 10 Gbps
g3.8xlarge	2	16 GiB	32	244 GiB	10 Gbps
g3.16xlarge	4	32 GiB	64	488 GiB	25 Gbps
g4dn.xlarge	1	16 GiB	4	16 GiB	Up to 25 Gbps
g4dn.2xlarge	1	16 GiB	8	32 GiB	Up to 25 Gbps
g4dn.4xlarge	1	16 GiB	16	64 GiB	Up to 25 Gbps
g4dn.8xlarge	1	16 GiB	32	128 GiB	50 Gbps
g4dn.12xlarge	4	64 GiB	48	192 GiB	50 Gbps
g4dn.16xlarge	1	16 GiB	64	256 GiB	50 Gbps
g5.xlarge	1	24 GiB	4	16 GiB	Up to 10 Gbps
g5.2xlarge	1	24 GiB	8	32 GiB	Up to 10 Gbps
g5.4xlarge	1	24 GiB	16	64 GiB	Up to 25 Gbps
g5.8xlarge	1	24 GiB	32	128 GiB	25 Gbps
g5.16xlarge	1	24 GiB	64	256 GiB	25 Gbps
g5.12xlarge	4	96 GiB	48	192 GiB	40 Gbps

Instance type	GPUs	GPU memory	vCPUs	Memory	Network bandwidth
g5.24xlarge	4	96 GiB	96	384 GiB	50 Gbps
g5.48xlarge	8	192 GiB	192	768 GiB	100 Gbps
g5g.xlarge	1	16 GiB	4	8 GiB	Up to 10 Gbps
g5g.2xlarge	1	16 GiB	8	16 GiB	Up to 10 Gbps
g5g.4xlarge	1	16 GiB	16	32 GiB	Up to 10 Gbps
g5g.8xlarge	1	16 GiB	32	64 GiB	12 Gbps
g5g.16xlarge	2	32 GiB	64	128 GiB	25 Gbps
g5g.metal	2	32 GiB	64	128 GiB	25 Gbps
g6.xlarge	1	24 GiB	4	16 GiB	Up to 10 Gbps
g6.2xlarge	1	24 GiB	8	32 GiB	Up to 10 Gbps
g6.4xlarge	1	24 GiB	16	64 GiB	Up to 25 Gbps
g6.8xlarge	1	24 GiB	32	128 GiB	25 Gbps
g6.16xlarge	1	24 GiB	64	256 GiB	25 Gbps
g6.12xlarge	4	96 GiB	48	192 GiB	40 Gbps
g6.24xlarge	4	96 GiB	96	384 GiB	50 Gbps
g6.48xlarge	8	192 GiB	192	768 GiB	100 Gbps
g6e.xlarge	1	48 GiB	4	32 GiB	Up to 20 Gbps
g6e.2xlarge	1	48 GiB	8	64 GiB	Up to 20 Gbps
g6e.4xlarge	1	48 GiB	16	128 GiB	20 Gbps
g6e.8xlarge	1	48 GiB	32	256 GiB	25 Gbps

Instance type	GPUs	GPU memory	vCPUs	Memory	Network bandwidth
g6e.16xlarge	1	48 GiB	64	512 GiB	35 Gbps
g6e.12xlarge	4	192 GiB	48	384 GiB	100 Gbps
g6e.24xlarge	4	192 GiB	96	768 GiB	200 Gbps
g6e.48xlarge	8	384 GiB	192	1536 GiB	400 Gbps
gr6.4xlarge	1	24 GiB	16	128 GiB	Up to 25 Gbps
gr6.8xlarge	1	24 GiB	32	256 GiB	25 Gbps
p2.xlarge	1	12 GiB	4	61 GiB	High
p2.8xlarge	8	96 GiB	32	488 GiB	10 Gbps
p2.16xlarge	16	192 GiB	64	732 GiB	20 Gbps
p3.2xlarge	1	16 GiB	8	61 GiB	Up to 10 Gbps
p3.8xlarge	4	64 GiB	32	244 GiB	10 Gbps
p3.16xlarge	8	128 GiB	64	488 GiB	25 Gbps
p3dn.24xlarge	8	256 GiB	96	768 GiB	100 Gbps
p4d.24xlarge	8	320 GiB	96	1152 GiB	400 Gbps
p4de.24xlarge	8	640 GiB	96	1152 GiB	400 Gbps
p5.48xlarge	8	640 GiB	192	2 TiB	3200 Gbps
p5e.48xlarge	8	1128 GiB	192	2 TiB	3200 Gbps
p5en.48xlarge	8	1128 GiB	192	2 TiB	3200 Gbps
p6-b200.48xlarge	8	1440 GiB	192	2 TiB	100 Gbps

Instance type	GPUs	GPU memory	vCPUs	Memory	Network bandwidth
trn1.2xlarge	1	32 GiB	8	32 GiB	Up to 12.5 Gbps
trn1.32xlarge	16	512 GiB	128	512 GiB	800 Gbps
trn1n.32xlarge	16	512 GiB	128	512 GiB	1600 Gbps
trn2.48xlarge	16	1.5 TiB	192	2 TiB	3.2 Tbps
inf1.xlarge	1	8 GiB	4	8 GiB	Up to 25 Gbps
inf1.2xlarge	1	8 GiB	8	16 GiB	Up to 25 Gbps
inf1.6xlarge	4	32 GiB	24	48 GiB	25 Gbps
inf1.24xlarge	16	128 GiB	96	192 GiB	100 Gbps
inf2.xlarge	1	32 GiB	4	16 GiB	Up to 15 Gbps
inf2.8xlarge	1	32 GiB	32	128 GiB	Up to 25 Gbps
inf2.24xlarge	6	192 GiB	96	384 GiB	50 Gbps
inf2.48xlarge	12	384 GiB	192	768 GiB	100 Gbps
dl1.24xlarge	8	256 GiB	96	768 GiB	400 Gbps
dl2q.24xlarge	8	128 GiB	96	768 GiB	100 Gbps

Note

For GPU jobs AWS Batch only supports instance types that have NVIDIA GPUs. For example, the G4ad family is not supported for GPU scheduling. You can still use G4ad on AWS Batch by defining only the vcpu and memory requirements in the job definition, then accessing the host GPUs directly through customization in an Amazon EC2 launch template user data with an Amazon ECS or Amazon EKS compute optimized AMI, or a customized AMI for using AMD GPUs.

Instance types that use an ARM64 architecture are supported for GPU jobs on custom AMIs provided to AWS Batch or Amazon EC2 user data to access the GPUs by customized code and configurations. For example, the G5g instance family.

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 `p3`, `p4`, `p5`, `p6`, `g3`, `g3s`, `g4`, `g5`, or `g6` 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.

Topics

- Create a GPU-based Kubernetes cluster on Amazon EKS
- Create an Amazon EKS GPU job definition
- Run a GPU job in your Amazon EKS cluster

Create a 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
{
  "computeEnvironmentName": "My-Eks-GPU-CE1",
  "type": "MANAGED",
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:<region>:<account>:cluster/<cluster-name>",
    "kubernetesNamespace": "my-aws-batch-namespace"
  },
  "computeResources": {
    "type": "EC2",
    "allocationStrategy": "BEST_FIT_PROGRESSIVE",
    "minvCpus": 0,
    "maxvCpus": 1024,
    "instanceTypes": [
      "p3dn.24xlarge",
      "p4d.24xlarge"
    ],
    "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-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 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"
  operator: "Exists"

$ kubectl apply -f nvidia-device-plugin.yml
```

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.

```
$ 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
```

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.

```
$ cat <<EOF > ./batch-eks-gpu-jd.json
{
    "jobDefinitionName": "MyGPUJobOnEks_Smi",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "hostNetwork": true,
            "containers": [
                {
                    "image": "nvcr.io/nvidia/cuda:10.2-runtime-centos7",
                    "command": ["nvidia-smi"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi",
                            "nvidia.com/gpu": "1"
                        }
                    }
                }
            ]
        }
    }
}
EOF

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

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.

```
$ 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}'
{
  "podName": "aws-batch.f3d697c4-3bb5-3955-aa6c-977fcf1cb0ca",
  "nodeName": "ip-192-168-59-101.ec2.internal"
}
```

View AWS Batch jobs in a job queue

You can view and filter your jobs in AWS Batch. This feature provides an option to view an existing job queue and filter its jobs by one of three options.

Search and filter are able to retrieve jobs that are not in a terminal state (`SUCCEEDED` or `FAILED`). Once a job's state is `SUCCEEDED` or `FAILED` you should be able to retrieve the job for up to seven days. You are still able to view a job's CloudWatch or Amazon EventBridge logs.

Use this procedure to list all the jobs in a job queue in the AWS Batch console. Optionally, use the **Filter results** field to narrow the results based on the criteria you specify.

1. Navigate to the AWS Batch console.
2. In the Navigation pane, choose **Jobs**.
3. Expand the **Job queue** dropdown list and choose the job queue that you want to search within.

    > **Note**
    >
    > You can search for jobs within only one job queue at a time.

4. In the **Filter results** field, enter keywords to include in the search results. You can use this field to search by **Job name**, **Status**, or **Job ID**. Depending on the property, there may be additional operators, such as equals (=) or contains (:) that you must define.

    > **Note**
    >
    > SageMaker Training job queues only support filtering by **Job name** and **Job ID**

5. Choose **Search**.

Search AWS Batch for jobs in a job queue

You can search and filter your jobs in AWS Batch using Job search. This feature provides an option to search within an existing job queue and filter its jobs.

Search and filter are able to retrieve jobs that are not in a terminal state (`SUCCEEDED` or `FAILED`). Once a job's state is `SUCCEEDED` or `FAILED` you should be able to retrieve the job for up to seven days. You are still able to view a job's CloudWatch or Amazon EventBridge logs.

To search using multiple criteria simultaneously, use the **Advanced search** feature. For example, you can include any or all of the following filters: **Status**, **Date range**, and **Additional criteria** (such as, a job name, job definition, or job ID).

Search AWS Batch jobs (AWS Console)

Use this procedure to search the jobs in a job queue in the AWS Batch console.

1. Navigate to the AWS Batch console.

2. In the Navigation pane, choose **Jobs**.
3. Turn on **Advanced search**.
4. Expand the **Job queue** dropdown list and choose the job queue that you want to search within.

   > **Note**
   >
   > You can search for jobs within only one job queue at a time.

5. For **Search options**:

   a. For the **Status** dropdown list you can choose one or more statuses to filter on. For more information, see Job states and Service job status.

   > **Note**
   >
   > Array job parents are updated to `PENDING` when any child job is updated to `RUNNABLE` and remain in `PENDING` status while child jobs are running. To view these jobs, filter by `PENDING` status until all child jobs reach a terminal state.

   b. Choose **Date range** to filter the results based on a date and time range.

   - Choose **Relative mode** to search for jobs that have a created date within a time range counting backwards from the current date and time.
   - Choose **Absolute mode** to search for jobs that have a created date within a date and time range that you specify.

   c. In the **Additional criteria** field, enter keywords to include in the search results. For example, you can use this field to search by **Job name**, **Job definition**, or **Job ID**. Depending on the property, there may be additional operators, such as equals (=) or contains (:) that you must define.

   > **Note**
   >
   > SageMaker Training job queues only support filtering by **Job name** and **Job ID**

6. Choose **Search**.

Search and filter AWS Batch jobs (AWS CLI)

Use this procedure to list all the jobs in a job queue with the AWS CLI. Optionally, use the **-filters** parameter to narrow the results based on the criteria you specify.

Search job queue (AWS CLI)

You can use the list-jobs command to search and filter a job queue.

For example you can search a job queue based on the job name:

```
aws batch list-jobs \
    --job-queue my-job-queue \
    --filters name=JOB_NAME,values="my-job"
```

In the preceding command, make the following changes:

- Replace *`my-job-queue`* with the name of your job queue.
- Replace *`my-job`* with the name of your job.

Search service job queue (AWS CLI)

You can use the list-service-jobs command to search and filter a service job queue.

For example you can search a service job queue based on the job name:

```
aws batch list-service-jobs \
    --job-queue my-sm-queue \
    --filters name=JOB_NAME,values="my-sm-job"
```

In the preceding command, make the following changes:

- Replace *`my-sm-queue`* with the name of your service job queue.
- Replace *`my-sm-job`* with the name of your service job.

Networking modes for AWS Batch jobs

The following table describes the networking modes and typical usage for AWS Batch job types. See the links in the "Job type" column for more details regarding considerations and behaviors.

Job Type	Supported Network Mode(s)	Typical Usage
ECS-EC2 simple job	`host`	Used for the highest scalable embarrassingly parallel batch workloads that only require egress to vpc defined in Compute Environment.
ECS-EC2 multi-node parallel job	`awsvpc`	Used for tightly-coupled, multi-host (node) distributed workloads modeled as a single job with coordinated communications between task nodes.
ECS-Fargate simple job	`awsvpc`	True serverless for embarrassingly parallel batch workloads. Typically lowest TCO and highest container isolation job model.
EKS-EC2 simple job	host and pod	Used for highly scalable embarrassingly parallel batch workloads that only require egress to vpc defined in Compute Environment. Default is host networking.
EKS-EC2 multi-node parallel job	host and pod	Used for tightly-coupled, multi-host (node) distributed workloads modeled as a single job with coordinated communications between pod nodes. Default is host networking.

View AWS Batch job logs in CloudWatch Logs

You can configure your AWS Batch jobs to send log information to Amazon 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.

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**. For more detailed information on sorting and filtering job in the job queue, see View AWS Batch jobs in a job queue and Search for jobs in a job queue
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 AWS Batch for jobs in a job queue.

4. For **Status**, choose the job status that you want.
5. Choose the job that you want and the **Details** page will open.
6. On the **Details** page, scroll down to **Log stream name** and choose the link. The link opens the Amazon CloudWatch Logs page for the job.
7. (Optional) If this is the first time you've viewed the logs you may be asked for authorization.

   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**.

Review AWS Batch job information

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

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 **Search and filter** to find a job faster. For more information, see Search AWS Batch for jobs in a job queue.

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 in the AWS CLI Command Reference.

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
- AWS Batch IAM policies, roles, and permissions
- AWS Batch IAM execution role
- Create a virtual private cloud
- Use an interface endpoint to Access AWS Batch
- Compliance validation for AWS Batch
- Infrastructure security in AWS Batch
- Cross-service confused deputy prevention
- Logging AWS Batch API calls with AWS CloudTrail
- Troubleshoot AWS Batch IAM

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
- Identity-based policy examples for AWS Batch
- AWS managed policies for AWS Batch

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshoot AWS Batch IAM)
- **Service administrator** - determine user access and submit permission requests (see How AWS Batch works with IAM)
- **IAM administrator** - write policies to manage access (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 as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM 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 defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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 include 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. You must specify a principal in a resource-based policy.

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.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries –** Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs) –** Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies –** Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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

IAM feature	AWS Batch support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No

IAM feature	AWS Batch support
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	Yes

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 Define custom IAM permissions with customer managed 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. 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

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. 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",
      "batch:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "batch:Describe*"
```

To view examples of AWS Batch identity-based policies, see Identity-based policy examples for AWS Batch.

Policy resources for AWS Batch

Supports policy resources: 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 `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, 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.

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 specifies when statements execute based on defined criteria. 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. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS Batch condition keys, see Condition Keys for AWS Batch in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Batch.

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 called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Use temporary credentials with AWS Batch

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS Batch

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. 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 Create 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.

Identity-based policy examples for AWS Batch

By default, users and roles don't have permission to create or modify AWS Batch resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) 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
- Using the AWS Batch console
- Allow users to view their own permissions

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 or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more

information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer 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 Secure API access with MFA 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",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

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.

To view the JSON for the policy, see BatchServiceRolePolicy in the *AWS managed policies Reference Guide*.

AWS managed policy: AWSBatchServiceRolePolicyForSageMaker

`AWSServiceRoleForAWSBatchWithSagemaker` 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:

- `sagemaker` – Allows AWS Batch to manage SageMaker AI training jobs and other SageMaker AI resources.
- `iam:PassRole` – Allows AWS Batch to pass customer-defined execution roles to SageMaker AI for job execution. The resource constraint allows passing roles to SageMaker AI services.

To view the JSON for the policy, see AWSBatchServiceRolePolicyForSageMaker in the *AWS managed policies Reference Guide*.

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.

To view the JSON for the policy, see AWSBatchServiceRole in the *AWS managed policies Reference Guide*.

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.

To view the JSON for the policy, see AWSBatchFullAccess in the *AWS managed policies Reference Guide*.

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.

Change	Description	Date
AWSBatchServiceRolePolicyForSageMaker policy added	Added new AWS managed policy for the **AWSBatchServiceRolePolicyForSageMaker** service-linked role that allows AWS Batch to manage SageMaker AI on your behalf.	July 31, 2025
BatchServiceRolePolicy policy updated	Updated to add support for describing Spot Fleet request history and Amazon EC2 Auto Scaling activities.	December 5, 2023
AWSBatchServiceRole policy added	Updated to add statement IDs, grant AWS Batch permissions to `ec2:DescribeSpotFleetRequestHistory` and `autoscaling:DescribeScalingActivities` .	December 5, 2023
BatchServiceRolePolicy policy updated	Updated to add support for describing Amazon EKS clusters.	October 20, 2022
AWSBatchFullAccess policy updated	Updated to add support for listing and describing Amazon EKS clusters.	October 20, 2022
BatchServiceRolePolicy policy updated	Updated to add support for Amazon EC2 Capacity Reservation groups that are managed by AWS Resource Groups. For more information, see Work with Capacity Reservation groups in *Amazon EC2 User Guide*.	May 18, 2022
BatchServiceRolePolicy and AWSBatchServiceRole policies updated	Updated to add support for describing the status of AWS Batch managed instances in Amazon EC2 so that unhealthy instances are replaced.	December 6, 2021

Change	Description	Date
BatchServiceRolePolicy policy updated	Updated to add support for placement group, capacity reservation, elastic GPU, and Elastic Inference resources in Amazon EC2.	March 26, 2021
BatchServiceRolePolicy policy added	With the **BatchServiceRolePolicy** managed policy for the **AWSServiceRoleForBatch** 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.	March 10, 2021
AWSBatchFullAccess - add permission to add service-linked role	Add IAM permissions to allow the **AWSServiceRoleForBatch** service-linked role to be added to the account.	March 10, 2021
AWS Batch started tracking changes	AWS Batch started tracking changes for its AWS managed policies.	March 10, 2021

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*.

Topics

- IAM policy structure
- Resource: Example policies for AWS Batch
- Resource: AWS Batch managed policy

IAM policy structure

The following topics explain the structure of an IAM policy.

Topics

- Policy syntax
- API actions for AWS Batch
- Amazon Resource Names for AWS Batch
- Confirm 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.

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
    "key":"value"
    }
      }
    }
  ]
}
```

There are four primary 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 API 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 Resource: Example policies for AWS Batch.

API 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": "*"
```

Confirm 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*.

Resource: Example policies for AWS Batch

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.

The following examples show policy statements that you can use to control the permissions that users have for AWS Batch.

Examples

- Resource: Read-only access for AWS Batch
- Resource: Restrict to POSIX user, Docker image, privilege level, and role on job submission

- Resource: Restrict to job definition prefix on job submission
- Resource: Restrict to a job queue
- Deny action when all conditions match strings
- Resource: Deny action when any condition keys match strings
- Resource: Use the batch:ShareIdentifier condition key
- Manage SageMaker AI resources with AWS Batch
- Resource: Restrict job submission by resource tags on job definition and job queue

Resource: Read-only access for AWS Batch

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:Describe*",
                "batch:List*",
                "batch:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

Resource: Restrict 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": [
                "arn:aws:batch:us-east-2:999999999999:job-definition/JobDefA_*"
            ],
            "Condition": {
                "StringEquals": {
                    "batch:User": [
                        "nobody"
                    ],
                    "batch:Image": [
                        "999999999999.dkr.ecr.us-east-2.amazonaws.com/myImage"
                    ]
                },
                "Bool": {
                    "batch:Privileged": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "batch:DeregisterJobDefinition"
            ],
            "Resource": [
                "arn:aws:batch:us-east-2:999999999999:job-definition/JobDefA_*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::999999999999:role/MyBatchJobRole"
            ]
        }
    ]
}
```

Resource: 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": [
                "arn:aws:batch:us-east-2:111122223333:job-definition/JobDefA_*",
                "arn:aws:batch:us-east-2:111122223333:job-queue/*"
            ]
        }
    ]
}
```

Resource: Restrict to a job queue

Use the following policy to submit jobs to a specific job queue that's named **queue1** with any job definition name.

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:us-east-2:888888888888:job-definition/*",
                "arn:aws:batch:us-east-2:888888888888:job-queue/queue1"
            ]
        }
    ]
}
```

Deny action when all conditions 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": "Allow",
      "Action": [
        "batch:RegisterJobDefinition"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": "batch:RegisterJobDefinition",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "batch:Image": "string1",
          "batch:LogDriver": "string2"
        }
      }
    }
  ]
}
```

Resource: 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 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": "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"
          ]
        }
      }
    }
  ]
}
```

Resource: 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:us-east-2:555555555555:job-definition/JobDefA",
        "arn:aws:batch:us-east-2:555555555555:job-queue/jobqueue1"
      ],
      "Condition": {
        "StringEquals": {
          "batch:ShareIdentifier": [
            "lowCpu"
          ]
        }
      }
    }
  ]
}
```

Manage SageMaker AI resources with AWS Batch

This policy allows AWS Batch to manage SageMaker AI resources.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/
*AWSServiceRoleForAWSBatchWithSagemaker",
            "Condition": {
                 "StringEquals": {
                     "iam:AWSServiceName": "sagemaker-
queuing.batch.amazonaws.com"
                 }
             }
         },
         {
             "Effect": "Allow",
             "Action": "iam:PassRole",
             "Resource": "*",
             "Condition": {
                 "StringEquals": {
                     "iam:PassedToService": [
                         "sagemaker.amazonaws.com"
                      ]
                  }
              }
          }
    ]
}
```

Resource: Restrict job submission by resource tags on job definition and job queue

Use the following policy to submit jobs only when both the job queue has the tag `Environment=dev` and the job definition has the tag `Project=calc`. This policy demonstrates how to use resource tags to control access to AWS Batch resources during job submission.

> **Important**
>
> When submitting jobs with policies that evaluate job definition resource tags, you must submit jobs using the job definition revision format (`job-definition:revision`). If you submit jobs without specifying a revision, job definition tags will not be evaluated,

potentially bypassing your intended access controls. The `*:*` pattern in the resource ARN enforces that submissions must include a revision, ensuring tag policies are always effectively applied.

This policy uses two separate statements because it applies different tag conditions to different resource types. When scoping resource-level access for job submission, you must provide both job queue and job definition resource types.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "batch:SubmitJob",
      "Resource": "arn:aws:batch:*:*:job-queue/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "dev"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "batch:SubmitJob",
      "Resource": "arn:aws:batch:*:*:job-definition/*:*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "calc"
        }
      }
    }
  ]
}
```

Resource: AWS Batch managed policy

AWS Batch provides a managed policy that you can attach to users. This policy 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.

To view the JSON for the policy, see AWSBatchFullAccess in the *AWS managed policies Reference Guide*.

AWS Batch IAM execution 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 IAM execution 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"
      ],
      "Resource": "*"
    }
  ]
}
```

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.

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*.

Tutorial: Create the IAM execution role

If your account doesn't already have an IAM execution role, use the following steps to create the 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 **Elastic Container Service**. Then choose **Elastic Container Service Task** 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**.

Tutorial: Check the IAM execution role

Use the following procedure to check that your account already has the IAM execution role and attach the managed IAM policy, if needed.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsTaskExecutionRole`. If you can't find the role, see Tutorial: Create the IAM execution 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"
    }
  ]
}
```

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.

AWS Batch uses two different service-linked roles:

- AWSServiceRoleForBatch - For AWS Batch operations including compute environments.
- AWSServiceRoleForAWSBatchWithSagemaker - For SageMaker AI workload management and queuing.

Topics

- Using roles for AWS Batch
- Using roles for AWS Batch with SageMaker AI

Using 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 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 roles** 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** – Allows AWS Batch to create and manage AWS resources on your behalf.

The AWSServiceRoleForBatch service-linked role trusts the following services to assume the role:

- `batch.amazonaws.com`

The role permissions policy named BatchServiceRolePolicy allows AWS Batch to complete the following actions on the specified 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 your users, groups, or roles 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 create a compute environment in the AWS Management Console, the AWS CLI, or the AWS API, 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. 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 AWS 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 create a compute environment, AWS Batch creates the service-linked role for you again.

Editing a service-linked role for AWS Batch

AWS Batch does not allow you to edit the AWSServiceRoleForBatch service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

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.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/batch.amazonaws.com/
AWSServiceRoleForBatch",
    "Condition": {"StringLike": {"iam:AWSServiceName": "batch.amazonaws.com"}}
}
```

Deleting a service-linked role for AWS Batch

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up 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

1. Open the IAM console at https://console.aws.amazon.com/iam/.
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.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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/.
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:

   ```
   $ aws iam delete-service-linked-role --role-name AWSServiceRoleForBatch
   ```

2. Use the following command to check the status of the deletion task:

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   > **Note**
   >
   > You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them. Or, it might not report any resources. To learn how to clean up the resources for a service that 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 roll, 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.

Using roles for AWS Batch with SageMaker AI

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.

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 roles** 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 **AWSServiceRoleForAWSBatchWithSagemaker** – Allows AWS Batch to queue and manage SageMaker Training jobs on your behalf.

The AWSServiceRoleForAWSBatchWithSagemaker service-linked role trusts the following services to assume the role:

- `sagemaker-queuing.batch.amazonaws.com`

The role permissions policy allows AWS Batch to complete the following actions on the specified resources:

- `sagemaker` – Allows AWS Batch to manage SageMaker training jobs, transform jobs, and other SageMaker AI resources.
- `iam:PassRole` – Allows AWS Batch to pass customer-defined execution roles to SageMaker AI for job execution. The resource constraint allows passing roles to SageMaker AI services.

You must configure permissions to allow your users, groups, or roles 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 create a service environment using `CreateServiceEnvironment` in the AWS Management Console, the AWS CLI, or the AWS API, AWS Batch creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a service environment using `CreateServiceEnvironment`, AWS Batch creates the service-linked role for you again.

To view the JSON for the policy, see AWSBatchServiceRolePolicyForSageMaker in the *AWS managed policies Reference Guide*.

Editing a service-linked role for AWS Batch

AWS Batch does not allow you to edit the AWSServiceRoleForAWSBatchWithSagemaker service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS Batch

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

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 service environments that use the role in all AWS Regions in a single partition.

To check whether the service-linked role has an active session

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and then the AWSServiceRoleForAWSBatchWithSagemaker 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 AWSServiceRoleForAWSBatchWithSagemaker 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 AWSServiceRoleForAWSBatchWithSagemaker service-linked role

You must dissociate all job queue's from all service environments then you must delete all service environments that use the AWSServiceRoleForAWSBatchWithSagemaker role in all AWS Regions before you can delete the AWSServiceRoleForAWSBatchWithSagemaker role.

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Environments**, and then **Service environments**.
4. Select all **Service environments**.
5. Choose **Disable**. Wait for the **State** to change to **DISABLED**.
6. Select the service environment.
7. Choose **Delete**. Confirm that you want to delete the service environment by choosing **Delete service environment**.
8. Repeat steps 1–7 for all service 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/.
2. In the navigation pane of the IAM console, choose **Roles**. Then select the check box next to AWSServiceRoleForAWSBatchWithSagemaker, 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:

   ```
   $ aws iam delete-service-linked-role --role-name
    AWSServiceRoleForAWSBatchWithSagemaker
   ```

2. Use the following command to check the status of the deletion task:

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-
   id
   ```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   > **Note**
   >
   > You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them. Or, it might not report any resources. To learn how to clean up the resources for a service that 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 roll, call DeleteServiceLinkedRole. In the request, specify the AWSServiceRoleForAWSBatchWithSagemaker 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.

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 instance role in the *Amazon Elastic Container Service Developer Guide*.

Topics

- Check your account's Amazon ECS instance role

Check your account's Amazon ECS instance role

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.

Tutorial: Check for the `ecsInstanceRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsInstanceRole`. If the role 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.

Tutorial: 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",
         "Principal": { "Service": "ec2.amazonaws.com"},
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

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
   ```

3. Use the create-instance-profile command to create an instance profile that's named `ecsInstanceRole`.

> **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.

```
{
    "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.

```
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.

```
$ 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.

```
$ 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 \
    arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \
  --role-name \
    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.

JSON

```
{
  "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": "*"
    }
  ]
}
```

Create 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.

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

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

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 Create 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 Create a job queue.
- Create a job definition to run your jobs with. For more information, see Create 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 Tutorial: submit a job.

Use an interface endpoint to Access AWS Batch

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 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 names:

- **com.amazonaws.*`region`*.batch**
- **com.amazonaws.*`region`*.batch-fips** *(For FIPS-compliant endpoints, see AWS Batch endpoints and quotas)*

For example:

```
com.amazonaws.us-east-2.batch
```

```
com.amazonaws.us-east-2.batch-fips
```

In the `aws-cn` partition, the format is different:

```
cn.com.amazonaws.region.batch
```

For example:

```
cn.com.amazonaws.cn-northwest-1.batch
```

Private DNS names for AWS Batch interface endpoints

If you enable private DNS for the interface endpoint, you can use specific DNS names to connect to AWS Batch, We provide these options:

- **batch.*region* .amazonaws.com**
- **batch.*region* .api.aws**

For FIPS-compliant endpoints:

- **batch-fips.*region* .api.aws**
- **fips.batch.*region* .amazonaws.com** *is not supported*

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.

```
{
   "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. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

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.

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.

Cross-service confused deputy prevention

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:servicename:*: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: 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:compute-environment/testCE",
            "arn:aws:batch:us-east-1:123456789012:job/*"
          ]
        }
      }
    }
  ]
}
```

Example: 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.

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:compute-environment/*",
            "arn:aws:batch:us-east-1:123456789012:job/*"
          ]
        }
      }
    }
  ]
}
```

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.

Topics

- AWS Batch information in CloudTrail
- Reference: Understanding AWS Batch log file entries

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.

Reference: 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.

```
{
  "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"
}
```

Troubleshoot AWS Batch IAM

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 Cross account resource access in IAM in the *IAM User Guide*.

AWS Step Functions

You can use the AWS Batch console to view details about your Step Functions state machines and the functions that they use.

Sections

- Tutorial: View state machine details
- Tutorial: Edit a state machine
- Tutorial: Run a state machine

Tutorial: View 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.
   <result>

   The AWS Batch console opens the **Details** page.
   </result>

For more information, see Step Functions in the *AWS Step Functions Developer Guide*.

Tutorial: Edit 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*.

Tutorial: Run 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 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
- Tutorial: Use AWS user notifications with AWS Batch
- AWS Batch jobs as EventBridge targets
- Tutorial: Listen for AWS Batch job events using 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 programmatically 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*.

```
{
    "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": [],
        "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/first-
run-job-definition:1",
        "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": []
        },
        "propagateTags": false,
        "platformCapabilities": []
    }
}
```

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 Jobs stuck in a RUNNABLE status. The same reason is also available in the `statusReason` field in the `DescribeJobs` API action.

Example Job queue blocked event

Job queue blocked 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": [],
        "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/first-
run-job-definition:1",
        "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": []
        },
        "propagateTags": false,
        "platformCapabilities": []
    }
}
```

Service job state change events

Anytime that an existing service job changes states, an event is created. For more information about service job states, see Mapping AWS Batch service job status to SageMaker AI status.

> **Note**
>
> Events aren't created for the initial job submission.

Example Service Job State Change Event

Service job state change events are delivered in the following format. The `detail` section resembles the response that's returned from a DescribeServiceJob 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*.

> **Note**
>
> The `tags` and `serviceRequestPayload` fields are not included in the event `detail`.

```
{
  "version": "0",
  "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
  "detail-type": "Batch Service 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:service-job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8"
  ],
  "detail": {
    "attempts": [
      {
        "serviceResourceId": {
          "name": "TrainingJobArn",
          "value": "arn:aws:sagemaker:us-east-1:123456789012:training-job/AWSBatchmy-
training-job88b610a69aa8380ca5b0a7aba3f81cb8"
        },
        "startedAt": 1641944300058,
        "stoppedAt": 1641944400058,
        "statusReason": "Received status from SageMaker: Training job completed"
      }
    ],
    "createdAt": 1641944200058,
    "jobArn": "arn:aws:batch:us-east-1:123456789012:service-job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
    "jobId": "0bb17543-ece6-4480-b1a7-a556d344746b",
    "jobName": "event-test",
    "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/HappyPathJobQueue",
    "latestAttempt": {
      "serviceResourceId": {
        "name": "TrainingJobArn",
        "value": "arn:aws:sagemaker:us-east-1:123456789012:training-job/AWSBatchmy-training-job88b610a69aa8380ca5b0a7aba3f81cb8"
      }
    },
    "serviceJobType": "SAGEMAKER_TRAINING",
    "startedAt": 1641944300058,
    "status": "SUCCEEDED",
    "statusReason": "Received status from SageMaker: Training job completed",
    "stoppedAt": 1641944400058,
    "timeoutConfig": {
      "attemptDurationSeconds": 60
    }
  }
}
```

Service job queue blocked events

Anytime that AWS Batch detects a blocked queue, an event is created in Amazon CloudWatch Events. The reason for the blocked queue is available in the `statusReason` field in the `DescribeServiceJob` API action.

Example Service Job Queue Blocked Event

Service job queue blocked events are delivered in the following format. The `detail` section resembles the response that's returned from the DescribeServiceJob 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*.

> **Note**
>
> The `tags` and `serviceRequestPayload` fields are not included in the event `detail`.

```
{
  "version": "0",
  "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
  "detail-type": "Batch Service 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:service-job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8"
  ],
  "detail": {
    "attempts": [],
    "createdAt": 1641944200058,
    "jobArn": "arn:aws:batch:us-east-1:123456789012:service-job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8",
    "jobId": "6271dfdf-d8a7-41b1-a4d2-55a2224f5375",
    "jobName": "event-test",
    "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/HappyPathJobQueue",
    "serviceJobType": "SAGEMAKER_TRAINING",
    "status": "RUNNABLE",
    "statusReason": "blocked-reason",
    "timeoutConfig": {
      "attemptDurationSeconds": 60
    }
  }
}
```

Tutorial: Use 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, Amazon Q Developer in chat applications 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.

Topics

- Tutorial: Create a scheduled AWS Batch job
- Tutorial: Create a rule with an event pattern
- Tutorial: Pass event information to an AWS Batch target on a schedule using the EventBridge input transformer

Tutorial: Create 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

> **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 **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*.

Tutorial: Create 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*.

Tutorial: Pass 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.

    ```
    {
        "instance": "$.detail.jobId",
        "state": "$.detail.status"
    }
    ```

21. For **Template**, enter the following.

    ```
    {
        "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: Listen for AWS Batch job events using 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 tutorials 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.

Topics

- Tutorial: Create the Lambda function
- Tutorial: Register the event rule
- Tutorial: Test your configuration

Tutorial: 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:

```
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**.

Tutorial: Register the 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 in the *Amazon EventBridge User Guide*.

To create 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.

    ```
    {
      "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:
    1. For **Maximum age of event**, enter a value between one minute (00:01) and 24 hours (24:00).
    2. For **Retry attempts**, enter a number between 0 and 185.
    3. 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**.

Tutorial: 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. Submit a new AWS Batch job. For more information, see Tutorial: submit a job.
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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 Amazon 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 tutorials to create one.

Topics

- Tutorial: Create and subscribe to an Amazon SNS topic
- Tutorial: Register an event rule
- Tutorial: Test your rule
- Alternate rule: Batch job queue blocked

Tutorial: Create and subscribe to an Amazon SNS topic

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.

To create an Amazon SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
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**.

Tutorial: Register an 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.

    ```
    {
      "detail-type": [
        "Batch Job State Change"
      ],
      "source": [
        "aws.batch"
      ],
      "detail": {
        "status": [
          "FAILED"
        ]
      }
    }
    ```

    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 in the *Amazon EventBridge User Guide*.
17. Choose **Next**.
18. Review the details of the rule and choose **Create rule**.

Tutorial: 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. Submit a new AWS Batch job. For more information, see Tutorial: submit a job. For the job's command, substitute this command to exit the container with an exit code of 1.

```
/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 these tutorials with the following alterations:

1. **In Tutorial: Create and subscribe to an Amazon SNS topic**, use *`BlockedJobQueue`* as the topic name.
2. **In Tutorial: Register an event rule**, use the following pattern in the JSON editor:

```
{
   "detail-type": [
     "Batch Job Queue Blocked"
   ],
   "source": [
     "aws.batch"
   ]
}
```

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*.

> **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.
>
> ```
> $ 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

```
{
  "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*.

Monitor AWS Batch

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Batch and your AWS solution.

We strongly encourage you to collect monitoring data from all parts of your AWS solution to make it easier to debug a multi-point failure, if one occurs. Start by creating a monitoring plan that answers the following questions. If you're not sure how to answer these, you can still use Amazon CloudWatch Logs to establish your performance baselines.

- What are your monitoring goals?
- Which resources will you monitor?
- How often will you monitor these resources?
- Which monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

Your next step is to establish a baseline of normal AWS Batch performance in your environment by measuring performance at various times and under different load conditions. As you monitor AWS Batch, keep historical monitoring data so that you can compare it with current performance data. This will help you identify normal performance patterns and performance anomalies, and devise methods to address issues.

The topics in this section can help you start logging and monitoring AWS Batch.

Topics

- Using CloudWatch Logs with AWS Batch
- AWS Batch CloudWatch Container Insights
- Use CloudWatch Logs to monitor AWS Batch on Amazon EKS jobs

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? 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 “Tutorial: Add a CloudWatch Logs IAM policy”.
- 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 “Use the awslogs log driver”.

Topics

- Tutorial: Add a CloudWatch Logs IAM policy
- Install and configure the CloudWatch agent
- Tutorial: View CloudWatch Logs

Tutorial: 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.

Note

If the `ECS-CloudWatchLogs` policy isn't attached to the `ecsInstanceRole` role, basic metrics can still be sent to CloudWatch Logs. However, the basic metrics don't include log data or detailed metrics such as free disk space.

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 or AWS Batch API Reference.

To create the `ECS-CloudWatchLogs` IAM policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. Choose **JSON**, then enter the following policy:

   JSON

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

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`

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose `ecsInstanceRole`. If the role doesn't exist, follow the procedures in Amazon ECS instance role 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 and Advanced details in the *Amazon EC2 User Guide*.

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 and Getting started with AWS Batch tutorials.

Note

Launch templates are not supported on AWS Fargate resources.

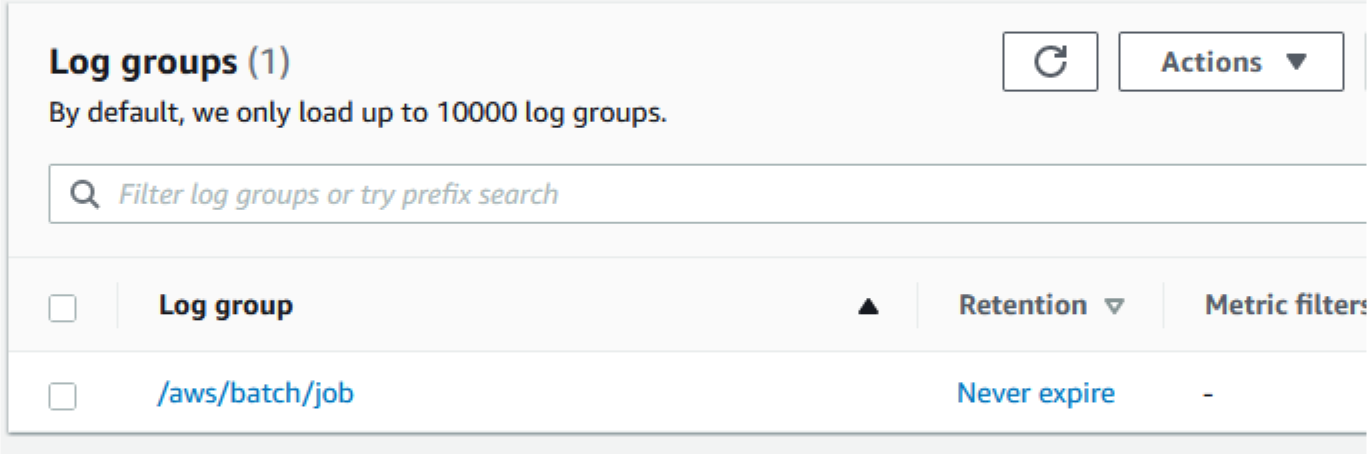
Tutorial: View CloudWatch Logs

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

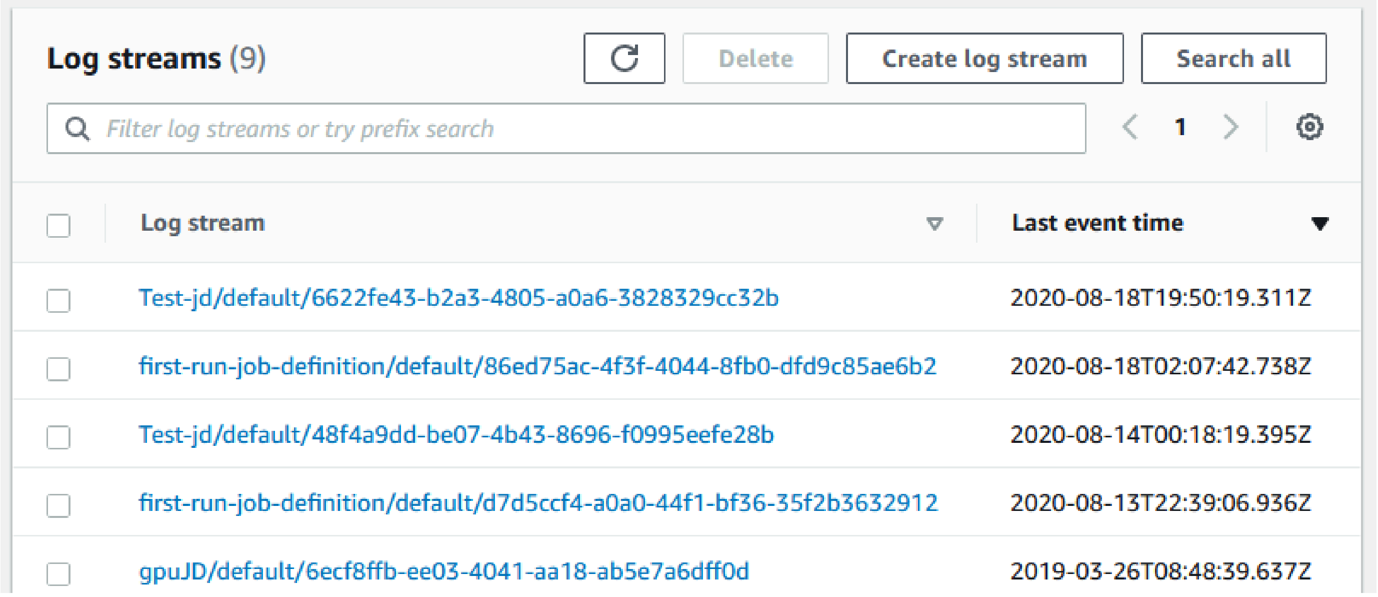
1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**, then choose **Log groups**.



By default, we only load up to 10000 log groups.

☐	Log group	Retention	Metric filters
☐	/aws/batch/job	Never expire	-

3. Choose a log group to view.

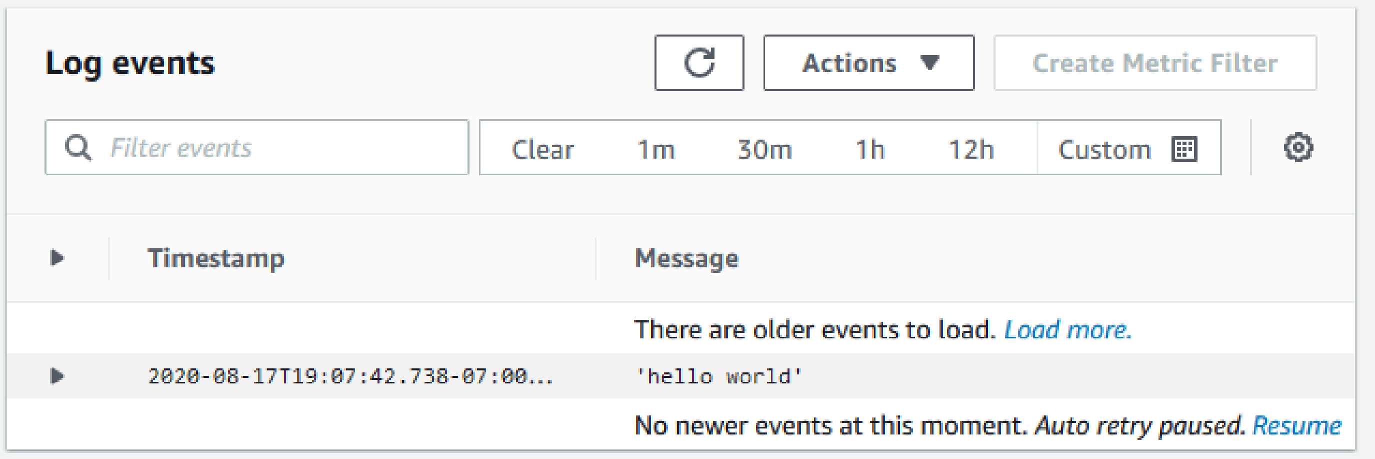


☐	Log stream	Last event time
☐	Test-jd/default/6622fe43-b2a3-4805-a0a6-3828329cc32b	2020-08-18T19:50:19.311Z
☐	first-run-job-definition/default/86ed75ac-4f3f-4044-8fb0-dfd9c85ae6b2	2020-08-18T02:07:42.738Z
☐	Test-jd/default/48f4a9dd-be07-4b43-8696-f0995eefe28b	2020-08-14T00:18:19.395Z
☐	first-run-job-definition/default/d7d5ccf4-a0a0-44f1-bf36-35f2b3632912	2020-08-13T22:39:06.936Z
☐	gpuJD/default/6ecf8ffb-ee03-4041-aa18-ab5e7a6dff0d	2019-03-26T08:48:39.637Z

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**.


Log events
Actions
Create Metric Filter
Filter events
Clear
1m
30m
1h
12h
Custom
Timestamp
Message
There are older events to load. Load more.
No newer events at this moment. Auto retry paused. Resume

```
2020-08-17T19:07:42.738-07:00...    'hello world'
```

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

Topics

- Turn on Container Insights

Turn on Container Insights

Complete the following steps to turn on Container Insights for AWS Batch compute environments.

1. Open the AWS Batch console.
2. Choose **Environments**.
3. Choose the compute environment that you want.
4. On the **Container insights** tab, 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.

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 Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

Prerequisites

- Attach the `CloudWatchAgentServerPolicy` policy to the AWS Identity and Access Management policy of your worker nodes. For more information, see Verify prerequisites.

Install the add-on

For instructions on how to install AWS for Fluent Bit and create the CloudWatch groups, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

You must provide the following additional configuration data when installing the add-on:

- If you install the add-on with the AWS Management Console you need to provide the following tolerations in **Configuration values**:

```
{
  "tolerations": [
    {
      "key": "batch.amazonaws.com/batch-node",
      "operator": "Exists"
    }
  ]
}
```

- If you install the add-on with the AWS CLI then add the following arguments:

```
--configuration-values '{"tolerations":[{"key":"batch.amazonaws.com/batch-
node","operator":"Exists"}]}'
```

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.

Tag your AWS Batch resources

To help you manage your AWS Batch resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.

Topics

- Tag basics
- Tag your resources
- Tag restrictions
- Tutorial: Manage tags using the console
- Manage tags using the CLI or API

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.

Tag 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.

Tag support for AWS Batch resources

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (AWS Batch API, AWS CLI, AWS SDK)
AWS Batch compute environments	Yes	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	Yes

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (AWS Batch API, AWS CLI, AWS SDK)
		puteEnvironment API operation.	
AWS Batch jobs	Yes	Yes	Yes
AWS Batch job definitions	Yes	No	Yes
AWS Batch job queues	Yes	No	Yes
AWS Batch scheduling policies	Yes	No	Yes

Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple 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.

Tutorial: Manage 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.

Add 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.

Add and delete 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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.

Manage 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.

Tag support for AWS Batch resources

Task	API action	AWS CLI	AWS Tools for Windows PowerShell
Add or overwrite one or more tags.	TagResource	tag-resource	Add-BATResourceTag
Delete one or more tags.	UntagResource	untag-resource	Remove-BATResourceTag
List tags for a resource	ListTagsForResource	list-tags-for-resource	Get-BATResourceTag

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.

```
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.

```
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.

```
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.

Task	API action	AWS CLI	AWS Tools for Windows PowerShell
Create a compute environment	CreateComputeEnvironment	create-compute-environment	New-BATComputeEnvironment
Create a job queue	CreateJobQueue	create-job-queue	New-BATJobQueue
Create a scheduling policy	CreateSchedulingPolicy	create-scheduling-policy	New-BATSchedulingPolicy
Register a job definition	RegisterJobDefinition	register-job-definition	Register-BATJobDefinition
Submit a job	SubmitJob	submit-job	Submit-BATJob
Create a consumable resource	CreateConsumableResource	create-consumable-resource	Create-BATConsumableResource

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.

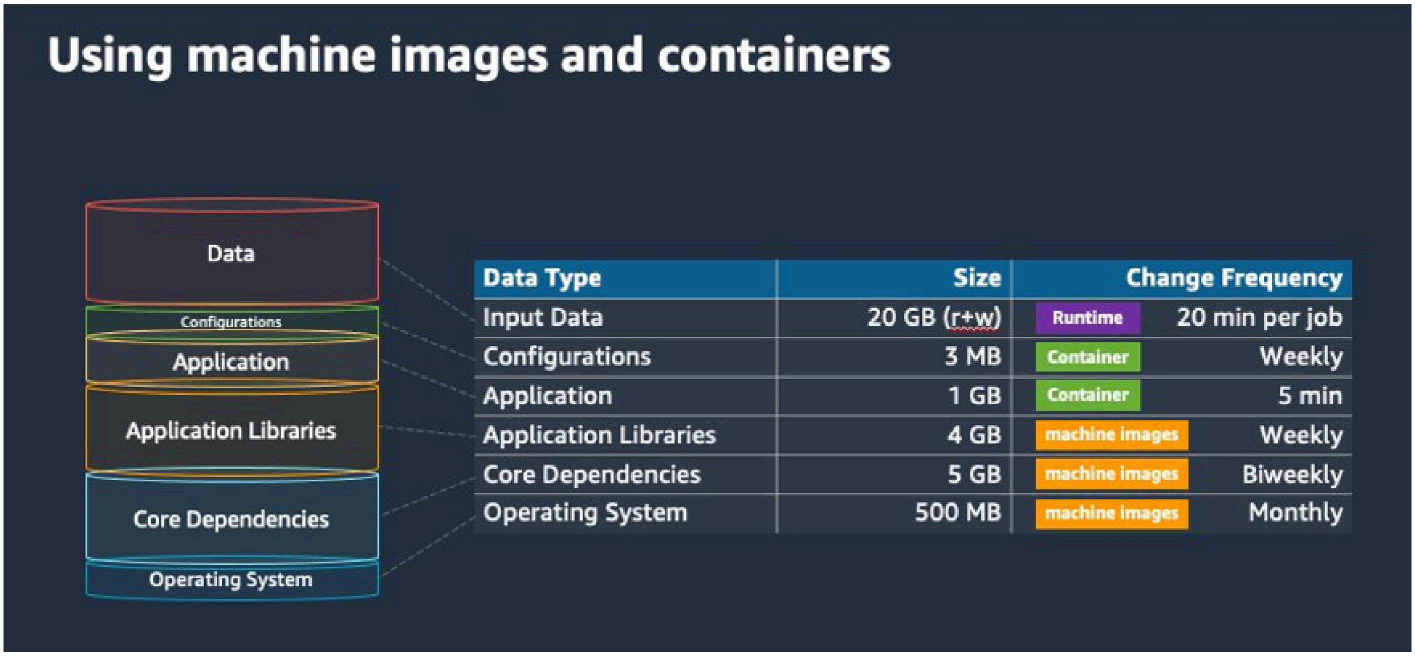
- **Check your Amazon EC2 quotas** – Check your Amazon EC2 quotas (also known as limits) in the Service Quotas panel of the AWS Management Console. If necessary, request a quota increase for your peak number of Amazon EC2 instances. Remember that Amazon EC2 Spot and Amazon On-Demand instances have separate 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.



Using machine images and containers

Data Type	Size	Change Frequency	
Input Data	20 GB (r+w)	Runtime	20 min per job
Configurations	3 MB	Container	Weekly
Application	1 GB	Container	5 min
Application Libraries	4 GB	machine images	Weekly
Core Dependencies	5 GB	machine images	Biweekly
Operating System	500 MB	machine images	Monthly

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 Instance type allocation strategies for AWS Batch.
- **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"
  ```

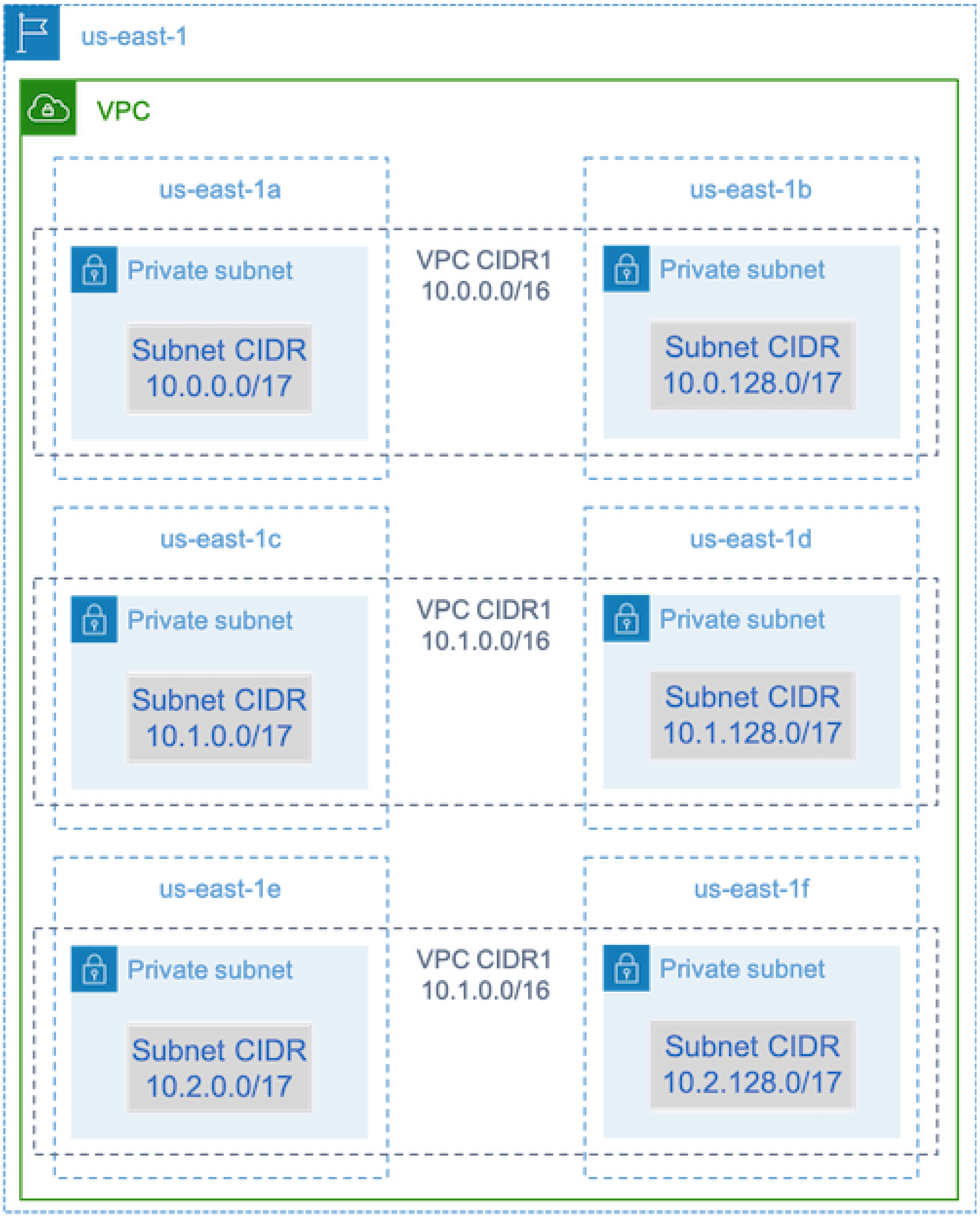
```
    },{
      "onReason" : "*",
        "action": "EXIT"
    }]
}
```

- **Use the Spot Interruption Dashboard** – You can use the Spot Interruption Dashboard to track Spot interruptions. The application provides metrics on Amazon EC2 Spot instances that are reclaimed and which Availability Zones that Spot instances are in. For more information, see Spot Interruption Dashboard

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 EC2resources?.
- **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 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.

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*

Contents

- AWS Batch
  - Optimal instance type configuration to receive automatic instance family updates
  - INVALID compute environment
    - Incorrect role name or ARN
    - Repair 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
- AWS Batch on Amazon EKS job is stuck in STARTING status
    - Scenario: Persisted Volume Claim Attach or Mount Failure
- Verify that the aws-auth ConfigMap is configured correctly
- RBAC permissions or bindings aren't configured properly

AWS Batch

Review the following topics to find review processes and potential solutions to common issues that you may encounter when using AWS Batch.

Topics

- Optimal instance type configuration to receive automatic instance family updates
- INVALID compute environment
- Jobs stuck in a RUNNABLE status
- Spot Instances not tagged on creation
- Spot Instances not scaling down
- Can't retrieve Secrets Manager secrets
- Can't override job definition resource requirements
- Error message when you update the desiredvCpus setting

Optimal instance type configuration to receive automatic instance family updates

> **Note**
>
> Starting on 11/01/2025 the behavior of `optimal` is going to be changed to match `default_x86_64`. During the change your instance families could be updated to a newer generation. You do not need to perform any actions for the upgrade to happen.

AWS Batch supported a single option in **instanceTypes** for `optimal` to match the demand of your job queues. We've introduced two new instance type options: `default_x86_64` and `default_arm64`. We will use `default_x86_64` if you make no instance type selection. These new options will automatically select cost-effective instance types across different families and generations based on your job queue requirements, allowing you to get your workloads running quickly.

As sufficient capacity of new instance types become available in an AWS Region, the corresponding default pool will be automatically updated with the new instance type. The existing `optimal` option will continue to be supported and is not being deprecated, as it will be supported by the underlying default pools to provide updated instances going forward. If you are using '`optimal`, no action is needed on your part.

However, please be aware that only `ENABLED` and `VALID` Compute Environments (CEs) will be automatically updated with new instance types. If you have any `DISABLED` or `INVALID` CEs, they will receive updates once they are re-enabled and set to a `VALID` state.

`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.

> **Important**
>
> AWS Batch creates and manages multiple AWS resources on your behalf and within your account, including Amazon EC2 Launch Templates, Amazon EC2 Auto Scaling Groups, Amazon EC2 Spot Fleets, and Amazon ECS Clusters. These managed resources

are configured specifically to ensure optimal AWS Batch operation. Manually modifying these Batch-managed resources, unless explicitly stated in AWS Batch documentation, may result in unexpected behavior resulting in `INVALID` Compute Environment, sub-optimal instance scaling behavior, delayed workload processing, or unexpected costs. These manual modifications can not be deterministically supported by the AWS Batch service. Always use the supported Batch APIs or the Batch console to manage your Compute Environments.

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 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 Repair an `INVALID` compute environment.

Repair 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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.

   > **Tip**
   >
   > For directions on how to create a service linked role, see Using roles for AWS Batch.

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 Job queue blocked events 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. Manually canceling the job will allow the subsequent job to move to the head of the queue but 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:** `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 [Using service-linked roles for AWS Batch](#) service-linked role (SLR) or the [AWS managed policy: **AWSBatchServiceRole** policy](#) 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:JOB_RESOURCE_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 managed policies for AWS Batch and address any gaps. Note:You can't configure a programmable action through the `jobStateTimeLimitActions` parameter to resolve this error.

   It's a best practice to use the Using service-linked roles for AWS Batch 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.`

5. **Reason:** Your compute environment has an unsupported instance type configuration. This can occur when instance types are not available in your selected Availability Zones, or when your launch template or launch configuration contains settings incompatible with the specified instance types. To resolve this, verify that your instance types are supported in your specified AWS Region and Availability Zones, check that your launch template settings are compatible with your instance types, and consider updating to newer generation instance types. For more information about finding supported instance types, see Finding an Amazon EC2 instance type in the *Amazon EC2 User Guide*.

   - **`statusReason` message while the job is stuck:** `MISCONFIGURATION:EC2_INSTANCE_CONFIGURATION_UNSUPPORTED - Your compute environment associated with this job queue has an unsupported instance type configuration.`

6. **Reason:** All compute environments are invalid. For more information, see `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.`

7. **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 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 Tutorial: Create 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*.

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 and Creating a role to delegate permissions to an AWS Service 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

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**.

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.

   ```
   $ 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.

   ```
   $ 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 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.

```
"containerOverrides": {
   "memory": 8192,
   "vcpus": 4
}
```

Change them to the following:

```
"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.

```
{
   "containerProperties": {
      "memory": 4096,
      "vcpus": 2,
}
```

Change them to the following:

```
"containerProperties": {
   "resourceRequirements": [
      {
         "type": "MEMORY",
         "value": "4096"
      },
      {
         "type": "VCPU",
         "value": "2"
      }
   ],
}
```

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
- AWS Batch on Amazon EKS job is stuck in STARTING status
- Verify that the aws-auth ConfigMap is configured correctly
- RBAC permissions or bindings aren't configured properly

Review the following topics to find review processes and potential solutions to common issues that you may encounter when using AWS Batch on Amazon Elastic Kubernetes Service.

`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:

- `1.34`
- `1.33`
- `1.32`

- `1.31`
- `1.30`
- `1.29`

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`.

> **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-authConfigMap` 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 2: Prepare 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*.

AWS Batch on Amazon EKS job is stuck in `STARTING` status

A Job may remain in `STARTING` status when the Pod is stuck in `PENDING` on `ContainerCreating` for any long running requests from the kubelet (`pull`, `log`, `exec`, and `attach`) until the Pod startup issue is resolved or the Job is terminated. In the qualifying scenarios below AWS Batch will terminate the job on your behalf, otherwise the job must be terminated manually using the TerminateJob API.

To verify the reason a Job may be stuck in `STARTING`, use Tutorial: Map a running job to a pod and a node to find the `podName`, and describe the Pod:

```
% kubectl describe pod aws-batch.000c8190-87df-31e7-8819-176fe017a24a -n my-aws-batch-
namespace
Name:             aws-batch.000c8190-87df-31e7-8819-176fe017a24a
Namespace:        my-aws-batch-namespace
...
Containers:
  default:
...
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
...
Conditions:
  Type                        Status
  PodReadyToStartContainers   False
  Initialized                 True
  Ready                       False
  ContainersReady             False
  PodScheduled                True
...
Events:
  Type     Reason       Age    From     Message
  ----     ------       ----   ----     -------
  Warning  FailedMount  2m32s  kubelet  Unable to attach or mount volumes: ...
```

Consider configuring your EKS cluster to Send control plane logs to CloudWatch Logs for full visibility.

Scenario: Persisted Volume Claim Attach or Mount Failure

Jobs using Persistent Volume Claims where the volume fails to attach or mount are candidates for termination. This can be a result of an incorrectly configured Job Definition. See Create a single-node job definition on Amazon EKS resources for more details.

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`.

   ```
   $ 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**
   >
   > 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-authconfigmap.

   > **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 2: Prepare 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:

```
$ 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.

```
Name:         aws-batch-cluster-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources                                      Non-Resource URLs  Resource Names
 Verbs
  ---------                                      -----------------  --------------
 -----
  configmaps                                     []                 []
 [get list watch]
  nodes                                          []                 []
 [get list watch]
  pods                                           []                 []
 [get list watch]
  daemonsets.apps                                []                 []
 [get list watch]
  deployments.apps                               []                 []
 [get list watch]
  replicasets.apps                               []                 []
 [get list watch]
  statefulsets.apps                              []                 []
 [get list watch]
  clusterrolebindings.rbac.authorization.k8s.io  []                 []
 [get list]
```

```
  clusterroles.rbac.authorization.k8s.io          []                  []
 [get list]
  namespaces                                      []                  []
 [get]
  events                                          []                  []
 [list]
```

```
$ kubectl describe role aws-batch-compute-environment-role -n my-aws-batch-namespace
```

The following is example output.

```
Name:         aws-batch-compute-environment-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources                               Non-Resource URLs  Resource Names  Verbs
  ---------                               -----------------  --------------  -----
  pods                                    []                 []              [create
 get list watch delete patch]
  serviceaccounts                         []                 []              [get list]
  rolebindings.rbac.authorization.k8s.io  []                 []              [get list]
  roles.rbac.authorization.k8s.io         []                 []              [get list]
```

To resolve this issue, re-apply the RBAC permissions and `rolebinding` commands. For more information, see Step 2: Prepare your Amazon EKS cluster for AWS Batch.

Resource: AWS Batch service quotas

The following table provides the service quotas for AWS Batch that can't be changed. Each quota is Region specific.

Resource	Quota
Maximum number of job queues. For more information, see *Job queues*.	50
Maximum number of compute environments across Amazon ECS and Amazon EKS. For more information, see *Compute environments for AWS Batch*.	50
Maximum number of compute environments per Amazon EKS cluster.	5
Maximum number of compute environments for each job queue	3
Maximum number of job dependencies for a job	20
Maximum job definition size (for `RegisterJobDefinition` API operations)	24 KiB
Maximum job payload size (for `SubmitJob` API operations)	30 KiB
Maximum array size for array jobs	10000
Maximum number of jobs in `SUBMITTED` state	1000000
Maximum number of transactions per second (TPS) for each account for `SubmitJob` operations	50
Maximum number of consumable resources	50k
Maximum number of service environments. For more information, see *Service environments*.	50
Maximum number of service environments for each job queue	1
Maximum size of `SubmitServiceJob` request	30 KiB
Maximum job service request payload size (for `SubmitServiceJob` API operations)	10 KiB

Resource	Quota
Maximum number of transactions per second (TPS) for each account for `SubmitServiceJob` operations	5
Maximum number of attempts with retry strategy for a service job	10

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*.

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.

Change	Description	Date
Added default_x86_64 and default_arm64	Added new `default_x86_64` and `default_arm64` for **Allowed instance types**.	August 15, 2025
Added service environments and service jobs	Added service environments and service jobs for using AWS Batch with SageMaker AI.	July 30, 2025
Added AWSServiceRoleForAWSBatchWithSagemaker and AWSBatchServiceRolePolicyForSageMaker	Added new AWS service-linked role `AWSServiceRoleForAWSBatchWithSagemaker` and managed policy `AWSBatchServiceRolePolicyForSageMaker` that allows AWS Batch to manage SageMaker AI on your behalf.	July 30, 2025
Adds support for EKS AL 2023 AMIs	How to upgrade from EKS AL2 to EKS AL2023	June 24, 2025
Adds support for FireLens and ECS Exec command	Adds support for FireLens and ECS Exec command.	April 15, 2025
Adds support for resource-aware scheduling for AWS Batch	Adds support for resource-aware scheduling for AWS Batch for Amazon Elastic Container Service, Amazon	February 27, 2025

	Elastic Kubernetes Service, and AWS Fargate.	
Updated AWS Batch supported Amazon EKS versions	Updated the Amazon EKS versions that AWS Batch supports to remove version 1.22.	March 11, 2024
Updated AWS Batch supported Amazon EKS versions	Updated the Amazon EKS versions that AWS Batch supports to include version 1.29.	February 29, 2024
Automated job retries	Corrected the code sample.	February 29, 2024
Adds support for multi-container jobs for AWS Batch	Adds support for multi-container jobs for AWS Batch for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, and AWS Fargate.	February 28, 2024
Updated AWS Batch supported Amazon EKS versions	Updated the Amazon EKS versions that AWS Batch supports to include version 1.28	January 27, 2024

Updated BatchServiceRolePolicy and AWSBatchServiceRole	**BatchServiceRolePolicy** Updated to add support for describing Spot Fleet request history and Amazon EC2 Auto Scaling activities. **AWSBatchServiceRole** Updated to add statement IDs, grant AWS Batch permissions to `ec2:DescribeSpotFleetRequestHistory` and `autoscaling:DescribeScalingActivities`.	December 5, 2023
AWS Batch on Amazon EKS	AWS Batch adds support for running jobs on Amazon EKS clusters.	October 25, 2022
Cross-service confused deputy prevention for AWS Batch	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.	June 6, 2022

Interface VPC Endpoints (AWS PrivateLink)	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 Direct Connect.	April 15, 2022
Enhanced compute environment updates	AWS Batch enhanced support updates to compute environments.	April 14, 2022
AWS managed policy updates - Update to existing policies	AWS Batch updated existing managed policies.	December 6, 2021
Fair-share scheduling	AWS Batch adds support for adding scheduling policies to job queues.	November 9, 2021
Amazon EFS	AWS Batch adds support for adding Amazon EFS file systems to your job definitions.	April 1, 2021
Added service-linked role	AWS Batch adds the **AWSServiceRoleForBatch** service-linked role.	March 10, 2021
AWS Fargate support	AWS Batch adds support for running jobs on Fargate resources.	December 3, 2020

Resource tagging	AWS Batch adds support for adding metadata tags to your compute environments, job definitions, job queues, and jobs.	October 7, 2020
Secrets	AWS Batch adds support for passing secrets to jobs.	October 1, 2020
Logging	AWS Batch adds support for specifying additional log drivers for jobs.	October 1, 2020
Allocation strategies	AWS Batch adds support for multiple strategies to choose instance types.	October 16, 2019
EFA support	AWS Batch adds support for Elastic Fabric Adapter (EFA) devices.	August 2, 2019
GPU scheduling	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.	April 4, 2019
Multi-node parallel jobs	AWS Batch adds support for multi-node parallel jobs. You can use this feature run single jobs that span over multiple Amazon EC2 instances.	November 19, 2018
Resource-level permissions	AWS Batch supports resource-level permissions on several API operations.	November 12, 2018

Amazon EC2 Launch template support	AWS Batch adds support for using launch templates with compute environments.	November 12, 2018
AWS Batch job timeouts	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.	April 5, 2018
AWS Batch jobs as EventBridge targets	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.	March 1, 2018
CloudTrail auditing for AWS Batch	CloudTrail can audit calls made to AWS Batch API actions.	January 10, 2018
Array jobs	AWS Batch adds support for array jobs. You can use array jobs for parameter sweep and Monte Carlo workloads.	November 28, 2017
Expanded AWS Batch tagging	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.	October 26, 2017

AWS Batch event stream for EventBridge	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.	October 24, 2017
Automated job retries	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.	March 28, 2017
AWS Batch general availability	AWS Batch is introduced, designed as a means for you to run batch computing workloads on the AWS Cloud.	January 5, 2017