Amazon EMR

Amazon EMR on EKS Development Guide

Amazon EMR: Amazon EMR on EKS Development Guide

Table of Contents

What is Amazon EMR on EKS

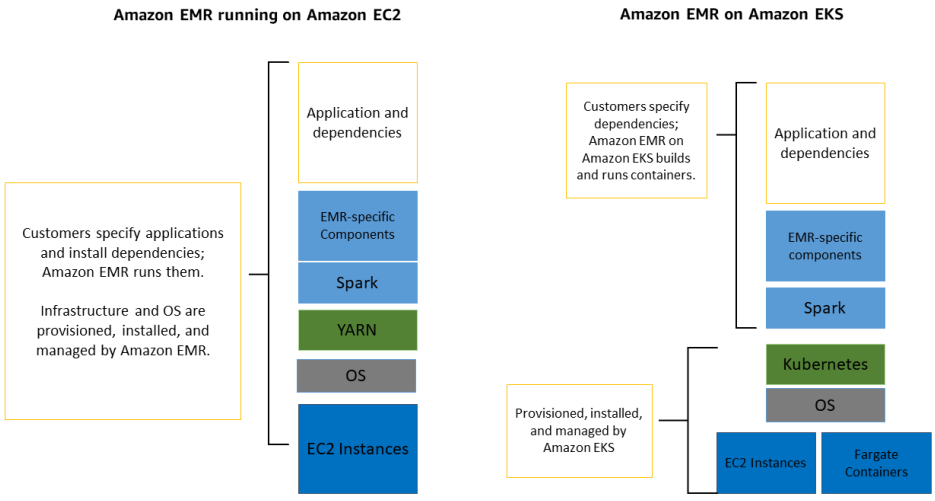
Amazon EMR on EKS provides a deployment option for Amazon EMR that allows you to run open-source big data frameworks on Amazon Elastic Kubernetes Service (Amazon EKS). With this deployment option, you can focus on running analytics workloads while Amazon EMR on EKS builds, configures, and manages containers for open-source applications.

If you already use Amazon EMR, you can now run Amazon EMR based applications with other types of applications on the same Amazon EKS cluster. This deployment option also improves resource utilization and simplifies infrastructure management across multiple Availability Zones. If you already run big data frameworks on Amazon EKS, you can now use Amazon EMR to automate provisioning and management, and run Apache Spark more quickly.

Amazon EMR on EKS enables your team to collaborate more efficiently and process vast amounts of data more easily and cost-effectively:

- You can run applications on a common pool of resources without having to provision infrastructure. You can use Amazon EMR Studio (Preview) and the AWS SDK or AWS CLI to develop, submit, and diagnose analytics applications running on EKS clusters. You can run scheduled jobs on Amazon EMR on EKS using self-managed Apache Airflow or Amazon Managed Workflows for Apache Airflow (MWAA).
- Infrastructure teams can centrally manage a common computing platform to consolidate Amazon EMR workloads with other container-based applications. You can simplify infrastructure management with common Amazon EKS tools and take advantage of a shared cluster for workloads that need different versions of open-source frameworks. You can also reduce operational overhead with automated Kubernetes cluster management and OS patching. With Amazon EC2 and AWS Fargate, you can enable multiple compute resources to meet performance, operational, or financial requirements.

The following diagram shows the two different deployment models for Amazon EMR.



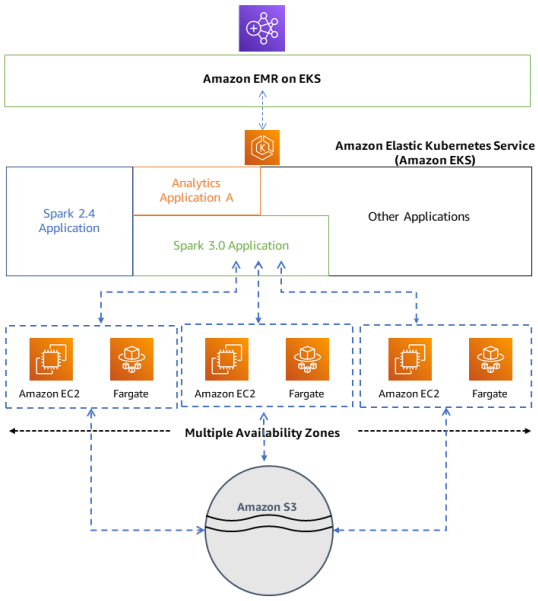
Topics

Architecture

Amazon EMR on EKS loosely couples applications to the infrastructure that they run on. Each infrastructure layer provides orchestration for the subsequent layer. When you submit a job to Amazon EMR, your job definition contains all of its application-specific parameters. Amazon EMR uses these parameters to instruct Amazon EKS about which pods and containers to deploy. Amazon EKS then brings online the computing resources from Amazon EC2 and AWS Fargate required to run the job.

With this loose coupling of services, you can run multiple, securely isolated jobs simultaneously. You can also benchmark the same job with different compute backends or spread your job across multiple Availability Zones to improve availability.

The following diagram illustrates how Amazon EMR on EKS works with other AWS services.



Concepts

Kubernetes namespace

Amazon EKS uses Kubernetes namespaces to divide cluster resources between multiple users and applications. These namespaces are the foundation for multi-tenant environments. A Kubernetes namespace can have either Amazon EC2 or AWS Fargate as the compute provider. This flexibility provides you with different performance and cost options for your jobs to run on.

Virtual cluster

A virtual cluster is a Kubernetes namespace that Amazon EMR is registered with. Amazon EMR uses virtual clusters to run jobs and host endpoints. Multiple virtual clusters can be backed by the same

physical cluster. However, each virtual cluster maps to one namespace on an EKS cluster. Virtual clusters do not create any active resources that contribute to your bill or that require lifecycle management outside the service.

Job run

A job run is a unit of work, such as a Spark jar, PySpark script, or SparkSQL query, that you submit to Amazon EMR on EKS. One job can have multiple job runs. When you submit a job run, you include the following information:

- A virtual cluster where the job should run.
- A job name to identify the job.
- The execution role — a scoped IAM role that runs the job and allows you to specify which resources can be accessed by the job.
- The Amazon EMR release label that specifies the version of open-source applications to use.
- The artifacts to use when submitting your job, such as spark-submit parameters.

By default, logs are uploaded to the Spark History server and are accessible from the AWS Management Console. You can also push event logs, execution logs, and metrics to Amazon S3 and Amazon CloudWatch.

Amazon EMR containers

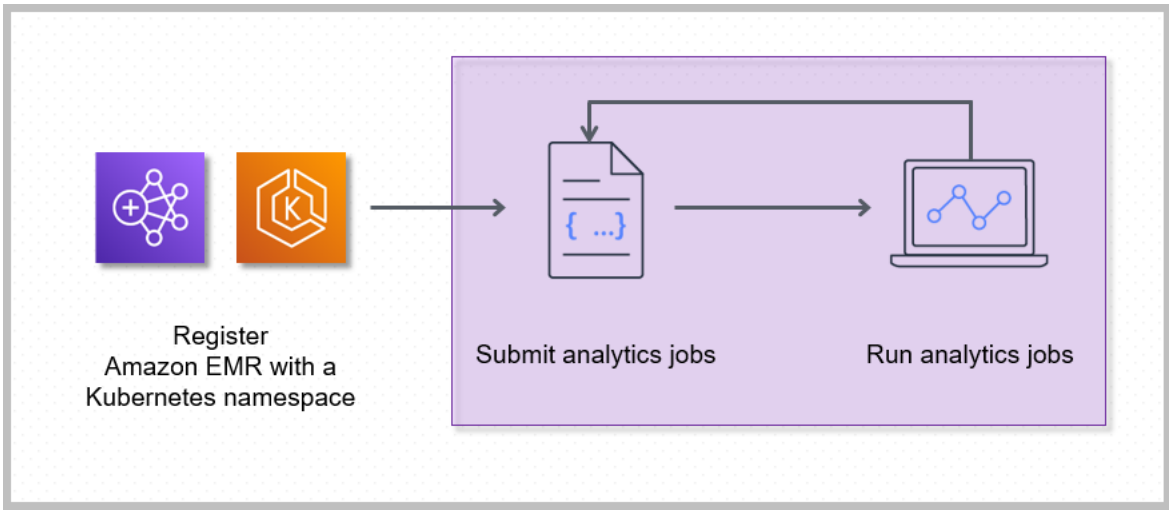
Amazon EMR containers is the API name for Amazon EMR on EKS. The `emr-containers` prefix is used in the following scenarios:

- It is the prefix in the CLI commands for Amazon EMR on EKS. For example, `aws emr-containers start-job-run`.
- It is the prefix before IAM policy actions for Amazon EMR on EKS. For example, `"Action": [ "emr-containers:StartJobRun"]`. For more information, see Policy actions for Amazon EMR on EKS.
- It is the prefix used in Amazon EMR on EKS service endpoints. For example, `emr-containers.us-east-1.amazonaws.com`. For more information, see Amazon EMR on EKS Service Endpoints.

How the components work together

The following steps and diagram illustrate the Amazon EMR on EKS workflow:

- Use an existing Amazon EKS cluster or create one by using the eksctl command line utility or Amazon EKS console.
- Create a virtual cluster by registering Amazon EMR with a namespace on an EKS cluster.
- Submit your job to the virtual cluster using the AWS CLI or SDK.



Registering Amazon EMR with a Kubernetes namespace on Amazon EKS creates a virtual cluster. Amazon EMR can then run analytics workloads on that namespace. When you use Amazon EMR on EKS to submit Spark jobs to the virtual cluster, Amazon EMR on EKS requests the Kubernetes scheduler on Amazon EKS to schedule pods.

For each job that you run, Amazon EMR on EKS creates a container with an Amazon Linux 2 base image, Apache Spark, and associated dependencies. Each job runs in a pod that downloads the container and starts to run it. The pod terminates after the job terminates. If the container's image has been previously deployed to the node, then a cached image is used and the download is bypassed. Sidecar containers, such as log or metric forwarders, can be deployed to the pod. After the job terminates, you can still debug it using Spark application UI in the Amazon EMR console.

Setting up

Complete the following tasks to get set up for Amazon EMR on EKS. If you've already signed up for Amazon Web Services (AWS) and have been using Amazon EKS, you are almost ready to use Amazon EMR on EKS. If you have already completed any of these steps, you may skip them and move on to the next step.

Note
You can also follow the Amazon EMR on EKS Workshop to set up all the necessary resources to run Spark jobs on Amazon EMR on EKS. The workshop also provides automation by using CloudFormation templates to create the resources necessary for you to get started.

1. Install the AWS CLI (p. 5)
2. Install `eksctl` (p. 7)
3. Set up an Amazon EKS cluster (p. 8)
4. Enable cluster access for Amazon EMR on EKS (p. 11)
5. Enable IAM Roles for Service Accounts (IRSA) on the EKS cluster (p. 14)
6. Create a job execution role (p. 15)
7. Update the trust policy of the job execution role (p. 15)
8. Grant users access to Amazon EMR on EKS (p. 16)
9. Register the Amazon EKS cluster with Amazon EMR (p. 18)

Install the AWS CLI

You can install the latest version of the AWS CLI for macOS, Linux, or Windows.

Important
To set up Amazon EMR on EKS, you must have the latest version of AWS CLI installed.

To install or update the AWS CLI for macOS

1. If you currently have the AWS CLI installed, determine which version that you have installed.

```
aws --version
```

2. If you have an earlier version of AWS CLI, then use the following command to install the latest AWS CLI version 2. For other installation options, or to upgrade your currently installed version 2, see Upgrading the AWS CLI version 2 on macOS.

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /
```

If you're unable to use the AWS CLI version 2, then ensure that you have the latest version of the AWS CLI version 1 installed using the following command.

```
pip3 install awscli --upgrade --user
```

To install or update the AWS CLI for Linux

1. If you currently have the AWS CLI installed, determine which version that you have installed.

```
aws --version
```

2. If you have an earlier version of AWS CLI, then use the following command to install the latest AWS CLI version 2. For other installation options, or to upgrade your currently installed version 2, see Upgrading the AWS CLI version 2 on Linux.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

If you're unable to use the AWS CLI version 2, then ensure that you have the latest version of the AWS CLI version 1 installed using the following command.

```
pip3 install --upgrade --user awscli
```

To install or update the AWS CLI for Windows

1. If you currently have the AWS CLI installed, determine which version that you have installed.

```
aws --version
```

2. If you have an earlier version of AWS CLI, then use the following command to install the latest AWS CLI version 2. For other installation options, or to upgrade your currently installed version 2, see Upgrading the AWS CLI version 2 on Windows.

    1. Download the AWS CLI MSI installer for Windows (64-bit) at https://awscli.amazonaws.com/AWSCLIV2.msi
    2. Run the downloaded MSI installer and follow the onscreen instructions. By default, the AWS CLI installs to C:\Program Files\Amazon\AWSCLIV2.

    If you're unable to use the AWS CLI version 2, then ensure that you have the latest version of the AWS CLI version 1 installed using the following command.

```
pip3 install --user --upgrade awscli
```

Configure your AWS CLI credentials

Both eksctl and the AWS CLI require that you have AWS credentials configured in your environment. The `aws configure` command is the fastest way to set up your AWS CLI installation for general use.

```
$ aws configure
AWS Access Key ID [None]: <AKIAIOSFODNN7EXAMPLE>
AWS Secret Access Key [None]: <wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY>
Default region name [None]: <region-code>
Default output format [None]: <json>
```

When you type this command, the AWS CLI prompts you for four pieces of information: Access key, secret access key, AWS Region, and output format. This information is stored in a profile (a collection of settings) named `default`. This profile is used when you run commands unless you specify another one. For more information, see Configuring the AWS CLI in the AWS Command Line Interface User Guide.

Install eksctl

You need to install the latest version of eksctl command line utility on macOS, Linux, or Windows. For more information, see https://eksctl.io/.

Important
To set up Amazon EMR on EKS, you must have eksctl 0.34.0 version or later.

To install or upgrade eksctl on macOS using Homebrew

The easiest way to get started with Amazon EKS and macOS is by installing eksctl with Homebrew. The eksctl Homebrew recipe installs eksctl and any other dependencies that are required for Amazon EKS, such as kubectl. The recipe also installs the aws-iam-authenticator, which is required if you don't have the AWS CLI version 1.16.156 or later installed.

1. If you do not already have Homebrew installed on macOS, install it with the following command.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
install.sh)"
```

2. Install the Weaveworks Homebrew tap.

```
brew tap weaveworks/tap
```

3. 1. Install or upgrade eksctl.
   - Install eksctl with the following command.

     ```
     brew install weaveworks/tap/eksctl
     ```

   - If eksctl is already installed, run the following command to upgrade.

     ```
     brew upgrade eksctl & brew link --overwrite eksctl
     ```

   2. Test that your installation was successful with the following command. You must have eksctl 0.34.0 version or later.

      ```
      eksctl version
      ```

To install or upgrade eksctl on Linux using curl

1. Download and extract the latest release of eksctl with the following command.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/
download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. Move the extracted binary to /usr/local/bin.

```
sudo mv /tmp/eksctl /usr/local/bin
```

3. Test that your installation was successful with the following command. You must have eksctl 0.34.0 version or later.

```
eksctl version
```

To install or upgrade eksctl on Windows using Chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing Chocolatey.
2. Install or upgrade eksctl.
   - Install the binaries with the following command.

     ```
     chocolatey install -y eksctl
     ```
   - If they are already installed, run the following command to upgrade:

     ```
     chocolatey upgrade -y eksctl
     ```
3. Test that your installation was successful with the following command. You must have eksctl 0.34.0 version or later.

   ```
   eksctl version
   ```

Set up an Amazon EKS cluster

Amazon EKS is a managed service that makes it easy for you to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Follow the steps outlined below to create a new Kubernetes cluster with nodes in Amazon EKS.

Prerequisites

Before creating an Amazon EKS cluster, you must install and configure the following tools and resources that you need to create and manage an Amazon EKS cluster,

- The latest version of AWS CLI.
- kubectl version 1.20 or later.
- The latest version of eksctl .

For more information, see Install the AWS CLI (p. 5), Installing kubectl, Install eksctl (p. 7).

Create an Amazon EKS cluster using eksctl

Take the following steps to create an Amazon EKS cluster using eksctl.

Important
To get started quickly, you can create an EKS cluster and the nodes with default settings. But for production use, we recommend that you customize the settings for the cluster and nodes to meet your specific requirements. For a list of all settings and options, run the command eksctl create cluster -h. For more information, see Creating and Managing Clusters in the eksctl documentation.

1. **Create an Amazon EC2 key pair.**

   If you don't have an existing key pair, you can run the following command to create a new key pair. Replace `us-west-2` with the Region where you want to create your cluster.

   ```
   aws ec2 create-key-pair --region us-west-2 --key-name myKeyPair
   ```

   Save the returned output in a file on your local computer. For more information, see Creating or importing a key pair in the Amazon EC2 User Guide for Linux Instances.

   > **Note**
   > A key pair is not required for creating an EKS cluster. But specifying the key pair allows you to SSH to nodes once they're created. You can specify a key pair only when you create the node group.

2. **Create an EKS cluster.**

   Run the following command to create an EKS cluster and nodes. Replace *my-cluster* and *myKeyPair* with your own cluster name and key pair name. Replace *us-west-2* with the Region where you want to create your cluster. For more information about Amazon EKS supported Regions, see Amazon Elastic Kubernetes Service endpoints and quotas.

   ```
   eksctl create cluster \
   --name my-cluster \
   --region us-west-2 \
   --with-oidc \
   --ssh-access \
   --ssh-public-key myKeyPair \
   --instance-types=m5.xlarge \
   --managed
   ```

   > **Important**
   > When creating an EKS cluster, use m5.xlarge as the instance type, or any other instance type with a higher CPU and memory. Using an instance type with lower CPU or memory compared to m5.xlarge may lead to job failure due to insufficient resources available in the cluster. To see all resources created, view the stack named `eksctl-my-cluster-cluster` in the AWS Cloud Formation console.

   The cluster and node creation process takes several minutes. You'll see several lines of output when the cluster and nodes are created. The following example demonstrates the last line of output.

   ```
   ...
   [#]  EKS cluster "my-cluster" in "us-west-2" region is ready
   ```

   `eksctl` created a `kubectl` config file in `~/.kube` or added the new cluster's configuration within an existing config file in `~/.kube`.

3. **View and validate resources**

   Run the following command to view your cluster nodes.

   ```
   kubectl get nodes -o wide
   ```

   The following shows an example output.

   ```
   Amazon EC2 node output
   ```

```
NAME                                          STATUS   ROLES    AGE    VERSION
      INTERNAL-IP      EXTERNAL-IP     OS-IMAGE          KERNEL-VERSION
 CONTAINER-RUNTIME
ip-192-168-12-49.us-west-2.compute.internal    Ready    none     6m7s
 v1.18.9-eks-d1db3c   192.168.12.49    52.35.116.65    Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   docker://19.3.6
ip-192-168-72-129.us-west-2.compute.internal   Ready    none     6m4s
 v1.18.9-eks-d1db3c   192.168.72.129   44.242.140.21   Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   docker://19.3.6
```

For more information, see View nodes.

Use the following command to view the workloads running on your cluster.

```
kubectl get pods --all-namespaces -o wide
```

The following shows an example output.

```
Amazon EC2 output

NAMESPACE     NAME                           READY   STATUS    RESTARTS   AGE     IP
        NODE                                                  NOMINATED NODE   READINESS GATES
kube-system   aws-node-6ctpm                 1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   none             none
kube-system   aws-node-cbntg                 1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    none             none
kube-system   coredns-559b5db75d-26t47       1/1     Running   0          14m
 192.168.78.81    ip-192-168-72-129.us-west-2.compute.internal   none             none
kube-system   coredns-559b5db75d-9rvnk       1/1     Running   0          14m
 192.168.29.248   ip-192-168-12-49.us-west-2.compute.internal    none             none
kube-system   kube-proxy-l8pbd               1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    none             none
kube-system   kube-proxy-zh85h               1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   none             none
```

For more information about what you see here, see View workloads.

Create an EKS cluster using AWS Management Console and AWS CLI

You can also use AWS Management Console and AWS CLI to create an EKS cluster. Follow the steps at Getting started with Amazon EKS – AWS Management Console and AWS CLI. This way gives you visibility into how each resource is created for the EKS cluster and how the resources interact with each other.

Important
When creating nodes for an EKS cluster, use m5.xlarge as the instance type, or any other instance type with a higher CPU and memory.

Create an EKS cluster with AWS Fargate

You can also create an EKS cluster with pods running on AWS Fargate.

1. To create an EKS cluster with pods running on Fargate, follow the steps outlined at Getting Started with AWS Fargate using Amazon EKS.

Note
Amazon EMR on EKS needs CoreDNS for running jobs on EKS cluster. If you want to run your pods only on Fargate, you must follow the steps at Updating CoreDNS.

2. Run the following command to view your cluster nodes.

```
kubectl get nodes -o wide
```

The following shows an example Fargate output.

```
Fargate node output

NAME                                                  STATUS   ROLES    AGE
 VERSION              INTERNAL-IP       EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
                CONTAINER-RUNTIME
fargate-ip-192-168-141-147.us-west-2.compute.internal   Ready    none
 8m3s    v1.18.8-eks-7c9bda   192.168.141.147   none          Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
fargate-ip-192-168-164-53.us-west-2.compute.internal    Ready    none
 7m30s   v1.18.8-eks-7c9bda   192.168.164.53    none          Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
```

For more information, see View nodes.

3. Run the following command to view the workloads running on your cluster.

```
kubectl get pods --all-namespaces -o wide
```

The following shows an example Fargate output.

```
Fargate output

NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE   IP
           NODE                                                    NOMINATED NODE
 READINESS GATES
kube-system   coredns-69dfb8f894-9z95l   1/1     Running   0          18m
 192.168.164.53    fargate-ip-192-168-164-53.us-west-2.compute.internal    none
     none
kube-system   coredns-69dfb8f894-c8v66   1/1     Running   0          18m
 192.168.141.147   fargate-ip-192-168-141-147.us-west-2.compute.internal   none
     none
```

For more information, see View workloads.

Enable cluster access for Amazon EMR on EKS

You must allow Amazon EMR on EKS access to a specific namespace in your cluster by taking the following actions: creating a Kubernetes role, binding the role to a Kubernetes user, and mapping the Kubernetes user with the service linked role `AWSServiceRoleForAmazonEMRContainers`. These actions are automated in `eksctl` when the IAM identity mapping command is used with `emr-containers` as the service name. You can perform these operations easily by using the following command.

```
eksctl create iamidentitymapping \
    --cluster my_eks_cluster \
    --namespace kubernetes_namespace \
```

```
    --service-name "emr-containers"
```

Replace *my_eks_cluster* with the name of your Amazon EKS cluster and replace *kubernetes_namespace* with the Kubernetes namespace created to run Amazon EMR workloads.

Important
You must download the latest eksctl using the previous step Install `eksctl` (p. 7) to use this functionality.

Manual steps to enable cluster access for Amazon EMR on EKS

You can also use the following manual steps to enable cluster access for Amazon EMR on EKS.

1. **Create a Kubernetes role in a specific namespace**

   Run the following command to create a Kubernetes role in a specific namespace. This role grants the necessary RBAC permissions to Amazon EMR on EKS.

```
namespace=my-namespace

cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods", "pods/
log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
EOF
```

2. **Create a Kubernetes role binding scoped to the namespace**

   Run the following command to create a Kubernetes role binding in the given namespace. This role binding grants the permissions defined in the role created in the previous step to a user named

`emr-containers`. This user identifies service-linked roles for Amazon EMR on EKS and thus allows Amazon EMR on EKS to perform actions as defined by the role you created.

```
namespace=my-namespace

cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: emr-containers
  namespace: ${namespace}
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
EOF
```

3. **Update Kubernetes `aws-auth` configuration map**

   You can use one of the following options to map the Amazon EMR on EKS service-linked role with the `emr-containers` user that was bound with the Kubernetes role in the previous step.

   Option 1: Using `eksctl`

   Run the following `eksctl` command to map the Amazon EMR on EKS service-linked role with the `emr-containers` user.

   ```
   eksctl create iamidentitymapping \
       --cluster my-cluster-name \
       --arn "arn:aws:iam::my-account-id:role/AWSServiceRoleForAmazonEMRContainers" \
       --username emr-containers
   ```

   Option 2: Without using eksctl

   1. Run the following command to open the `aws-auth` configuration map in text editor.

      ```
      kubectl edit -n kube-system configmap/aws-auth
      ```

      Note
      If you receive an error stating `Error from server (NotFound): configmaps "aws-auth" not found`, see the steps in Add user roles in the Amazon EKS User Guide to apply the stock ConfigMap.

   2. Add Amazon EMR on EKS service-linked role details to the `mapRoles` section of the `ConfigMap`, under `data`. Add this section if it does not already exist in the file. The updated `mapRoles` section under data looks like the following example.

      ```
      apiVersion: v1
      data:
        mapRoles: |
          - rolearn: arn:aws:iam::<your-account-id>:role/
      AWSServiceRoleForAmazonEMRContainers
            username: emr-containers
          - ... <other previously existing role entries, if there's any>.
      ```

   3. Save the file and exit your text editor.

Enable IAM Roles for Service Accounts (IRSA) on the EKS cluster

The IAM roles for service accounts feature is available on Amazon EKS versions 1.14 and later and for EKS clusters that are updated to versions 1.13 or later on or after September 3rd, 2019. To use this feature, you can update existing EKS clusters to version 1.14 or later. For more information, see Updating an Amazon EKS cluster Kubernetes version.

If your cluster supports IAM roles for service accounts, it has an OpenID Connect issuer URL associated with it. You can view this URL in the Amazon EKS console, or you can use the following AWS CLI command to retrieve it.

Important
You must use the latest version of the AWS CLI to receive the proper output from this command.

```
aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer" --
output text
```

The expected output is as follows.

```
https://oidc.eks.<region-code>.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

To use IAM roles for service accounts in your cluster, you must create an OIDC identity provider using either eksctl or the AWS Management Console.

To create an IAM OIDC identity provider for your cluster with `eksctl`

Check your `eksctl` version with the following command. This procedure assumes that you have installed `eksctl` and that your `eksctl` version is 0.32.0 or later.

```
eksctl version
```

For more information about installing or upgrading eksctl, see Installing or upgrading eksctl.

Create your OIDC identity provider for your cluster with the following command. Replace *cluster_name* with your own value.

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

To create an IAM OIDC identity provider for your cluster with the AWS Management Console

Retrieve the OIDC issuer URL from the Amazon EKS console description of your cluster, or use the following AWS CLI command.

Use the following command to retrieve the OIDC issuer URL from the AWS CLI.

```
aws eks describe-cluster --name <cluster_name> --query "cluster.identity.oidc.issuer" --
output text
```

Use the following steps to retrieve the OIDC issuer URL from the Amazon EKS console.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation panel, choose **Identity Providers**, and then choose **Create Provider**.
    1. For **Provider Type**, choose **Choose a provider type**, and then choose **OpenID Connect**.
    2. For **Provider URL**, paste the OIDC issuer URL for your cluster.
    3. For Audience, type sts.amazonaws.com and choose **Next Step**.
3. Verify that the provider information is correct, and then choose **Create** to create your identity provider.

Create a job execution role

You must create an IAM role to run workloads on Amazon EMR on EKS. We refer to this role as the *job execution role* in this documentation. For more information about how to create IAM roles, see Creating IAM roles in the IAM User Guide.

You must create an IAM policy that specifies the permissions for the job execution role and then attach the IAM policy to the job execution role. For more information, see Creating IAM Policies in the IAM User Guide.

The following policy for the job execution role allows access to resource targets, Amazon S3, and CloudWatch. These permissions are necessary to monitor jobs and access logs.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

For more information, see Using job execution roles, Configure a job run to use S3 logs, and Configure a job run to use CloudWatch Logs.

Update the trust policy of the job execution role

When you use IAM Roles for Service Accounts (IRSA) to run jobs on a Kubernetes namespace, an administrator must create a trust relationship between the job execution role and the identity of the

EMR managed service account. The trust relationship can be created by updating the trust policy of the job execution role. Note that the EMR managed service account is automatically created at job submission, scoped to the namespace where the job is submitted.

Run the following command to update the trust policy.

```
aws emr-containers update-role-trust-policy \
      --cluster-name cluster \
      --namespace namespace \
      --role-name iam_role_name_for_job_execution
```

For more information, see Using job execution roles with Amazon EMR on EKS (p. 65).

Important
The operator running the above command must have these permissions: `eks:DescribeCluster`, `iam:GetRole`, `iam:UpdateAssumeRolePolicy`.

Grant users access to Amazon EMR on EKS

For any actions that you perform on Amazon EMR on EKS, you need a corresponding IAM permission for that action. You must create an IAM policy that allows you to perform the Amazon EMR on EKS actions and attach the policy to the IAM user or role that you use.

This topic provides steps for creating a new policy and attaching it to an IAM user. It also covers the basic permissions that you need to set up your Amazon EMR on EKS environment. We recommend that you refine the permissions to specific resources whenever possible based on your business needs.

Creating a new IAM policy and attaching it to an IAM user in the IAM console

Create a new IAM policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane of the IAM console, choose **Policies**.
3. On the **Policies** page, choose **Create Policy**.
4. In the **Create Policy** window, navigate to the **Edit JSON** tab. Create a policy document with one or more JSON statements as shown in the examples following this procedure. Next, choose **Review policy**.
5. On the **Review Policy** screen, enter your **Policy Name**, for example `AmazonEMROnEKSPolicy`. Enter an optional description, and then choose **Create policy**.

Attach the policy to an IAM user or role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the check box next to the policy created in the previous section. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.

5. Choose the user or role to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user or role to attach the policy to, choose **Attach policy**.

Permissions for managing virtual clusters

To manage virtual clusters in your AWS account, create an IAM policy with the following permissions. These permissions allow you to create, list, describe, and delete virtual clusters in your AWS account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "emr-containers.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:CreateVirtualCluster",
                "emr-containers:ListVirtualClusters",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:DeleteVirtualCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

When the `CreateVirtualCluster` operation is invoked for the first time from an AWS account, you also need the `CreateServiceLinkedRole` permissions to create the service-linked role for Amazon EMR on EKS. For more information, see Using service-linked roles for Amazon EMR on EKS (p. 63).

Permissions for submitting jobs

To submit jobs on the virtual clusters in your AWS account, create an IAM policy with the following permissions. These permissions allow you to start, list, describe, and cancel job runs for the all virtual clusters in your account. You should consider adding permissions to list or describe virtual clusters, which allow you to check the state of the virtual cluster before submitting jobs.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:StartJobRun",
                "emr-containers:ListJobRuns",
                "emr-containers:DescribeJobRun",
                "emr-containers:CancelJobRun"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Permissions for debugging and monitoring

To get access to logs pushed to Amazon S3 and CloudWatch, or to view application event logs in the Amazon EMR console, create an IAM policy with the following permissions. We recommend that you refine the permissions to specific resources whenever possible based on your business needs.

Important
If you haven't created an Amazon S3 bucket, you need to add `s3:CreateBucket` permission to the policy statement. If you haven't created a log group, you need to add `logs:CreateLogGroup` to the policy statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:DescribeJobRun",
                "elasticmapreduce:CreatePersistentAppUI",
                "elasticmapreduce:DescribePersistentAppUI",
                "elasticmapreduce:GetPersistentAppUIPresignedURL"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:Get*",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about how to configure a job run to push logs to Amazon S3 and CloudWatch, see Configure a job run to use S3 logs and Configure a job run to use CloudWatch Logs.

Register the Amazon EKS cluster with Amazon EMR

Registering your cluster is the final required step to set up Amazon EMR on EKS to run workloads.

Use the following command to create a virtual cluster with a name of your choice for the Amazon EKS cluster and namespace that you set up in previous steps.

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
    "id": "cluster_name",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "namespace_name"
        }
    }
}'
```

Alternatively, you can create a JSON file that includes the required parameters for the virtual cluster and then run the `create-virtual-cluster` command with the path to the JSON file. For more information, see Managing virtual clusters (p. 49).

Note
To validate the successful creation of a virtual cluster, view the status of virtual clusters using the `list-virtual-clusters` operation or by going to the **Virtual Clusters** page in the Amazon EMR console.

Getting started

This topic helps you get started using Amazon EMR on EKS by deploying a Spark Python application on a virtual cluster.

Before you begin, make sure that you have completed the steps in Setting up (p. 5).

You will need the following information from the setup steps:

- Virtual cluster ID for the Amazon EKS cluster and Kubernetes namespace registered with Amazon EMR

  Important
  When creating an EKS cluster, make sure to use m5.xlarge as the instance type, or any other instance type with a higher CPU and memory. Using an instance type with lower CPU or memory than m5.xlarge may lead to job failure due to insufficient resources available in the cluster.
- Name of the IAM role used for job execution
- Release label for the Amazon EMR release (for example, `emr-6.3.0-latest`)
- Destination targets for logging and monitoring:
  - Amazon CloudWatch log group name and log stream prefix
  - Amazon S3 location to store event and container logs

Important
Amazon EMR on EKS jobs use Amazon CloudWatch and Amazon S3 as destination targets for monitoring and logging. You can monitor job progress and troubleshoot failures by viewing the job logs sent to these destinations. To enable logging, the IAM policy associated with the IAM role for job execution must have the required permissions to access the target resources. If the IAM policy doesn't have the required permissions, you must follow the steps outlined in Update the trust policy of the job execution role (p. 15), Configure a job run to use Amazon S3 logs, and Configure a job run to use CloudWatch Logs before running this sample job.

Run a Spark Python application

Take the following steps to run a simple `wordcount.py` Spark Python application on Amazon EMR on EKS. The application `entryPoint` file is located at `s3://`*`REGION`*`.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py`. The *`REGION`* is the Region in which your Amazon EMR on EKS virtual cluster resides, such as *`us-east-1`*.

1. Update the IAM policy for the job execution role with the required permissions, as the following policy statements demonstrate.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadFromLoggingAndInputScriptBuckets",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::*.elasticmapreduce",
                "arn:aws:s3:::*.elasticmapreduce/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*"
            ]
        },
        {
            "Sid": "WriteToLoggingAndOutputDataBuckets",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*"
            ]
        },
        {
            "Sid": "DescribeAndCreateCloudwatchLogStream",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Sid": "WriteToCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

- The first statement `ReadFromLoggingAndInputScriptBuckets` in this policy grants `ListBucket` and `GetObjects` access to the following Amazon S3 buckets:
  - *REGION*`.elasticmapreduce` - the bucket where the application `entryPoint` file is located.
  - *DOC-EXAMPLE-BUCKET-OUTPUT* - a bucket that you define for your output data.
  - *DOC-EXAMPLE-BUCKET-LOGGING* - a bucket that you define for your logging data.
- The second statement `WriteToLoggingAndOutputDataBuckets` in this policy grants the job with permissions to write data to your output and logging buckets respectively.
- The third statement `DescribeAndCreateCloudwatchLogStream` grants the job with permissions to describe and create Amazon CloudWatch Logs.
- The fourth statement `WriteToCloudwatchLogs` grants permissions to write logs to an Amazon CloudWatch log group named *my_log_group_name* under a log stream named *my_log_stream_prefix*.

2. Initiate the sample application using the following command. Replace all the replaceable *red italicized* values with appropriate values. The *REGION* is the Region in which your Amazon EMR on EKS virtual cluster resides, such as *us-east-1*.

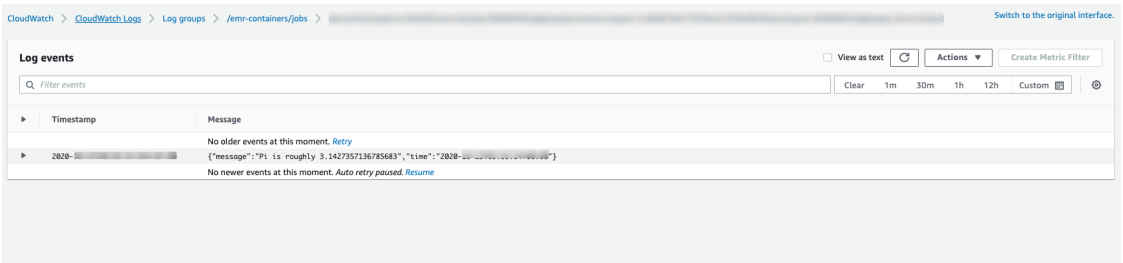
```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.3.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py",
    "entryPointArguments": ["s3://DOC-EXAMPLE-BUCKET-OUTPUT/wordcount_output"],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://DOC-EXAMPLE-BUCKET-LOGGING"
    }
  }
}'
```

The output data from this job will be available at `s3://DOC-EXAMPLE-BUCKET-OUTPUT/wordcount_output`.

You can also create a JSON file with specified parameters for your job run. Then run the `start-job-run` command with a path to the JSON file. For more information, see Submit a job run (p. 30). For more details about configuring job run parameters, see Options for configuring a job run (p. 31).

3. To monitor the progress of the job or to debug failures, you can inspect logs uploaded to Amazon S3, CloudWatch Logs, or both. Refer to log path in Amazon S3 at Configure a job run to use S3 logs and for Cloudwatch logs at Configure a job run to use CloudWatch Logs. To see logs in CloudWatch Logs, follow the instructions below.

   - Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
   - In the **Navigation** pane, choose **Logs**. Then choose **Log groups**.
   - Choose the log group for Amazon EMR on EKS and then view the uploaded log events.



Customizing Docker images for Amazon EMR on EKS

You can use customized Docker images with Amazon EMR on EKS. Customizing the Amazon EMR on EKS runtime image provides the following benefits:

- Package application dependencies and runtime environment into a single immutable container that promotes portability and simplifies dependency management for each workload.
- Install and configure packages that are optimized to your workloads. These packages may not be widely available in the public distribution of Amazon EMR runtimes.
- Integrate Amazon EMR on EKS with current established build, test, and deployment processes within your organization, including local development and testing.
- Apply established security processes, such as image scanning, that meet compliance and governance requirements within your organization.

Topics

How to customize Docker images

Take the following steps to customize Docker images for Amazon EMR on EKS.

Prerequisites

- Complete the Setting up (p. 5) steps for Amazon EMR on EKS.
- Install Docker in your environment. For more information, see Get Docker.

Step 1: Retrieve a base image from Amazon Elastic Container Registry (Amazon ECR)

Take the following steps to retrieve an Amazon EMR on EKS base image from Amazon ECR. The base image contains the Amazon EMR runtime and connectors used to access other AWS services.

1. Choose a base image URI. The image URI follows this format, *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/spark/*container-image-tag*, as the following example demonstrates.

   ```
   895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-5.32.0-20210129
   ```

   To choose a base image in your Region, see How to select a base image URI (p. 28).
2. Log in to the Amazon ECR repository where the base image is stored. Replace *895885662937* and *us-west-2* with the Amazon ECR registry account and the AWS Region you have selected.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-
stdin 895885662937.dkr.ecr.us-west-2.amazonaws.com
```

3. Pull the base image into your local Workspace. Replace *emr-6.2.0-20210129* with the container image tag you have selected.

```
docker pull 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.2.0-20210129
```

Step 2: Customize a base image

Take the following steps to customize the base image you have pulled from Amazon ECR.

1. Create a new `Dockerfile` on your local Workspace.
2. Edit the `Dockerfile` you just created and add the following content. This `Dockerfile` uses the container image you have pulled from `895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.2.0-20210129`.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.2.0-20210129
USER root
### Add customization commands here ####
USER hadoop:hadoop
```

3. Add commands in the `Dockerfile` to customize the base image. For example, add a command to install Python libraries, as the following `Dockerfile` demonstrates.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.2.0-20210129
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. From the same directory where the `Dockerfile` is created, run the following command to build the Docker image. Provide a name for the Docker image, for example, *emr6.2_custom*.

```
docker build -t emr6.2_custom .
```

Step 3: (Optional but recommended) Validate a custom image

We recommend that you test the compatibility of your custom image before publishing it. You can use the Amazon EMR on EKS custom image CLI to check if your image has the required file structures and correct configurations for running on Amazon EMR on EKS.

> **Note**
> The Amazon EMR on EKS custom image CLI cannot confirm that your image is free of error. Use caution when removing dependencies from the base images.

Take the following steps to validate your custom image.

1. Download and install Amazon EMR on EKS custom image CLI. For more information, see Amazon EMR on EKS custom image CLI Installation Guide.
2. Run the following command to test the installation.

```
emr-on-eks-custom-image --version
```

The following shows an example of the output.

```
Amazon EMR on EKS Custom Image CLI
Version: x.xx
```

3. Run the following command to validate your custom image.

```
emr-on-eks-custom-image validate-image -i image_name -r release_version [-t image_type]
```

- `-i` specifies the local image URI that needs to be validated. This can be the image URI, any name or tag that you defined for your image.
- `-r` specifies the exact release version for the base image, for example, `emr-5.32.0`.
- `-t` specifies the image type. If this is a Spark image, input `spark`. The default value is `spark`. The current Amazon EMR on EKS custom image CLI version only supports Spark runtime images.

If you run the command successfully and the custom image meets all the required configurations and file structures, the returned output displays the results of all of the tests, as the following example demonstrates.

```
Amazon EMR on EKS Custom Image Test
Version: x.xx
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: xxx
[INFO] Created On: 2021-05-17T20:50:07.986662904Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to /home/hadoop : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] SPARK_HOME is set with value: /usr/lib/spark : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] File Structure Test for spark-jars in /usr/lib/spark/jars: PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for bin-files in /usr/bin: PASS
... Start Running Sample Spark Job
[INFO] Sample Spark Job Test with local:///usr/lib/spark/examples/jars/spark-
examples.jar : PASS
----------------------------------------------------------------
Overall Custom Image Validation Succeeded.
----------------------------------------------------------------
```

If the custom image doesn't meet the required configurations or file structures, error messages occur. The returned output provides information about the incorrect configurations or file structures.

Step 4: Publish a custom image

Publish the new Docker image to your Amazon ECR registry.

1. Run the following command to create an Amazon ECR repository for storing your Docker image. Provide a name for your repository, for example, *emr6.2_custom_repo*. Replace *us-west-2* with your Region.

```
aws ecr create-repository \
    --repository-name emr6.2_custom_repo \
    --image-scanning-configuration scanOnPush=true \
    --region us-west-2
```

For more information, see Create a repository in the *Amazon ECR User Guide*.

2. Run the following command to authenticate to your default registry.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
```

For more information, see Authenticate to your default registry in the *Amazon ECR User Guide*.

3. Tag and publish an image to the Amazon ECR repository you created.

Tag the image.

```
docker tag emr6.2_custom aws_account_id.dkr.ecr.us-
west-2.amazonaws.com/emr6.2_custom_repo
```

Push the image.

```
docker push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/emr6.2_custom_repo
```

For more information, see Push an image to Amazon ECR in the *Amazon ECR User Guide*.

Step 5: Submit a Spark workload in Amazon EMR using a custom image

After a custom image is built and published, you can submit an Amazon EMR on EKS job using a custom image.

First, create a start-job-run-request.json file and specify `spark.kubernetes.container.image` parameter to reference the custom image, as the following example JSON file demonstrates.

Note
You can use `local://` scheme to refer to files available in the custom image as shown with `entryPoint` argument in the JSON snippet below. You can also use the `local://` scheme to refer to application dependencies. All files and dependencies that are referred using `local://` scheme must already be present at the specified path in the custom image.

```
{
    "name": "spark-custom-image",
    "virtualClusterId": "virtual-cluster-id",
    "executionRoleArn": "execution-role-arn",
    "releaseLabel": "emr-6.2.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
        "entryPointArguments": [
                  "10"
              ],
```

```
        "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.kubernetes.container.image=123456789012.dkr.ecr.us-west-2.amazonaws.com/
emr6.2_custom_repo"
       }
    }
}
```

You can also reference the custom image by using `applicationConfiguration` properties as the following example demonstrates.

```
{
    "name": "spark-custom-image",
    "virtualClusterId": "virtual-cluster-id",
    "executionRoleArn": "execution-role-arn",
    "releaseLabel": "emr-6.2.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
        "entryPointArguments": [
                  "10"
             ],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
       }
    },
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "spark-defaults",
                "properties": {
                    "spark.kubernetes.container.image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/emr6.2_custom_repo"
                }
            }
        ]
    }
}
```

Then run the `start-job-run` command to submit the job.

```
aws emr-containers start-job-run --cli-input-json file://./start-job-run-request.json
```

In the JSON examples above, replace *emr-6.2.0-latest* with your Amazon EMR release version. In the Step 1: Retrieve a base image from Amazon Elastic Container Registry (Amazon ECR) (p. 23), the selected base image tag is *emr-6.2.0-20210129*, so the corresponding Amazon EMR release version can be either *emr-6.2.0-latest* or *emr-6.2.0-20210129*. It's strongly recommended you use the `-latest` release version to ensure that the selected version contains the latest security updates. For more information about Amazon EMR release versions and corresponding image tags, see How to select a base image URI (p. 28).

Note
You can use `spark.kubernetes.driver.container.image` and `spark.kubernetes.executor.container.image` to specify a different image for driver and executor pods.

How to select a base image URI

The base Docker images you can select are stored in Amazon Elastic Container Registry (Amazon ECR). The image URI follows this format: *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/spark/*container-image-tag*, as the following example demonstrates..

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-5.32.0-20210129
```

Use the information in the following table to select a container image tag for your base image.

Note
Beginning with Amazon EMR release version 6.3.0, you can use an image tag with `:latest` to automatically get the latest base images in each build. If you are using an older Amazon EMR release, make sure to use the base image tag with the latest date. For example, `emr-6.2.0-20210129`.

Releases	Amazon EMR on EKS release versions	Container image tags
Amazon EMR 6.3.0 releases (p. 88)	emr-6.3.0-latest (p. 89)	emr-6.3.0:latest
	emr-6.3.0-20210429 (p. 89)	emr-6.3.0:20210429
Amazon EMR 6.2.0 releases (p. 89)	emr-6.2.0-latest (p. 90)	emr-6.2.0-20210129
	emr-6.2.0-20210129 (p. 90)	emr-6.2.0-20210129
	emr-6.2.0-20201218 (p. 90)	emr-6.2.0-20201218
	emr-6.2.0-20201201 (p. 91)	emr-6.2.0-20201201
Amazon EMR 5.33.0 releases (p. 91)	emr-5.33.0-latest (p. 91)	emr-5.33.0-20210323
	emr-5.33.0-20210323 (p. 92)	emr-5.33.0-20210323
Amazon EMR 5.32.0 releases (p. 92)	emr-5.32.0-latest (p. 92)	emr-5.32.0-20210129
	emr-5.32.0-20210129 (p. 93)	emr-5.32.0-20210129
	emr-5.32.0-20201218 (p. 93)	emr-5.32.0-20201218
	emr-5.32.0-20201201 (p. 93)	emr-5.32.0-20201201

To avoid high network latency, you should pull a base image from your closest AWS Region. Select the Amazon ECR registry account corresponding to the Region you are pulling the image from based on the following table.

Regions	Amazon ECR registry accounts	
ap-northeast-1	059004520145	
ap-northeast-2	996579266876	
ap-south-1	235914868574	
ap-southeast-1	671219180197	
ap-southeast-2	038297999601	

Regions	Amazon ECR registry accounts	
ca-central-1	351826393999	
eu-central-1	107292555468	
eu-north-1	830386416364	
eu-west-1	483788554619	
eu-west-2	118780647275	
eu-west-3	307523725174	
sa-east-1	052806832358	
us-east-1	755674844232	
us-east-2	711395599931	
us-west-1	608033475327	
us-west-2	895885662937	

Considerations

By customizing Docker images, you can choose the exact runtime for your job at a granular level. Make sure to follow these best practices when using this feature:

- Security is a shared responsibility between AWS and you. You are responsible for the security patching of the binaries you add to the image. Follow the Security best practices (p. 72), especially Get the latest security updates for custom images (p. 72) and Apply principle of least privilege (p. 72).
- Beginning with Amazon EMR release version 6.3.0, you can use an image tag with `:latest` to automatically get the latest base images in each build. If you are using an older Amazon EMR release, make sure to use the base image tag with the latest date. For example, `emr-6.2.0-20210129`.
- When you customize a base image, make sure to change the Docker user to `hadoop:hadoop` to ensure that the jobs are not run using the root user.
- Amazon EMR on EKS mounts files on top of the image's configurations, such as the `spark-defaults.conf`, at runtime. To override these configuration files, we recommend that you use the `applicationOverrides` parameter during the job submission and not only modify the files directly in the custom image.
- Amazon EMR on EKS mounts certain folders at runtime. Any modifications made to these folders are not available in the container. If you want to add an application or its dependencies for custom images, we recommend that you choose a directory that is not part of the following predefined paths:
  - `/var/log/fluentd`
  - `/var/log/spark/user`
  - `/var/log/spark/apps`
  - `/mnt`
  - `/tmp`
  - `/home/hadoop`
- Your customized image can be uploaded on any Docker compatible repository, such as Amazon ECR, Docker Hub, or a private enterprise repository. For more information about configuring the Amazon EKS cluster authentication with the selected Docker repository, see Pull an Image from a Private Registry.

Job runs

A job run is a unit of work, such as a Spark jar, PySpark script, or SparkSQL query, that you submit to Amazon EMR on EKS. This topic provides an overview of managing job runs using the AWS CLI, viewing job runs using the Amazon EMR console, and troubleshooting common job run errors.

Note
Before submitting a job run, you must first complete the steps in Setting up (p. 5).

Topics

Managing job runs using the AWS CLI

This topic covers how to manage job runs using the AWS Command Line Interface (AWS CLI).

Topics

Submit a job run

To submit a job run using a JSON file with specified parameters

1. Create a `start-job-run-request.json` file and specify the required parameters for your job run, as the following example JSON file demonstrates. For more information about the parameters, see Options for configuring a job run (p. 31).

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.2.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["arguments_list"],
```

```
      "sparkSubmitParameters": "--class <main_class> --conf spark.executor.instances=2
 --conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

2. Use the `start-job-run` command with a path to the `start-job-run-request.json` file stored locally or in Amazon S3.

```
aws emr-containers start-job-run \
--cli-input-json file://./start-job-run-request.json
```

To start a job run using the `start-job-run` command

- Supply all the specified parameters in the `StartJobRun` command, as the following example demonstrates.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": [arguments_list], "sparkSubmitParameters": "--class
 <main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification": "spark-
defaults", "properties": {"spark.driver.memory": "2G"}}], "monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName": "log_group_name",
 "logStreamNamePrefix": "log_stream_prefix"}, "persistentAppUI":"ENABLED",
 "s3MonitoringConfiguration": {"logUri": "s3://my_s3_log_location" }}}'
```

Options for configuring a job run

Use the following options to configure job run parameters:

- `--execution-role-arn`: You must provide an IAM role that is used for running jobs. For more information, see Using job execution roles with Amazon EMR on EKS (p. 65).

- `--release-label`: You can deploy Amazon EMR on EKS with Amazon EMR versions 5.32.0 and 6.2.0 and later. Amazon EMR on EKS is not supported in previous Amazon EMR release versions. For more information, see Amazon EMR on EKS release versions (p. 88).
- `--job-driver`: Job driver is used to provide input on the main job. This is a union type field where you can only pass one of the values for the job type that you want to run. Supported job types include:
  - Spark submit jobs - Used to run a command through Spark submit. You can use this job type to run Scala, PySpark, SparkR, SparkSQL and any other supported jobs through Spark Submit. This job type has the following parameters:
    - Entrypoint - This is the HCFS (Hadoop compatible file system) reference to the main jar/py file you want to run.
    - Entrypoint Args - This is the argument you want to pass to your JAR. You should handle reading this parameter using your entrypoint code.
    - SparkSubmitParameters - These are the additional spark parameters you want to send to the job. Use this parameter to override default Spark properties such as driver memory or number of executors like —conf or —class. For additional information, see Launching Applications with spark-submit.
- `--configuration-overrides`: You can override the default configurations for applications by supplying a configuration object. You can use a shorthand syntax to provide the configuration or you can reference the configuration object in a JSON file. Configuration objects consist of a classification, properties, and optional nested configurations. Properties consist of the settings you want to override in that file. You can specify multiple classifications for multiple applications in a single JSON object. The configuration classifications that are available vary by Amazon EMR release version. For a list of configuration classifications that are available for each release version of Amazon EMR, see Amazon EMR on EKS release versions (p. 88).

  If you pass the same configuration in an application override and in Spark submit parameters, the Spark submit parameters take precedence. The complete configuration priority list follows, in order of highest priority to lowest priority.
  - Configuration supplied when creating `SparkSession`.
  - Configuration supplied as part of `sparkSubmitParameters` using `—conf`.
  - Configuration provided as part of application overrides.
  - Optimized configurations chosen by Amazon EMR for the release.
  - Default open source configurations for the application.

  To monitor job runs using Amazon CloudWatch or Amazon S3, you must provide the configuration details for CloudWatch. For more information, see Configure a job run to use S3 logs (p. 33) and Configure a job run to use Amazon CloudWatch Logs (p. 33). If the S3 bucket or CloudWatch log group does not exist, then Amazon EMR creates it before uploading logs to the bucket.
- For an additional list of Kubernetes configuration options, see Spark Properties on Kubernetes.

  The following Spark configurations are not supported.
  - spark.kubernetes.authenticate.driver.serviceAccountName
  - spark.kubernetes.authenticate.executor.serviceAccountName
  - spark.kubernetes.container.image
  - spark.kubernetes.namespace
  - spark.kubernetes.driver.pod.name
  - spark.kubernetes.container.image.pullPolicy

Configure a job run to use S3 logs

To be able to monitor the job progress and to troubleshoot failures, you must configure your jobs to send log information to Amazon S3, Amazon CloudWatch Logs, or both. This topic helps you get started publishing application logs to Amazon S3 on your jobs that are launched with Amazon EMR on EKS.

S3 logs IAM policy

Before your jobs can send log data to Amazon S3, the following permissions must be included in the permissions policy for the job execution role. Replace *DOC-EXAMPLE-BUCKET-LOGGING* with the name of your logging bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*",
            ]
        }
    ]
}
```

Note
Amazon EMR on EKS can also create an Amazon S3 bucket. If an Amazon S3 bucket is not available, include the "`s3:CreateBucket`" permission in the IAM policy.

After you have given your execution role the proper permissions to send logs to Amazon S3, your log data are sent to the following Amazon S3 locations when `s3MonitoringConfiguration` is passed in the `monitoringConfiguration` section of a `start-job-run` request, as shown in Managing job runs using the AWS CLI (p. 30).

- Controller Logs - /*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*pod-name*/(stderr.gz/stdout.gz)
- Driver Logs - /*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/spark-*job-id*-driver/(stderr.gz/stdout.gz)
- Executor Logs - /*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/*executor-pod-name*/(stderr.gz/stdout.gz)

Configure a job run to use Amazon CloudWatch Logs

To monitor job progress and to troubleshoot failures, you must configure your jobs to send log information to Amazon S3, Amazon CloudWatch Logs, or both. This topic helps you get started using CloudWatch Logs on your jobs that are launched with Amazon EMR on EKS. For more information about CloudWatch Logs, see Monitoring Log Files in the Amazon CloudWatch User Guide.

CloudWatch Logs IAM policy

For your jobs to send log data to CloudWatch Logs, the following permissions must be included in the permissions policy for the job execution role. Replace *my_log_group_name* and

my_log_stream_prefix with names of your CloudWatch log group and log stream names, respectively. Amazon EMR on EKS creates the log group and log stream if they do not exist as long as the execution role ARN has appropriate permissions.

```
{
     "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

Note
Amazon EMR on EKS can also create a log stream. If a log stream does not exist, the IAM policy should include the `"logs:CreateLogGroup"` permission.

After you have given your execution role the proper permissions, your application sends its log data to CloudWatch Logs when `cloudWatchMonitoringConfiguration` is passed in the `monitoringConfiguration` section of a `start-job-run` request, as shown in Managing job runs using the AWS CLI (p. 30).

In the `StartJobRun` API, *log_group_name* is the log group name for CloudWatch, and *log_stream_prefix* is the log stream name prefix for CloudWatch. You can view and search these logs in the AWS Management Console.

- Controller logs - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*pod-name*/(stderr/stdout)
- Driver logs - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/spark-*job-id*-driver/(stderrstdout)
- Executor logs - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/*executor-pod-name*/(stderr/stdout)

List job runs

You can run `list-job-run` to show the states of job runs, as the following example demonstrates.

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

Describe a job run

You can run `describe-job-run` to get more details about the job, such as job state, state details, and job name, as the following example demonstrates.

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Cancel a job run

You can run `cancel-job-run` to cancel running jobs, as the following example demonstrates.

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Job run states

When you submit a job run to an Amazon EMR on EKS job queue, the job run enters the `PENDING` state. It then passes through the following states until it succeeds (exits with code `0`) or fails (exits with a non-zero code).

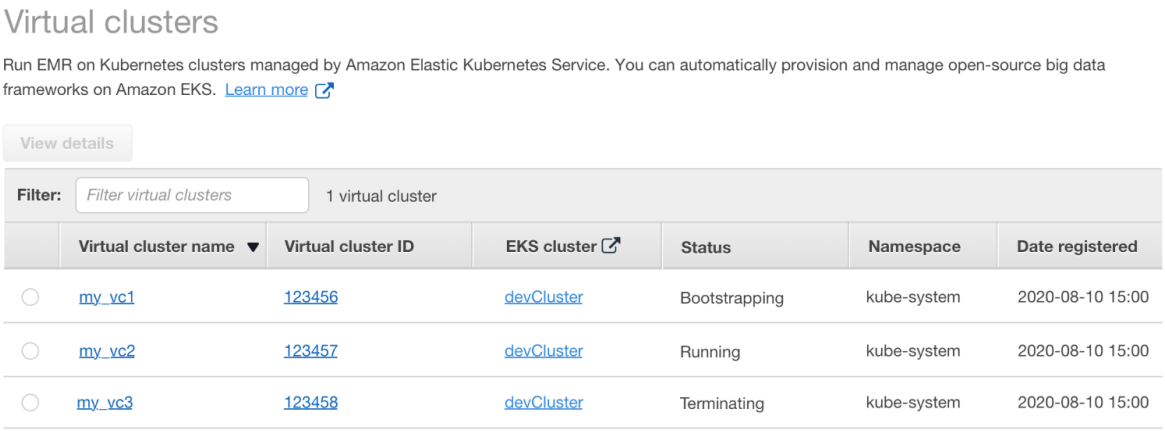
Job runs can have the following states:

- `PENDING` - The initial job state when the job run is submitted to Amazon EMR on EKS. The job is waiting to be submitted to the virtual cluster, and Amazon EMR on EKS is working on submitting this job.
- `SUBMITTED` - A job run that has been successfully submitted to the virtual cluster. The cluster scheduler then tries to run this job on the cluster.
- `RUNNING` - A job run that is running in the virtual cluster. In Spark applications, this means that the Spark driver process is in the `running` state.
- `FAILED` - A job run that failed to be submitted to the virtual cluster or that completed unsuccessfully. Look at StateDetails and FailureReason to find additional information about this job failure.
- `COMPLETED` - A job run that has completed successfully.
- `CANCEL_PENDING` - A job run has been requested for cancellation. Amazon EMR on EKS is trying to cancel the job on the virtual cluster.
- `CANCELLED` - A job run that was cancelled successfully.

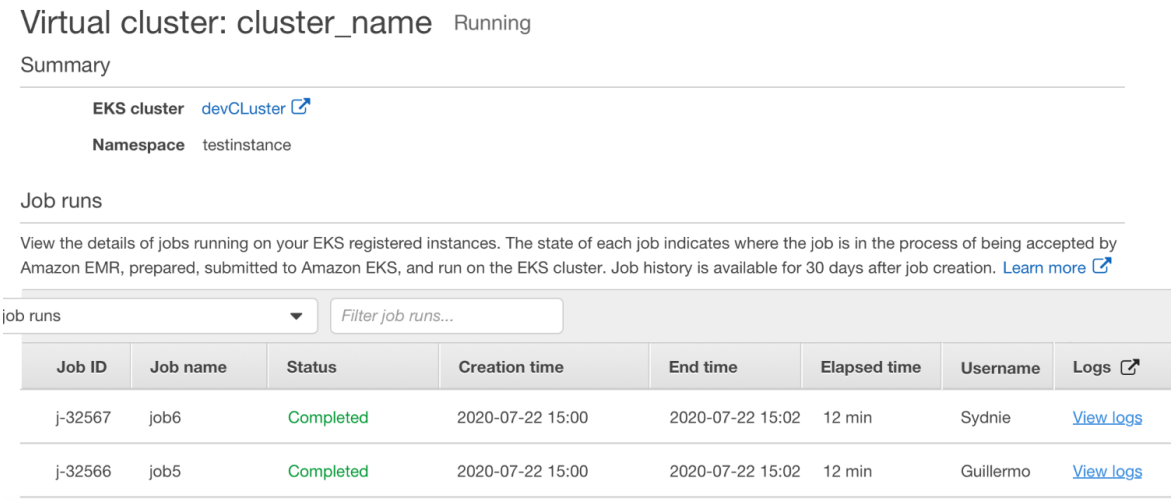
Viewing jobs in the Amazon EMR console

To view jobs in the Amazon EMR console, under **EMR on EKS**, choose **Virtual clusters**.

From the list of virtual clusters, select the virtual cluster for which you want to view logs.



On the **Job runs** table, select **View logs** to view the details of a job run.



Then open the application user interface to view the details of jobs running on Amazon EMR on EKS.

Note
Support for the one-click experience is enabled by default. It can be turned off by setting `persistentAppUI` to `DISABLED` in `monitoringConfiguration` during job submission. For more information, see View Persistent Application User Interfaces.

Common errors when running jobs

The following errors may occur when you run `StartJobRun` API.

Error Message	Error Condition	Recommended Next Step
error: argument --*argument* is required	Required parameters are missing.	Add the missing arguments to the API request.
An error occurred (AccessDeniedException) when calling the StartJobRun operation: User: *ARN* is not	Execution role is missing.	See Using Using job execution roles with Amazon EMR on EKS (p. 65).

Error Message	Error Condition	Recommended Next Step
authorized to perform: emr-containers:StartJobRun		
An error occurred (AccessDeniedException) when calling the StartJobRun operation: User: *ARN* is not authorized to perform: emr-containers:StartJobRun	Caller doesn't have permission to the execution role [valid / not valid format] via condition keys.	See Using job execution roles with Amazon EMR on EKS (p. 65).
An error occurred (AccessDeniedException) when calling the StartJobRun operation: User: *ARN* is not authorized to perform: emr-containers:StartJobRun	Job submitter and Execution role ARN are from different accounts.	Ensure that job submitter and execution role ARN are from the same AWS account.
1 validation error detected: Value *Role* at 'executionRoleArn' failed to satisfy the ARN regular expression pattern: ^arn:(aws[a-zA-Z0-9-]*):iam::(\d{12})?:(role((\u002F)|(\u002F[\u0021-\u007F]+\u002F))[\w+=,.@-]+)	Caller has permissions for the execution role via condition keys, but the role does not satisfy the constraints of ARN format.	Provide the execution role following the ARN format. See Using job execution roles with Amazon EMR on EKS (p. 65).
An error occurred (ResourceNotFoundException) when calling the StartJobRun operation: Virtual cluster *Virtual Cluster ID* doesn't exist.	Virtual cluster ID is not found.	Provide a virtual cluster ID registered with Amazon EMR on EKS.
An error occurred (ValidationException) when calling the StartJobRun operation: Virtual cluster state *state* is not valid to create resource JobRun.	Virtual cluster is not ready to execute job.	See Virtual cluster states (p. 50).
An error occurred (ResourceNotFoundException) when calling the StartJobRun operation: Release *release* doesn't exist.	The release specified in job submission is incorrect.	See Amazon EMR on EKS release versions (p. 88).

Error Message	Error Condition	Recommended Next Step
An error occurred (AccessDeniedException) when calling the StartJobRun operation: User: *ARN* is not authorized to perform: emr-containers:StartJobRun on resource: *ARN* with an explicit deny. An error occurred (AccessDeniedException) when calling the StartJobRun operation: User: *ARN* is not authorized to perform: emr-containers:StartJobRun on resource: *ARN*	User is not authorized to call StartJobRun.	See Using job execution roles with Amazon EMR on EKS (p. 65).
An error occurred (ValidationException) when calling the StartJobRun operation: configurationOverrides.monitoringConfiguration.s3MonitoringConfiguration.logUri failed to satisfy constraint : %s	S3 path URI syntax is not valid.	logUri should be in the format of s3://...

The following errors may occur when you run `DescribeJobRun` API before the job runs.

Error Message	Error Condition	Recommended Next Step
stateDetails: JobRun submission failed. Classification *classification* not supported. failureReason: VALIDATION_ERROR state: FAILED.	Parameters in StartJobRun are not valid.	See Amazon EMR on EKS release versions (p. 88).
stateDetails: Cluster *EKS Cluster ID* does not exist. failureReason: CLUSTER_UNAVAILABLE state: FAILED	The EKS cluster is not available.	Check if the EKS cluster exists and has the right permissions. For more information, see Setting up (p. 5).
stateDetails: Cluster *EKS Cluster ID* does not have sufficient permissions. failureReason: CLUSTER_UNAVAILABLE state: FAILED	Amazon EMR does not have permissions to access the EKS cluster.	Verify that permissions are set up for Amazon EMR on the registered namespace. For more information, see Setting up (p. 5).

Error Message	Error Condition	Recommended Next Step
stateDetails: Cluster *EKS Cluster ID* is currently not reachable. failureReason: CLUSTER_UNAVAILABLE state: FAILED	EKS cluster is not reachable.	Check if EKS Cluster exists and has the right permissions. For more information, see Setting up (p. 5).
stateDetails: JobRun submission failed due to an internal error. failureReason: INTERNAL_ERROR state: FAILED	An internal error has occurred with the EKS cluster.	N/A
stateDetails: Cluster *EKS Cluster ID* does not have sufficient resources. failureReason: USER_ERROR state: FAILED	There are insufficient resources in the EKS cluster to run the job.	Add more capacity to the EKS node group or set up EKS Autoscaler. For more information, see Cluster Autoscaler.

The following errors may occur when you run `DescribeJobRun` API after the job runs.

Error Message	Error Condition	Recommended Next Step
stateDetails: Trouble monitoring your JobRun. Cluster *EKS Cluster ID* does not exist. failureReason: CLUSTER_UNAVAILABLE state: FAILED	The EKS cluster does not exist.	Check if EKS Cluster exists and has the right permissions. For more information, see Setting up (p. 5).
stateDetails: Trouble monitoring your JobRun. Cluster *EKS Cluster ID* does not have sufficient permissions. failureReason: CLUSTER_UNAVAILABLE state: FAILED	Amazon EMR does not have permissions to access the EKS cluster.	Verify that permissions are set up for Amazon EMR on the registered namespace. For more information, see Setting up (p. 5).
stateDetails: Trouble monitoring your JobRun. Cluster *EKS Cluster ID* is currently not reachable.	The EKS cluster is not reachable.	Check if EKS Cluster exists and has the right permissions. For more information, see Setting up (p. 5).

Error Message	Error Condition	Recommended Next Step
failureReason: CLUSTER_UNAVAILABLE state: FAILED		
stateDetails: Trouble monitoring your JobRun due to an internal error failureReason: INTERNAL_ERROR state: FAILED	An internal error has occurred and is preventing JobRun monitoring.	N/A

Using pod templates

Beginning with Amazon EMR versions 5.33.0 or 6.3.0, Amazon EMR on EKS supports Spark's pod template feature. A pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. Pod templates are specifications that determine how to run each pod. You can use pod template files to define the driver or executor pod's configurations that Spark configurations do not support. For more information about the Spark's pod template feature, see Pod Template.

Note
The pod template feature only works with driver and executor pods. You cannot configure job controller pods using the pod template.

Common scenarios

You can define how to run Spark jobs on shared EKS clusters by using pod templates with Amazon EMR on EKS and save costs and improve resource utilization and performance.

- To reduce costs, you can schedule Spark driver tasks to run on Amazon EC2 On-Demand Instances while scheduling Spark executor tasks to run on Amazon EC2 Spot Instances.
- To increase resource utilization, you can support multiple teams running their workloads on the same EKS cluster. Each team will get a designated Amazon EC2 node group to run their workloads on. You can use pod templates to apply a corresponding toleration to their workload.
- To improve monitoring, you can run a separate logging container to forward logs to your existing monitoring application.

For example, the following pod template file demonstrates a common usage scenario.

```
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: spark-kubernetes-driver # This will be interpreted as driver Spark main container
    env:
```

```
    - name: RANDOM
      value: "random"
  volumeMounts:
    - name: shared-volume
      mountPath: /var/data
    - name: metrics-files-volume
      mountPath: /var/metrics/data
- name: custom-side-car-container # Sidecar container
  image: <side_car_container_image>
  env:
    - name: RANDOM_SIDECAR
      value: random
  volumeMounts:
    - name: metrics-files-volume
      mountPath: /var/metrics/data
  command:
    - /bin/sh
    - '-c'
    -  <command-to-upload-metrics-files>
initContainers:
- name: spark-init-container-driver # Init container
  image: <spark-pre-step-image>
  volumeMounts:
    - name: source-data-volume # Use EMR predefined volumes
      mountPath: /var/data
  command:
    - /bin/sh
    - '-c'
    -  <command-to-download-dependency-jars>
```

The pod template completes the following tasks:

- Add a new init container that is executed before the Spark main container starts. The init container shares the EmptyDir volume called `source-data-volume` with the Spark main container. You can have your init container run initialization steps, such as downloading dependencies or generating input data. Then the Spark main container consumes the data.
- Add another sidecar container that is executed along with the Spark main container. The two containers are sharing another `EmptyDir` volume called `metrics-files-volume`. Your Spark job can generate metrics, such as Prometheus metrics. Then the Spark job can put the metrics into a file and have the sidecar container upload the files to your own BI system for future analysis.
- Add a new environment variable to the Spark main container. You can have your job consume the environment variable.
- Define a node selector, so that the pod is only scheduled on the `emr-containers-nodegroup` node group. This helps to isolate compute resources across jobs and teams.

Enabling pod templates with Amazon EMR on EKS

To enable the pod template feature with Amazon EMR on EKS, configure the Spark properties `spark.kubernetes.driver.podTemplateFile` and `spark.kubernetes.executor.podTemplateFile` to point to the pod template files in Amazon S3. Spark then downloads the pod template file and uses it to construct driver and executor pods.

> **Note**
> Spark uses the job execution role to load the pod template, so the job execution role must have permissions to access Amazon S3 to load the pod templates. For more information, see Create a job execution role (p. 15).

You can use the `SparkSubmitParameters` to specify the Amazon S3 path to the pod template, as the following job run JSON file demonstrates.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["arguments_list"],
       "sparkSubmitParameters": "--class <main_class> \
        --conf spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
        --conf
 spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  }
}
```

Alternatively, you can use the `configurationOverrides` to specify the Amazon S3 path to the pod template, as the following job run JSON file demonstrates.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["arguments_list"],
       "sparkSubmitParameters": "--class <main_class> \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G",
          "spark.kubernetes.driver.podTemplateFile":"s3://path_to_driver_pod_template",
          "spark.kubernetes.executor.podTemplateFile":"s3://path_to_executor_pod_template"
         }
      }
    ]
  }
}
```

Note

1. You need to follow the security guidelines when using the pod template feature with Amazon EMR on EKS, such as isolating untrusted application code. For more information, see Security best practices (p. 72).
2. You cannot change the Spark main container names by using `spark.kubernetes.driver.podTemplateContainerName` and `spark.kubernetes.executor.podTemplateContainerName`, because these names

are hardcoded as `spark-kubernetes-driver` and `spark-kubernetes-executors`. If you want to customize the Spark main container, you must specify the container in a pod template with these hardcoded names.

Pod template fields

Consider the following field restrictions when configuring a pod template with Amazon EMR on EKS.

- Amazon EMR on EKS allows only the following fields in a pod template to enable proper job scheduling.

  These are the allowed pod level fields:
  - `apiVersion`
  - `kind`
  - `metadata`
  - `spec.activeDeadlineSeconds`
  - `spec.affinity`
  - `spec.containers`
  - `spec.enableServiceLinks`
  - `spec.ephemeralContainers`
  - `spec.hostAliases`
  - `spec.hostname`
  - `spec.imagePullSecrets`
  - `spec.initContainers`
  - `spec.nodeName`
  - `spec.nodeSelector`
  - `spec.overhead`
  - `spec.preemptionPolicy`
  - `spec.priority`
  - `spec.priorityClassName`
  - `spec.readinessGates`
  - `spec.restartPolicy`
  - `spec.runtimeClassName`
  - `spec.schedulerName`
  - `spec.subdomain`
  - `spec.terminationGracePeriodSeconds`
  - `spec.tolerations`
  - `spec.topologySpreadConstraints`
  - `spec.volumes`

  These are the allowed Spark main container level fields:
  - `env`
  - `envFrom`
  - `name`
  - `lifecycle`
  - `livenessProbe`
  - `readinessProbe`
  - `resources`

- `startupProbe`
- `stdin`
- `stdinOnce`
- `terminationMessagePath`
- `terminationMessagePolicy`
- `tty`
- `volumeDevices`
- `volumeMounts`
- `workingDir`

When you use any disallowed fields in the pod template, Spark throws an exception and the job fails. The following example shows an error message in the Spark controller log due to disallowed fields.

```
Executor pod template validation failed.
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR on EKS predefines the following parameters in a pod template. The fields that you specify in a pod template must not overlap with these fields.

  These are the predefined volume names:

  - `emr-container-communicate`
  - `config-volume`
  - `emr-container-application-log-dir`
  - `emr-container-event-log-dir`
  - `temp-data-dir`
  - `mnt-dir`
  - `home-dir`
  - `emr-container-s3`

  These are the predefined volume mounts that only apply to the Spark main container:

  - Name: `emr-container-communicate`; MountPath: `/var/log/fluentd`
  - Name: `emr-container-application-log-dir`; MountPath: `/var/log/spark/user`
  - Name: `emr-container-event-log-dir`; MountPath: `/var/log/spark/apps`
  - Name: `mnt-dir`; MountPath: `/mnt`
  - Name: `temp-data-dir`; MountPath: `/tmp`
  - Name: `home-dir`; MountPath: `/home/hadoop`

  These are the predefined environment variables that only apply to the Spark main container:

  - `SPARK_CONTAINER_ID`
  - `K8S_SPARK_LOG_URL_STDERR`
  - `K8S_SPARK_LOG_URL_STDOUT`
  - `SIDECAR_SIGNAL_FILE`

  Note
  You can still use these predefined volumes and mount them into your additional sidecar containers. For example, you can use `emr-container-application-log-dir` and mount it to your own sidecar container defined in the pod template.

If the fields you specify conflict with any of the predefined fields in the pod template, Spark throws an exception and the job fails. The following example shows an error message in the Spark application log due to conflicts with the predefined fields.

```
Defined volume mount path on main container must not overlap with reserved mount paths:
 [<reserved-paths>]
```

Sidecar container considerations

Amazon EMR controls the lifecycle of the pods provisioned by Amazon EMR on EKS. The sidecar containers should follow the same lifecycle of the Spark main container. If you inject additional sidecar containers into your pods, we recommend that you integrate with the pod lifecycle management that Amazon EMR defines so that the sidecar container can stop itself when the Spark main container exits.

To reduce costs, we recommend that you implement a process that prevents driver pods with sidecar containers from continuing to run after your job completes. The Spark driver deletes executor pods when the executor is done. However, when a driver program completes, the additional sidecar containers continue to run. The pod is billed until Amazon EMR on EKS cleans up the driver pod, usually less than one minute after the driver Spark main container completes. To reduce costs, you can integrate your additional sidecar containers with the lifecycle management mechanism that Amazon EMR on EKS defines for both driver and executor pods, as described in the following section.

Spark main container in driver and executor pods sends `heartbeat` to a file `/var/log/fluentd/main-container-terminated` every two seconds. By adding the Amazon EMR predefined `emr-container-communicate` volume mount to your sidecar container, you can define a sub-process of your sidecar container to periodically track the last modified time for this file. The sub-process then stops itself if it discovers that the Spark main container stops the `heartbeat` for a longer duration.

The following example demonstrates a sub-process that tracks the heartbeat file and stops itself. Replace *`your_volume_mount`* with the path where you mount the predefined volume. The script is bundled inside the image used by sidecar container. In a pod template file, you can specify a sidecar container with the following commands `sub_process_script.sh` and `main_command`.

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
INITIAL_HEARTBEAT_TIMEOUT_THREASHOLD=60
HEARBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
  # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THREASHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THREASHOLD
 seconds; aborting"
        terminate_main_process()
        exit 1
    fi
    sleep $SLEEP_DURATION;
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$file_to_watch" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $file_to_watch)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARBEAT_TIMEOUT_THRESHOLD" ]; then
```

```
        echo "Last heartbeat to file $FILE_TO_WATCH was more than
 $HEARBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
        terminate_main_process()
        exit 0
    fi
    sleep $SLEEP_DURATION;
done;
```

Using Spark event log rotation

With Amazon EMR 6.3.0 and later, you can turn on the Spark event log rotation feature for Amazon EMR on EKS. Instead of generating a single event log file, this feature rotates the file based on your configured time interval and removes the oldest event log files.

Rotating Spark event logs can help you avoid potential issues with a large Spark event log file generated for long running or streaming jobs. For example, you start a long running Spark job with an event log enabled with the `persistentAppUI` parameter. The Spark driver generates an event log file. If the job runs for hours or days and there is a limited disk space on the Kubernetes node, the event log file can consume all available disk space. Turning on the Spark event log rotation feature solves the problem by splitting the log file into multiple files and removing the oldest files.

> **Note**
> This feature only works with Amazon EMR on EKS and is not supported by Amazon EMR running on Amazon EC2.

To turn on the Spark event log rotation feature, configure the following Spark parameters:

- `spark.eventLog.rotation.enabled` - turns on log rotation. It is disabled by default in the Spark configuration file. Set it to true to turn on this feature.
- `spark.eventLog.rotation.interval` - specifies time interval for the log rotation. The minimum value is 60 seconds. The default value is 300 seconds.
- `spark.eventLog.rotation.minFileSize` - specifies a minimum file size to rotate the log file. The minimum and default value is 1 MB.
- `spark.eventLog.rotation.maxFilesToRetain` - specifies how many rotated log files to keep during cleanup. The valid range is 1 to 10. The default value is 2.

You can specify these parameters in the `sparkSubmitParameters` section of the `StartJobRun` API, as the following example shows.

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --conf
 spark.eventLog.rotation.minFileSize=1m --conf spark.eventLog.rotation.maxFilesToRetain=2"
```

Monitoring jobs

Topics

Monitor jobs with Amazon CloudWatch Events

Amazon EMR on EKS emits events when the state of a job run changes. Each event provides information, such as the date and time when the event occurred, along with further details about the event, such as the virtual cluster ID and the ID of the job run that was affected.

You can use events to track the activity and health of a jobs that you run on a virtual cluster. You can also use Amazon CloudWatch Events to define an action to take when a job run generates an event that matches a pattern that you specify. Events are useful for monitoring a specific occurrence during the lifecycle of a job run. For example, you can monitor when a job run changes state from `submitted` to `running`. For more information about CloudWatch Events, see the *Amazon CloudWatch Events User Guide*.

The following table lists Amazon EMR on EKS events along with the state or state change that the event indicates, the severity of the event, and event messages. Each event is represented as a JSON object that is sent automatically to an event stream. The JSON object includes further details about the event. The JSON object is particularly important when you set up rules for event processing using CloudWatch Events because rules seek to match patterns in the JSON object. For more information, see Events and Event Patterns and Amazon EMR on EKS Events in the *Amazon CloudWatch Events User Guide*.

Job run state change events

State	Severity	Message
SUBMITTED	INFO	Job Run *JobRunId* (*JobRunName*) was successfully submitted to virtual cluster *VirtualClusterId* at *Time* UTC.
RUNNING	INFO	Job Run *JobRunId* (*JobRunName*) in virtual cluster *VirtualClusterId* started running at *Time*.
COMPLETED	INFO	Job Run *jobRunId* (*JobRunName*) in virtual cluster *VirtualClusterId* completed at *Time*. The Job Run started running at *Time* and took *Num* minutes to complete.
CANCELLED	WARN	Cancellation request has succeeded for Job Run *JobRunId* (*JobRunName*) in virtual cluster *VirtualClusterId* at *Time* and the Job Run is now cancelled.
FAILED	ERROR	Job Run *JobRunId* (*JobRunName*) in virtual cluster *VirtualClusterId* failed at *Time*.

Automate Amazon EMR on EKS with CloudWatch Events

You can use Amazon CloudWatch Events to automate your AWS services to respond to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon Simple Notification Service (SNS) topic or an Amazon Simple Queue Service (SQS) queue

Some examples of using CloudWatch Events with Amazon EMR on EKS include the following:

- Activating a Lambda function when a job run succeeds
- Notifying an Amazon SNS topic when a job run fails

CloudWatch Events for "`detail-type:`" "`EMR Job Run State Change`" are generated by Amazon EMR on EKS for `SUBMITTED`, `RUNNING`, `CANCELLED`, `FAILED` and `COMPLETED` state changes.

Example: Set up a rule that invokes Lambda

Use the following steps to set up a CloudWatch Events rule that invokes Lambda when there is an "EMR Job Run State Change" event.

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

Add the Lambda function that you own as a new target and give CloudWatch Events permission to invoke the Lambda function as follows. Replace *123456789012* with your account ID.

```
aws events put-targets \
--rule cwe-test \
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com
```

Note
You cannot write a program that depends on the order or existence of notification events, as they might be out of sequence or missing. Events are emitted on a best effort basis.

Managing virtual clusters

A virtual cluster is a Kubernetes namespace that Amazon EMR is registered with. You can create, describe, list, and delete virtual clusters. They do not consume any additional resource in your system. A single virtual cluster maps to a single Kubernetes namespace. Given this relationship, you can model virtual clusters the same way you model Kubernetes namespaces to meet your requirements. See possible use cases in the Kubernetes Concepts Overview documentation.

To register Amazon EMR with a Kubernetes namespace on an Amazon EKS cluster, you need the name of the EKS cluster and the namespace that has been set up for running your workload. These registered clusters in Amazon EMR are called virtual clusters because they do not manage physical compute or storage but point to a Kubernetes namespace where your workload is scheduled.

Note
Before creating a virtual cluster, you must first complete the steps 1-8 in Setting up (p. 5).

Topics

Create a virtual cluster

Run the following command to create a virtual cluster by registering Amazon EMR with a namespace on an EKS cluster. Replace *virtual_cluster_name* with a name that you provide for your virtual cluster. Replace *eks_cluster_name* with the name of the EKS cluster. Replace the *namespace_name* with the namespace that you want to register Amazon EMR with.

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
    "id": "eks_cluster_name",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "namespace_name"
        }
    }
}'
```

Alternatively, you can create a JSON file that includes the required parameters for the virtual cluster, as the following example demonstrates.

```
{
    "name": "virtual_cluster_name",
    "containerProvider": {
        "type": "EKS",
        "id": "eks_cluster_name",
        "info": {
            "eksInfo": {
```

```
                "namespace": "namespace_name"
            }
        }
    }
}
```

Then run the following `create-virtual-cluster` command with the path to the JSON file.

```
aws emr-containers create-virtual-cluster \
--cli-input-json file://./create-virtual-cluster-request.json
```

Note
To validate the successful creation of a virtual cluster, view the status of virtual clusters by running the `list-virtual-clusters` command or by going to the **Virtual clusters** page in the Amazon EMR console.

List virtual clusters

Run the following command to view the status of virtual clusters.

```
aws emr-containers list-virtual-clusters
```

Describe a virtual cluster

Run the following command to get more details about a virtual cluster, such as namespace, status, and date registered. Replace *123456* with your virtual cluster ID.

```
aws emr-containers describe-virtual-cluster --id 123456
```

Delete a virtual cluster

Run the following command to delete a virtual cluster. Replace *123456* with your virtual cluster ID.

```
aws emr-containers delete-virtual-cluster --id 123456
```

Virtual cluster states

The following table describes the four possible states of a virtual cluster.

State	Description
RUNNING	Virtual cluster is in RUNNING state.
TERMINATING	The requested termination of the virtual cluster is in progress.

State	Description
TERMINATED	The requested termination is complete.
ARRESTED	The requested termination failed because of insufficient permissions.

Security in Amazon EMR on EKS

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 Amazon EMR, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon EMR on EKS. The following topics show you how to configure Amazon EMR on EKS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EMR on EKS resources.

Topics

Data protection

The AWS shared responsibility model applies to data protection in Amazon EMR on EKS. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ . For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- Use Amazon EMR on EKS encryption options to encrypt data at rest and in transit.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon EMR on EKS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon EMR on EKS or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Encryption at rest

Data encryption helps prevent unauthorized users from reading data on a cluster and associated data storage systems. This includes data saved to persistent media, known as data at rest, and data that may be intercepted as it travels the network, known as data in transit.

Data encryption requires keys and certificates. You can choose from several options, including keys managed by AWS Key Management Service, keys managed by Amazon S3, and keys and certificates from custom providers that you supply. When using AWS KMS as your key provider, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

Before you specify encryption options, decide on the key and certificate management systems you want to use. Then create the keys and certificates for the custom providers that you specify as part of encryption settings.

Encryption at rest for EMRFS data in Amazon S3

Amazon S3 encryption works with EMR File System (EMRFS) objects read from and written to Amazon S3. You specify Amazon S3 server-side encryption (SSE) or client-side encryption (CSE) as the **Default encryption mode** when you enable encryption at rest. Optionally, you can specify different encryption methods for individual buckets using **Per bucket encryption overrides**. Regardless of whether Amazon S3 encryption is enabled, Transport Layer Security (TLS) encrypts the EMRFS objects in transit between EMR cluster nodes and Amazon S3. For in-depth information about Amazon S3 encryption, see Protecting Data Using Encryption in the Amazon Simple Storage Service Developer Guide.

> **Note**
> When you use AWS KMS, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

Amazon S3 server-side encryption

When you set up Amazon S3 server-side encryption, Amazon S3 encrypts data at the object level as it writes the data to disk and decrypts the data when it is accessed. For more information about SSE, see Protecting Data Using Server-Side Encryption in the Amazon Simple Storage Service Developer Guide.

You can choose between two different key management systems when you specify SSE in Amazon EMR on EKS:

- **SSE-S3** - Amazon S3 manages keys for you.
- **SSE-KMS** - You use an AWS KMS customer master key (CMK) set up with policies suitable for Amazon EMR on EKS.

SSE with customer-provided keys (SSE-C) is not available for use with Amazon EMR on EKS.

Amazon S3 client-side encryption

With Amazon S3 client-side encryption, the Amazon S3 encryption and decryption takes place in the EMRFS client on your cluster. Objects are encrypted before being uploaded to Amazon S3 and decrypted after they are downloaded. The provider you specify supplies the encryption key that the client uses. The client can use keys provided by AWS KMS (CSE-KMS) or a custom Java class that provides the client-side master key (CSE-C). The encryption specifics are slightly different between CSE-KMS and CSE-C, depending on the specified provider and the metadata of the object being decrypted or encrypted. For more information about these differences, see Protecting Data Using Client-Side Encryption in the Amazon Simple Storage Service Developer Guide.

> **Note**
> Amazon S3 CSE only ensures that EMRFS data exchanged with Amazon S3 is encrypted; not all data on cluster instance volumes is encrypted. Furthermore, because Hue does not use EMRFS, objects that the Hue S3 File Browser writes to Amazon S3 are not encrypted.

Local disk encryption

Apache Spark supports encrypting temporary data written to local disks. This covers shuffle files, shuffle spills, and data blocks stored on disk for both caching and broadcast variables. It does not cover encrypting output data generated by applications with APIs such as `saveAsHadoopFile` or `saveAsTable`. It also may not cover temporary files created explicitly by the user. For more information, see Local Storage Encryption in the Spark documentation. Spark does not support encrypted data on local disk, such as intermediate data written to a local disk by an executor process when the data does not fit in memory. Data that is persisted to disk is scoped to the job runtime, and the key that is used to encrypt the data is generated dynamically by Spark for every job run. Once the Spark job terminates, no other process can decrypt the data.

For driver and executor pod, you encrypt data at rest that is persisted to the mounted volume. There are three different AWS native storage options you can use with Kubernetes: EBS, EFS, and FSx for Lustre. All three offer encryption at rest using a service managed key or a customer master key (CMK). For more information see the EKS Best Practices Guide. With this approach, all data persisted to the mounted volume is encrypted.

Key management

You can configure KMS to automatically rotate your CMKs. This rotates your keys once a year while saving old keys indefinitely so that your data can still be decrypted. For additional information, see Rotating customer master keys.

Encryption in transit

Several encryption mechanisms are enabled with in-transit encryption. These are open-source features, are application-specific, and may vary by Amazon EMR on EKS release. The following application-specific encryption features can be enabled with Amazon EMR on EKS:

- Spark
  - Internal RPC communication between Spark components, such as the block transfer service and the external shuffle service, is encrypted using the AES-256 cipher in Amazon EMR versions 5.9.0 and

later. In earlier releases, internal RPC communication is encrypted using SASL with DIGEST-MD5 as the cipher.

- HTTP protocol communication with user interfaces such as Spark History Server and HTTPS-enabled file servers is encrypted using Spark's SSL configuration. For more information, see SSL Configuration in Spark documentation.

For more information, see Spark security settings.

- You should allow only encrypted connections over HTTPS (TLS) using the aws:SecureTransport condition on Amazon S3 bucket IAM policies.
- Query results that stream to JDBC or ODBC clients are encrypted using TLS.

Identity and Access Management

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon EMR on EKS resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon EMR on EKS.

Service user – If you use the Amazon EMR on EKS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon EMR on EKS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon EMR on EKS, see Troubleshooting Amazon EMR on EKS identity and access (p. 70).

Service administrator – If you're in charge of Amazon EMR on EKS resources at your company, you probably have full access to Amazon EMR on EKS. It's your job to determine which Amazon EMR on EKS features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon EMR on EKS, see How Amazon EMR on EKS works with IAM (p. 59).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon EMR on EKS. To view example Amazon EMR on EKS identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon EMR on EKS (p. 66).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API

operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon EMR on EKS in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon EMR on EKS works with IAM

Before you use IAM to manage access to Amazon EMR on EKS, learn what IAM features are available to use with Amazon EMR on EKS.

IAM features you can use with Amazon EMR on EKS

IAM feature	Amazon EMR on EKS support
Identity-based policies (p. 59)	Yes
Resource-based policies (p. 60)	No
Policy actions (p. 60)	Yes
Policy resources (p. 61)	Yes
Policy condition keys (p. 61)	Yes
ACLs (p. 62)	No
ABAC (tags in policies) (p. 62)	Yes
Temporary credentials (p. 62)	Yes
Principal permissions (p. 63)	Yes
Service roles (p. 63)	No
Service-linked roles (p. 63)	Yes

To get a high-level view of how Amazon EMR on EKS and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon EMR on EKS

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon EMR on EKS

To view examples of Amazon EMR on EKS identity-based policies, see Identity-based policy examples for Amazon EMR on EKS (p. 66).

Resource-based policies within Amazon EMR on EKS

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Amazon EMR on EKS

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon EMR on EKS actions, see Actions, resources, and condition keys for Amazon EMR on EKS in the *Service Authorization Reference*.

Policy actions in Amazon EMR on EKS use the following prefix before the action:

```
emr-containers
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "emr-containers:action1",
      "emr-containers:action2"
         ]
```

To view examples of Amazon EMR on EKS identity-based policies, see Identity-based policy examples for Amazon EMR on EKS (p. 66).

Policy resources for Amazon EMR on EKS

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. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon EMR on EKS resource types and their ARNs, see Resources defined by Amazon EMR on EKS in the *Service Authorization Reference*. To learn which actions you can specify the ARN of each resource, see Actions, resources, and condition keys for Amazon EMR on EKS.

To view examples of Amazon EMR on EKS identity-based policies, see Identity-based policy examples for Amazon EMR on EKS (p. 66).

Policy condition keys for Amazon EMR on EKS

Supports policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon EMR on EKS condition keys and to learn which actions and resources you can use a condition key, see Actions, resources, and condition keys for Amazon EMR on EKS in the *Service Authorization Reference*.

To view examples of Amazon EMR on EKS identity-based policies, see Identity-based policy examples for Amazon EMR on EKS (p. 66).

Access control lists (ACLs) in Amazon EMR on EKS

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon EMR on EKS

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using Temporary credentials with Amazon EMR on EKS

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon EMR on EKS

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon EMR on EKS in the *Service Authorization Reference*.

Service roles for Amazon EMR on EKS

Supports service roles	No

Service-linked roles for Amazon EMR on EKS

Supports service-linked roles	Yes

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.

Using service-linked roles for Amazon EMR on EKS

Amazon EMR on EKS uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EMR on EKS. Service-linked roles are predefined by Amazon EMR on EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EMR on EKS easier because you don't have to manually add the necessary permissions. Amazon EMR on EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EMR on EKS 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 Amazon EMR on EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EMR on EKS

Amazon EMR on EKS uses the service-linked role named **AWSServiceRoleForAmazonEMRContainers**.

The `AWSServiceRoleForAmazonEMRContainers` service-linked role trusts the following services to assume the role:

- `emr-containers.amazonaws.com`

The role permissions policy `AmazonEMRContainersServiceRolePolicy` allows Amazon EMR on EKS to complete a set of actions on the specified resources, as the following policy statement demonstrates.

Note
Managed policy contents change, so the policy shown here may be out-of-date. View the most up-to-date policy AmazonEMRContainersServiceRolePolicy in the AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers"
            ],
            "Resource": "*"
        }
    ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EMR on EKS

You don't need to manually create a service-linked role. When you create a virtual cluster, Amazon EMR on EKS 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 virtual cluster, Amazon EMR on EKS creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **Amazon EMR on EKS** use case. In the AWS CLI or the AWS API, create a service-linked role with the `emr-containers.amazonaws.com` service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for Amazon EMR on EKS

Amazon EMR on EKS does not allow you to edit the `AWSServiceRoleForAmazonEMRContainers` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EMR on EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored

or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
> If the Amazon EMR on EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon EMR on EKS resources used by the `AWSServiceRoleForAmazonEMRContainers`

1. Open the Amazon EMR console.
2. Choose a virtual cluster.
3. On the `Virtual Cluster` page choose **Delete**.
4. Repeat this procedure for any other virtual clusters in your account.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEMRContainers` service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Amazon EMR on EKS service-linked roles

Amazon EMR on EKS supports using service-linked roles in all of the Regions where the service is available. For more information, see Amazon EMR on EKS service quotas (p. 86).

Using job execution roles with Amazon EMR on EKS

To use the `StartJobRun` command to submit a job run on an EKS cluster, you must first onboard a job execution role to be used with a virtual cluster. For more information, see Create a job execution role (p. 15) in Setting up (p. 5) steps.

The following permissions must be included in the trust policy for the job execution role.

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-sa-*-
*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

The trust policy in the preceding example grants permissions only to an EMR managed Kubernetes service account with a name that matches the `emr-containers-sa-*-*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME` pattern. Service accounts with this pattern will be automatically created at job submission, scoped to the namespace where the job is submitted. This trust

policy allows these service accounts to assume the execution role and get the temporary credentials of the execution role. Any other service account from a different EKS cluster or from a different namespace within the same EKS cluster can not assume the execution role.

You can run the following command to update the trust policy automatically in the format given above.

```
aws emr-containers update-role-trust-policy \
       --cluster-name cluster \
       --namespace namespace \
       --role-name iam_role_name_for_job_execution
```

Controlling access to the execution role

An EKS cluster administrator can create a multi-tenant Amazon EMR on EKS virtual cluster to which an IAM administrator can add multiple execution roles. These execution roles can be used to submit jobs by untrusted tenants running arbitrary code. Therefore, you may want to restrict those tenants so that they cannot run code that gains the permissions assigned to one or more of these execution roles. You can use condition keys to restrict access to execution roles. The IAM administrator can restrict the IAM policy attached to an IAM identity by using an optional string condition key named `emr-containers:ExecutionRoleArn`. This condition accepts a list of execution role ARNs that have permissions to use a virtual cluster, as the following permissions policy demonstrates.

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": "arn:aws:emr-containers:REGION:AWS_ACCOUNT_ID:/
virtualclusters/VIRTUAL_CLUSTER_ID",
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "execution_role_arn_1",
            "execution_role_arn_2",
            ...
          ]
        }
      }
    }
  ]
}
```

If you want to allow all execution roles that begin with a particular prefix, such as MyRole, the condition operator StringEquals can be replaced with a StringLike operator, and execution_role_arn value in the condition can be replaced with a wildcard * character. For example, `arn:aws:iam::AWS_ACCOUNT_ID:role/MyRole*`. All other string condition keys are also supported.

> **Note**
> Amazon EMR on EKS currently does not support tag-based access control (TBAC) for execution roles. You cannot grant permissions to the execution roles based on tags.

Identity-based policy examples for Amazon EMR on EKS

By default, IAM users and roles don't have permission to create or modify Amazon EMR on EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM

administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon EMR on EKS 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 using AWS managed policies** – To start using Amazon EMR on EKS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Amazon EMR on EKS console

To access the Amazon EMR on EKS console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon EMR on EKS 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 (IAM 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 you're trying to perform.

To ensure that users and roles can still use the Amazon EMR on EKS console, also attach the Amazon EMR on EKS `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": "*"
        }
    ]
}
```

Policies for tag-based access control

You can use conditions in your identity-based policy to control access to virtual clusters and job runs based on tags. For more information about tagging, see Tagging your Amazon EMR on EKS resources (p. 83).

The following examples demonstrate different scenarios and ways to use condition operators with Amazon EMR on EKS condition keys. These IAM policy statements are intended for demonstration purposes only and should not be used in production environments. There are multiple ways to combine policy statements to grant and deny permissions according to your requirements. For more information about planning and testing IAM policies, see the IAM User Guide.

Important
Explicitly denying permission for tagging actions is an important consideration. This prevents users from tagging a resource and thereby granting themselves permissions that you did not intend to grant. If tagging actions for a resource are not denied, a user can modify tags and circumvent the intention of the tag-based policies. For an example of a policy that denies tagging actions, see Deny access to add and remove tags (p. 70).

The examples below demonstrate identity-based permissions policies that are used to control the actions that are allowed with Amazon EMR on EKS virtual clusters.

Allow actions only on resources with specific tag values

In the following policy example, the StringEquals condition operator tries to match dev with the value for the tag department. If the tag department hasn't been added to the virtual cluster, or doesn't contain

the value dev, the policy doesn't apply, and the actions aren't allowed by this policy. If no other policy statements allow the actions, the user can only work with virtual clusters that have this tag with this value.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

You can also specify multiple tag values using a condition operator. For example, to allow actions on virtual clusters where the `department` tag contains the value `dev` or `test`, you could replace the condition block in the earlier example with the following.

```
"Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": ["dev", "test"]
        }
      }
```

Require tagging when a resource is created

In the example below, the tag needs to be applied when creating the virtual cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      }
    }
  ]
}
```

The following policy statement allows a user to create a virtual cluster only if the cluster has a `department` tag, which can contain any value.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/department": "false"
        }
      }
    }
  ]
}
```

Deny access to add and remove tags

The effect of this policy is to deny a user the permission to add or remove any tags on virtual clusters that are tagged with a `department` tag that contains the `dev` value.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",
        "emr-containers:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

Troubleshooting Amazon EMR on EKS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon EMR on EKS and IAM.

Topics

I am not authorized to perform an action in Amazon EMR on EKS

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` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `emr-containers:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-
containers: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 `emr-containers:`*`GetWidget`* action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amazon EMR on EKS

To allow others to access Amazon EMR on EKS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon EMR on EKS.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon EMR on EKS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon EMR on EKS supports these features, see How Amazon EMR on EKS works with IAM (p. 59).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Security best practices

Amazon EMR on EKS provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Apply principle of least privilege

Amazon EMR on EKS provides a granular access policy for applications using IAM roles, such as execution roles. These execution roles are mapped to Kubernetes service accounts through the IAM role's trust policy. Amazon EMR on EKS creates pods within a registered Amazon EKS namespace that execute user-provided application code. The job pods running the application code assume the execution role when connecting to other AWS services. We recommend that execution roles be granted only the minimum set of privileges required by the job, such as covering your application and access to log destination. We also recommend auditing the jobs for permissions on a regular basis and upon any change to application code.

Access control list for endpoints

Managed endpoints can be created only for those EKS clusters that have been configured to use at least one private subnet in your VPC. This configuration restricts access to the load balancers created by managed endpoints so that they can only be accessed from your VPC. To further enhance security, we recommend that you configure security groups with these load balancers so that they can restrict incoming traffic to a selected set of IP addresses.

Get the latest security updates for custom images

To use custom images with Amazon EMR on EKS, you can install any binaries and libraries on the image. You are responsible for the security patching of the binaries you add to the image. Amazon EMR on EKS images are regularly patched with latest security patches. To get the latest image, you must rebuild the custom images whenever there is a new base image version of the Amazon EMR release. For more information, see Amazon EMR on EKS release versions (p. 88) and How to select a base image URI (p. 28).

Limit pod credential access

Kubernetes supports several methods of assigning credentials to a pod. Provisioning multiple credentials providers can increase the complexity of your security model. Amazon EMR on EKS has adopted the use of IAM roles for services accounts (IRSA) as a standard credential provider within a registered EKS namespace. Other methods are not supported, including kube2iam, kiam and using an EC2 instance profile of the instance running on the cluster.

Isolate untrusted application code

Amazon EMR on EKS does not inspect the integrity of the application code submitted by users of the system. If you are running a multi-tenanted virtual cluster that is configured using multiple execution roles that can be used to submit jobs by untrusted tenants running arbitrary code, there is a risk of a malicious application escalating its privileges. In this situation, consider isolating execution roles with similar privileges into a different virtual cluster.

Role-based access control (RBAC) permissions

Administrators should strictly control Role-based access control (RBAC) permissions for Amazon EMR on EKS managed namespaces. At a minimum, the following permissions should not be granted to job submitters in Amazon EMR on EKS managed namespaces.

- Kubernetes RBAC permissions to modify configmap - because Amazon EMR on EKS uses Kubernetes configmaps to generate managed pod templates that have the managed service account name. This attribute should not be mutated.
- Kubernetes RBAC permissions to exec into Amazon EMR on EKS pods - to avoid giving access to managed pod templates that have the managed SA name. This attribute should not be mutated. This permission can also give access to the JWT token mounted into the pod which can then be used to retrieve the execution role credentials.
- Kubernetes RBAC permissions to create pods - to prevent users from creating pods using a Kubernetes ServiceAccount which may be mapped to an IAM role with more AWS privileges than the user.
- Kubernetes RBAC permissions to deploy mutating webhook - to prevent users from using the mutating webhook to mutate Kubernetes ServiceAccount name for pods created by Amazon EMR on EKS.
- Kubernetes RBAC permissions to read Kubernetes secrets - to prevent users from reading confidential data stored in these secrets.

Restrict access to nodegroup IAM role or instance profile credentials

- We recommend that you assign minimum AWS permissions to nodegroup's IAM role(s). This helps to avoid privilege escalation by code that may run using instance profile credentials of EKS worker nodes.
- To completely block access to instance profile credentials to all pods that runs in Amazon EMR on EKS managed namespaces, we recommend that you run `iptables` commands on EKS nodes. For more information, see Restricting access to Amazon EC2 instance profile credentials. However, it is important to properly scope your service account IAM roles so that your pods have all of the necessary permissions. For example, the node IAM role is assigned permissions to pull container images from Amazon ECR. If a pod isn't assigned those permissions, the pod can't pull container images from Amazon ECR. The VPC CNI plugin also needs to be updated. For more information, see Walkthrough: Updating the VPC CNI plugin to use IAM roles for service accounts.

Logging and monitoring

To detect incidents, receive alerts when incidents occur, and respond to them, use these options with Amazon EMR on EKS:

- Monitor Amazon EMR on EKS with AWS CloudTrail - AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon EMR on EKS. It captures calls from the Amazon EMR console and code calls to the Amazon EMR on EKS API operations as events. This allows you to determine the request that was made to Amazon EMR on EKS, the IP address from which the request

was made, who made the request, when it was made, and additional details. For more information, see Logging Amazon EMR on EKS API calls using AWS CloudTrail (p. 74).

- Use CloudWatch Events with Amazon EMR on EKS - CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources. CloudWatch Events becomes aware of operational changes as they occur, responds to them, and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information. To use CloudWatch Events with Amazon EMR on EKS, create a rule that triggers on an Amazon EMR on EKS API call via CloudTrail. For more information, see Monitor jobs with Amazon CloudWatch Events (p. 47).

Logging Amazon EMR on EKS API calls using AWS CloudTrail

Amazon EMR on EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EMR on EKS. CloudTrail captures all API calls for Amazon EMR on EKS as events. The calls captured include calls from the Amazon EMR on EKS console and code calls to the Amazon EMR on EKS API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon EMR on EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon EMR on EKS, 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.

Amazon EMR on EKS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon EMR on EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Amazon EMR on EKS, 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 Amazon EMR on EKS actions are logged by CloudTrail and are documented in Amazon EMR on EKS API documentation. For example, calls to the `CreateVirtualCluster`, `StartJobRun` and `ListJobRuns` actions 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 AWS Identity and Access Management (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 user Identity element.

Understanding Amazon EMR on EKS 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 `ListJobRuns` action.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
      }
    }
  },
  "eventTime": "2020-11-04T21:52:58Z",
  "eventSource": "emr-containers.amazonaws.com",
  "eventName": "ListJobRuns",
  "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": {
    "virtualClusterId": "1K48XXXXXXHCB"
  },
  "responseElements": null,
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "012345678910"
}
```


Compliance validation for Amazon EMR on EKS

Third-party auditors assess the security and compliance of Amazon EMR on EKS as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

To learn whether Amazon EMR or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  Note
  Not all services are compliant with HIPAA.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon EMR on EKS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon EMR on EKS offers integration with Amazon S3 through EMRFS to help support your data resiliency and backup needs.

Infrastructure security in Amazon EMR on EKS

As a managed service, Amazon EMR on EKS is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon EMR on EKS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and vulnerability analysis

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- Compliance validation for Amazon EMR on EKS (p. 75)
- Shared Responsibility Model
- Amazon Web Services: Overview of Security Processes (whitepaper)

Connect to Amazon EMR on EKS Using an Interface VPC Endpoint

You can connect directly to Amazon EMR on EKS using Interface VPC endpoints (AWS PrivateLink) in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use an interface VPC endpoint, communication between your VPC and Amazon EMR on EKS is conducted entirely within the AWS network. Each VPC endpoint is represented by one or more Elastic network interfaces (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to Amazon EMR on EKS without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Amazon EMR on EKS API.

You can create an interface VPC endpoint to connect to Amazon EMR on EKS using the AWS Management Console or AWS Command Line Interface (AWS CLI) commands. For more information, see Creating an Interface Endpoint.

After you create an interface VPC endpoint, if you enable private DNS hostnames for the endpoint, the default Amazon EMR on EKS endpoint resolves to your VPC endpoint. The default service name endpoint for Amazon EMR on EKS is in the following format.

```
emr-containers.Region.amazonaws.com
```

If you do not enable private DNS hostnames, Amazon VPC provides a DNS endpoint name that you can use in the following format.

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

For more information, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide. Amazon EMR on EKS supports making calls to all of its API Actions inside your VPC.

You can attach VPC endpoint policies to a VPC endpoint to control access for IAM principals. You can also associate security groups with a VPC endpoint to control inbound and outbound access based on the origin and destination of network traffic, such as a range of IP addresses. For more information, see Controlling Access to Services with VPC Endpoints.

Create a VPC Endpoint Policy for Amazon EMR on EKS

You can create a policy for Amazon VPC endpoints for Amazon EMR on EKS to specify the following:

- The principal that can or cannot perform actions
- The actions that can be performed
- The resources on which actions can be performed

For more information, see Controlling Access to Services with VPC Endpoints in the Amazon VPC User Guide.

Example VPC Endpoint Policy to Deny All Access From a Specified AWS Account

The following VPC endpoint policy denies AWS account *123456789012* all access to resources using the endpoint.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example VPC Endpoint Policy to Allow VPC Access Only to a Specified IAM Principal (User)

The following VPC endpoint policy allows full access only to the IAM user *lijuan* in AWS account *123456789012*. All other IAM principals are denied access using the endpoint.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/lijuan"
                ]
            }
        }
    ]
}
```

Example VPC Endpoint Policy to Allow Read-Only Amazon EMR on EKS Operations

The following VPC endpoint policy allows only AWS account *123456789012* to perform the specified Amazon EMR on EKS actions.

The actions specified provide the equivalent of read-only access for Amazon EMR on EKS. All other actions on the VPC are denied for the specified account. All other accounts are denied any access. For a list of Amazon EMR on EKS actions, see Actions, Resources, and Condition Keys for Amazon EMR on EKS.

```
{
    "Statement": [
        {
            "Action": [
                "emr-containers:DescribeJobRun",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:ListJobRuns",
                "emr-containers:ListTagsForResource",
                "emr-containers:ListVirtualClusters"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example VPC Endpoint Policy Denying Access to a Specified Virtual Cluster

The following VPC endpoint policy allows full access for all accounts and principals, but denies any access for AWS account *123456789012* to actions performed on the virtual cluster with cluster ID *A1B2CD34EF5G*. Other Amazon EMR on EKS actions that don't support resource-level permissions for virtual clusters are still allowed. For a list of Amazon EMR on EKS actions and their corresponding resource type, see Actions, Resources, and Condition Keys for Amazon EMR on EKS.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/
virtualclusters/A1B2CD34EF5G",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Set up cross-account access for Amazon EMR on EKS

You can set up cross-account access for Amazon EMR on EKS. Cross-account access enables users from one AWS account to run Amazon EMR on EKS jobs and access the underlying data that belongs to another AWS account.

Prerequisites

To set up cross-account access for Amazon EMR on EKS, you'll complete tasks while signed in to the following AWS accounts:

- `AccountA` - An AWS account where you have created an Amazon EMR on EKS virtual cluster by registering Amazon EMR with a namespace on an EKS cluster.
- `AccountB` - An AWS account that contains an Amazon S3 bucket or a DynamoDB table that you want your Amazon EMR on EKS jobs to access.

You must have the following ready in your AWS accounts before setting up cross-account access:

- An Amazon EMR on EKS virtual cluster in `AccountA` where you want to run jobs.
- A job execution role in `AccountA` that has the required permissions to run jobs in the virtual cluster. For more information, see Create a job execution role (p. 15) and Using job execution roles with Amazon EMR on EKS (p. 65).

How to access a cross-account Amazon S3 bucket or DynamoDB table

To set up cross-account access for Amazon EMR on EKS, complete the following steps.

1. Create an Amazon S3 bucket, `cross-account-bucket`, in `AccountB`. For more information, see Creating a bucket. If you want to have cross-account access to DynamoDB, you can also create a DynamoDB table in `AccountB`. For more information, see Creating a DynamoDB table.
2. Create a `Cross-Account-Role-B` IAM role in `AccountB` that can access the `cross-account-bucket`.
    1. Sign in to the IAM console.
    2. Choose **Roles** and create a new role: `Cross-Account-Role-B`. For more information about how to create IAM roles, see Creating IAM roles in the IAM User Guide.
    3. Create an IAM policy that specifies the permissions for `Cross-Account-Role-B` to access the `cross-account-bucket` S3 bucket, as the following policy statement demonstrates. Then attach the IAM policy to `Cross-Account-Role-B`. For more information, see Creating a New Policy in the IAM User Guide.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::cross-account-bucket",
                "arn:aws:s3:::cross-account-bucket/*"
```

```
            ]
        }
    ]
}
```

If DynamoDB access is required, create an IAM policy that specifies permissions to access the cross-account DynamoDB table. Then attach the IAM policy to `Cross-Account-Role-B`. For more information, see Create a DynamoDB table in the IAM user guide.

Following is a policy to access a DynamoDB table, `CrossAccountTable`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:MyRegion:AccountB:table/CrossAccountTable"
        }
    ]
}
```

3. Edit the trust relationship for the `Cross-Account-Role-B` role.

    1. To configure the trust relationship for the role, choose the **Trust Relationships** tab in the IAM console for the role created in Step 2: `Cross-Account-Role-B`.
    2. Select **Edit Trust Relationship**.
    3. Add the following policy document, which allows `Job-Execution-Role-A` in `AccountA` to assume this `Cross-Account-Role-B` role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA:role/Job-Execution-Role-A"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. Grant `Job-Execution-Role-A` in `AccountA` with - STS Assume role permission to assume `Cross-Account-Role-B`.

    1. In the IAM console for AWS account `AccountA`, select `Job-Execution-Role-A`.
    2. Add the following policy statement to the `Job-Execution-Role-A` to allow the `AssumeRole` action on the `Cross-Account-Role-B` role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
        }
    ]
}
```

5. For Amazon S3 access, set the following `spark-submit` parameters (`spark conf`) while submitting the job to Amazon EMR on EKS.

   Note
   By default, EMRFS uses the job execution role to access the S3 bucket from the job. But when `customAWSCredentialsProvider` is set to `AssumeRoleAWSCredentialsProvider`, EMRFS uses the corresponding role that you specify with `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` instead of the `Job-Execution-Role-A` for Amazon S3 access.

   - `--conf spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCreder`
   - `--conf spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:ro Cross-Account-Role-B \`
   - `--conf spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/ Cross-Account-Role-B \`

   Note
   You must set `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` for both executor and driver `env` in the job spark configuration.

   For DynamoDB cross-account access, you must set `--conf spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials`

6. Run the Amazon EMR on EKS job with cross-account access, as the following example demonstrates.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
 <main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
 --conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentialsProvider
 --conf
 spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
 spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/Cross-
Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification": "spark-
defaults", "properties": {"spark.driver.memory": "2G"}}], "monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName": "log_group_name",
 "logStreamNamePrefix": "log_stream_prefix"}, "persistentAppUI":"ENABLED",
 "s3MonitoringConfiguration": {"logUri": "s3://my_s3_log_location" }}}'
```

Tagging your Amazon EMR on EKS resources

To help you manage your Amazon EMR on EKS resources, you can assign your own metadata to each resource using tags. This topic provides an overview of the tags function and shows you how to create tags.

Topics

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 attributes such as 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 Amazon EMR on EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

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 Amazon EMR on EKS and are interpreted strictly as a string of characters.

A tag value can be an empty string, but not null. A tag key cannot be an empty string. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the earlier value.

If you use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

For tag-based access control policy examples, see Policies for tag-based access control (p. 68).

Tag your resources

You can tag new or existing virtual clusters and job runs that are in active states. The active states for job runs include: `PENDING`, `SUBMITTED`, `RUNNING`, and `CANCEL_PENDING`. The active states for virtual clusters include: `RUNNING`, `TERMINATING` and `ARRESTED`. For more information, see Job run states (p. 35) and Virtual cluster states (p. 50).

When a virtual cluster is terminated, tags are cleaned and no longer accessible.

If you're using the Amazon EMR on EKS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the tags parameter on the relevant API action. You can apply tags to existing resources using the `TagResource` API action.

You can use some resource-creating actions to specify tags for a resource when the resource is created. In this case, if tags cannot be applied while the resource is being created, the resource fails to be created. This mechanism 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 creating a resource.

The following table describes the Amazon EMR on EKS resources that can be tagged.

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon EMR on EKS API, AWS CLI, and AWS SDK)	API for creation (tags can be added during creation)
Virtual cluster	Yes	No. Tags associated with a virtual cluster do not propagate to job runs submitted to that virtual cluster.	Yes	CreateVirtualCluster
Job runs	Yes	No	Yes	StartJobRun

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.
- A tag value can be an empty string, but not null. A tag key cannot be an empty string.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use.

Work with tags using the AWS CLI and the Amazon EMR on EKS API

Use the following AWS CLI commands or Amazon EMR on EKS API operations to add, update, list, and delete the tags for your resources.

Task	AWS CLI	API action
Add or overwrite one or more tags	tag-resource	TagResource
List tags for a resource	list-tags-for-resource	ListTagsForResource
Delete one or more tags	untag-resource	UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing virtual cluster

The following command tags an existing virtual cluster.

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing virtual cluster

The following command deletes a tag from an existing virtual cluster.

```
aws emr-containers 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 emr-containers list-tags-for-resource --resource-arn resource_ARN
```

Amazon EMR on EKS service quotas

The following are the service endpoints and service quotas for Amazon EMR on EKS. To connect programmatically to an AWS service, you use an endpoint. In addition to the standard AWS endpoints, some AWS services offer FIPS endpoints in selected Regions. For more information, see AWS service endpoints. Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account. For more information, see AWS service quotas.

Service endpoints

Region name	Code	Endpoint	Protocol
US East (N. Virginia)	us-east-1	emr-containers.us-east-1.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	emr-containers.us-east-2.amazonaws.com	HTTPS
US West (N. California)	us-west-1	emr-containers.us-west-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	emr-containers.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	emr-containers.ap-northeast-1.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	emr-containers.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	emr-containers.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	emr-containers.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	emr-containers.ap-southeast-2.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	emr-containers.ca-central-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	emr-containers.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	emr-containers.eu-west-1.amazonaws.com	HTTPS
Europe (London)	eu-west-2	emr-containers.eu-west-2.amazonaws.com	HTTPS

Service quotas

Amazon EMR on EKS throttles the following API requests for each AWS account on a per-Region basis. For more information about how throttling is applied, see API Request Throttling in the *Amazon EC2 API Reference*. You can request an increase to API throttling quotas for your AWS account. You can request a quota adjustment through the Service Quotas dashboard.

API action	Bucket maximum capacity	Bucket refill rate (per second)
CancelJobRun	25	1
CreateManagedEndpoint	25	1
CreateVirtualCluster	25	1
DeleteManagedEndpoint	25	1
DeleteVirtualCluster	25	1
DescribeJobRun	25	1
DescribeVirtualCluster	25	1
ListJobRun	25	1
ListManagedEndpoint	25	1
ListVirtualCluster	25	1
StartJobRun	25	1
At the AWS account level, the bucket maximum capacity and refill rate for the sum of all API actions listed in this table	50	7

Amazon EMR on EKS release versions

An Amazon EMR release is a set of open-source applications from the big data ecosystem. Each release comprises different big data applications, components, and features that you select to have Amazon EMR on EKS deploy and configure when you run your job.

Beginning with Amazon EMR versions 5.32.0 and 6.2.0, you can deploy Amazon EMR on EKS. This deployment option is not available with earlier Amazon EMR release versions. You must specify a supported release version when you submit your job.

Amazon EMR on EKS uses the following form of release label: `emr-x.x.x-latest` or `emr-x.x.x-yyyymmdd` with a specific release date. For example, `emr-6.2.0-latest` or `emr-6.2.0-20210129`. Using `-latest` ensures that your Amazon EMR version always includes the latest security updates.

> **Note**
> For a comparison between Amazon EMR on EKS and Amazon EMR running on EC2, see Amazon EMR FAQs.

Topics

Amazon EMR 6.3.0 releases

The following Amazon EMR 6.3.0 releases are available for Amazon EMR on EKS:

Release notes for Amazon EMR 6.3.0

- New features - Beginning with Amazon EMR 6.3.0 in the 6.x release series, Amazon EMR on EKS supports Spark's pod template feature. You can also turn on the Spark event log rotation feature for Amazon EMR on EKS. For more information, see Using pod templates (p. 40) and Using Spark event log rotation (p. 46).
- Supported applications - Spark 3.1.1-amzn-0, Jupyter Enterprise Gateway (endpoints, public preview).
- Supported components - `aws-hm-client` (Glue connector), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Supported configuration classifications:

Classifications	Descriptions
`core-site`	Change values in Hadoop's core-site.xml file.

Classifications	Descriptions
emrfs-site	Change EMRFS settings.
spark-metrics	Change values in Spark's metrics.properties file.
spark-defaults	Change values in Spark's spark-defaults.conf file.
spark-env	Change values in the Spark environment.
spark-hive-site	Change values in Spark's hive-site.xml file.
spark-log4j	Change values in Spark's log4j.properties file.

Configuration classifications allow you to customize applications. These often correspond to a configuration XML file for the application, such as spark-hive-site.xml. For more information, see Configuring Applications.

emr-6.3.0-latest

Release notes: `emr-6.3.0-latest` currently points to `emr-6.3.0-20210429`.

Regions: `emr-6.3.0-latest` is available in all Regions supported by Amazon EMR on EKS. For more information, see Amazon EMR on EKS service endpoints.

Container image tag: `emr-6.3.0:latest`

> **Note**
> Beginning with Amazon EMR release 6.3.0, you can use the `latest` tagged image to automatically get the latest base images in each build. If you are using an older Amazon EMR release, make sure to use the latest dated base image for your custom image.

emr-6.3.0-20210429

Release notes: `emr-6.3.0-20210429` was released on April 29, 2021. This is the initial release of EMR version 6.3.0.

Regions: `emr-6.3.0-20210429` is available in all Regions supported by Amazon EMR on EKS. For more information, see Amazon EMR on EKS service endpoints.

Container image tag: `emr-6.3.0:20210429`

Amazon EMR 6.2.0 releases

The following Amazon EMR 6.2.0 releases are available for Amazon EMR on EKS:

Release notes for Amazon EMR 6.2.0

- Supported applications - Spark 3.0.1-amzn-0, Jupyter Enterprise Gateway (endpoints, public preview).
- Supported components - `aws-hm-client` (Glue connector), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Supported configuration classifications:

Classifications	Descriptions
`core-site`	Change values in Hadoop's core-site.xml file.
`emrfs-site`	Change EMRFS settings.
`spark-metrics`	Change values in Spark's metrics.properties file.
`spark-defaults`	Change values in Spark's spark-defaults.conf file.
`spark-env`	Change values in the Spark environment.
`spark-hive-site`	Change values in Spark's hive-site.xml file.
`spark-log4j`	Change values in Spark's log4j.properties file.

Configuration classifications allow you to customize applications. These often correspond to a configuration XML file for the application, such as spark-hive-site.xml. For more information, see Configuring Applications.

emr-6.2.0-latest

Release notes: `emr-6.2.0-latest` currently points to `emr-6.2.0-20210129`.

Regions: `emr-6.2.0-latest` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-6.2.0-20210129`

emr-6.2.0-20210129

Release notes: `emr-6.2.0-20210129` was released on January 29, 2021. Compared to `emr-6.2.0-20201218`, this version contains issue fixes and security updates.

Regions: `emr-6.2.0-20210129` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-6.2.0-20210129`

emr-6.2.0-20201218

Release notes: `emr-6.2.0-20201218` was released on December 18, 2020. Compared to `emr-6.2.0-20201201`, this version contains issue fixes and security updates.

Regions: `emr-6.2.0-20201218` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-6.2.0-20201218`

emr-6.2.0-20201201

Release notes: `emr-6.2.0-20201201` was released on December 1, 2020. This is the initial release of EMR version 6.2.0.

Regions: `emr-6.2.0-20201201` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-6.2.0-20201201`

Amazon EMR 5.33.0 releases

The following Amazon EMR 5.33.0 releases are available for Amazon EMR on EKS:

- emr-5.33.0-latest (p. 91)
- emr-5.33.0-20210323 (p. 92)

Release notes for Amazon EMR 5.33.0

- New feature - Beginning with Amazon EMR 5.33.0 in the 5.x release series, Amazon EMR on EKS supports Spark's pod template feature. For more information, see Using pod templates (p. 40).
- Supported applications - Spark 2.4.7-amzn-1, Jupyter Enterprise Gateway (endpoints, public preview).
- Supported components - `aws-hm-client` (Glue connector), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Supported configuration classifications:

Classifications	Descriptions
`core-site`	Change values in Hadoop's core-site.xml file.
`emrfs-site`	Change EMRFS settings.
`spark-metrics`	Change values in Spark's metrics.properties file.
`spark-defaults`	Change values in Spark's spark-defaults.conf file.
`spark-env`	Change values in the Spark environment.
`spark-hive-site`	Change values in Spark's hive-site.xml file.
`spark-log4j`	Change values in Spark's log4j.properties file.

Configuration classifications allow you to customize applications. These often correspond to a configuration XML file for the application, such as spark-hive-site.xml. For more information, see Configuring Applications.

emr-5.33.0-latest

Release notes: `emr-5.33.0-latest` currently points to `emr-5.33.0-20210323`.

Regions: `emr-5.33.0-latest` is available in all Regions supported by Amazon EMR on EKS. For more information, see Amazon EMR on EKS service endpoints.

Container image tag: `emr-5.33.0-20210323`

emr-5.33.0-20210323

Release notes: `emr-5.33.0-20210323` was released on March 23, 2021. This is the initial release of EMR version 5.33.0.

Regions: `emr-5.33.0-20210323` is available in all Regions supported by Amazon EMR on EKS. For more information, see Amazon EMR on EKS service endpoints.

Container image tag: `emr-5.33.0-20210323`

Amazon EMR 5.32.0 releases

The following Amazon EMR 5.32.0 releases are available for Amazon EMR on EKS:

Release notes for Amazon EMR 5.32.0

- Supported applications - Spark 2.4.7-amzn-0, Jupyter Enterprise Gateway (endpoints, public preview).
- Supported components - `aws-hm-client` (Glue connector), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Supported configuration classifications:

Classifications	Descriptions
`core-site`	Change values in Hadoop's core-site.xml file.
`emrfs-site`	Change EMRFS settings.
`spark-metrics`	Change values in Spark's metrics.properties file.
`spark-defaults`	Change values in Spark's spark-defaults.conf file.
`spark-env`	Change values in the Spark environment.
`spark-hive-site`	Change values in Spark's hive-site.xml file.
`spark-log4j`	Change values in Spark's log4j.properties file.

Configuration classifications allow you to customize applications. These often correspond to a configuration XML file for the application, such as spark-hive-site.xml. For more information, see Configuring Applications.

emr-5.32.0-latest

Release notes: `emr-5.32.0-latest` currently points to `emr-5.32.0-20210129`.

Regions: `emr-5.32.0-latest` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-5.32.0-20210129`

emr-5.32.0-20210129

Release notes: `emr-5.32.0-20210129` was released on January 29, 2021. Compared to `emr-5.32.0-20201218`, this version contains issue fixes and security updates.

Regions: `emr-5.32.0-20210129` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-5.32.0-20210129`

emr-5.32.0-20201218

Release notes: `5.32.0-20201218` was released on December 18, 2020. Compared to `5.32.0-20201201`, this version contains issue fixes and security updates.

Regions: `emr-5.32.0-20201218` is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-5.32.0-20201218`

emr-5.32.0-20201201

Release notes: `5.32.0-20201201` was released on December 1, 2020. This is the initial release of EMR version 5.32.0.

Regions: `5.32.0-20201201`d is available in the following Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Container image tag: `emr-5.32.0-20201201`