

[Documentation](#) > [Clusters](#) > Cluster Configurations

# Cluster Configurations

January 31, 2020

This article explains the configuration options available when you create and edit Databricks clusters. It focuses on creating and editing clusters using the UI. For other methods, see [Clusters CLI](#) and [Clusters API](#).

For help deciding what combination of configuration options suits your needs best, see [Plan Capacity and Control Cost](#).

Create Cluster

New Cluster

Cancel

Create Cluster

2-8 Workers: 61.0-244.0 GB Memory, 8-32 Cores, 2-8 DBU  
 1 Driver: 30.5 GB Memory, 4 Cores, 1 DBU

Cluster Name

Test

Cluster Mode

Standard

Databricks Runtime Version

Runtime: 5.0 (Scala 2.11, Spark 2.4.0)

Learn more

Python Version

2

Autopilot Options

☒ Enable autoscaling

☐ Enable autoscaling local storage

☒ Terminate after 120 minutes of inactivity

Worker Type

i3.xlarge

30.5 GB Memory, 4 Cores, 1 DBU

Min Workers

2

Max Workers

8

Driver Type

Same as worker

30.5 GB Memory, 4 Cores, 1 DBU

## ▼ Advanced Options

On-demand/Spot Composition ?

2-8 Workers: 61 0-244 0 GB Memory 8-32 Cores 2-8 DRAM

1 Driver

2-8 Workers

Instances

Spark

Tags

SSH

Logging

Init Scripts

Availability Zone ?

us-west-2a

Spot Bid Price ?

100

% of on-demand instance price

EBS Volume Type ?

None

# Volumes ?

0

Size in GB ?

The local storage of the instance and the EBS

IAM Role ?

None

## In this article

- [Cluster mode](#)
- [Pool](#)
- [Databricks Runtime](#)
- [Python version](#)
- [Cluster node type](#)
- [Cluster size and autoscaling](#)
- [AWS configurations](#)
- [Spark configuration](#)
- [Environment variables](#)
- [Cluster tags](#)
- [SSH access to clusters](#)
- [Cluster log delivery](#)
- [Init scripts](#)

# Cluster mode

Databricks supports two cluster modes: standard and high concurrency. The default cluster mode is standard.

## Note

- High concurrency clusters are configured to *not terminate* automatically.

## Standard clusters

Standard clusters are recommended for a single user. Standard can run workloads developed in any language: Python, R, Scala, and SQL.

## High concurrency clusters

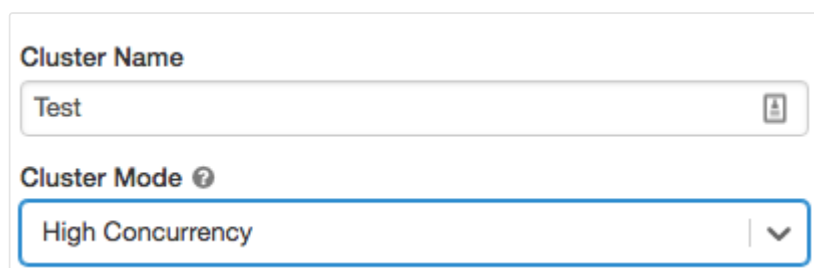
A high concurrency cluster is a managed cloud resource. The key benefits of high concurrency clusters are that they provide Apache Spark-native fine-grained sharing for maximum resource utilization and minimum query latencies. This sharing is accomplished with:

- **Preemption:** Proactively preempts Spark tasks from over-committed users to ensure all users get their fair share of cluster time and their jobs complete in a timely manner even when contending with dozens of other users. This uses Spark [Task Preemption for High Concurrency](#).
- **Fault isolation:** Creates an environment for each notebook, effectively isolating them from one another.

## Note

- High concurrency clusters work only for SQL, Python, and R. The performance, security, and fault isolation of high concurrency clusters is provided by running user code in separate processes, which is not possible in Scala.
- The [Table Access Control checkbox](#) is available only for high concurrency clusters.

To create a high concurrency cluster, in the Cluster Mode drop-down select **High Concurrency**.



The screenshot shows a web form for creating a Databricks cluster. It has two main sections. The first section is labeled 'Cluster Name' and contains a text input field with the value 'Test' and a small icon to the right. The second section is labeled 'Cluster Mode' with a help icon (question mark) to its right. Below this label is a dropdown menu that currently displays 'High Concurrency' and has a downward arrow on the right side.

For an example of how to create a high concurrency cluster using the Clusters API, see [High concurrency cluster example](#).

# Pool

■

To reduce cluster start time, you can attach a cluster to a predefined [pool](#) of idle instances. When attached to a pool, a cluster allocates its driver and worker nodes from the pool. If the pool does not have sufficient idle resources to accommodate the cluster's request, the pool expands by allocating new instances from the instance provider. When an attached cluster is terminated, the instances it used are returned to the pool and can be reused by a different cluster.

See [Use a Pool](#) to learn more about working with pools in Databricks.

## Databricks Runtime

Databricks runtimes are the set of core components that run on your [clusters](#). All Databricks runtimes include Apache Spark and add components and updates that improve usability, performance, and security.

Databricks offers several types of runtimes and several versions of those runtime types in the **Databricks Runtime Version** drop-down when you create or edit a cluster.

For details, see [Databricks Runtimes](#).

## Python version

### Important!

In anticipation of the upcoming end of life of Python 2, announced for 2020, Python 2 is not supported in Databricks Runtime 6.0 and above. Databricks Runtime 5.5 and below continue to support Python 2.

## Python clusters running Databricks Runtime 6.0 and above

[Databricks Runtime 6.0](#) and above supports only Python 3. For major changes related to the Python environment introduced by Databricks Runtime 6.0, see [Python environment](#) in the release notes.

# Python clusters running Databricks Runtime with Conda (Beta)

## Python clusters running Databricks Runtime 5.5 and below

For Databricks Runtime 5.5 and below, Spark jobs, Python notebook cells, and library installation all support both Python 2 and 3 (since 2.0.2-db3).

The default Python version for clusters created using the UI is Python 3. In Databricks Runtime 5.5 and below the default version for clusters created using the REST API is Python 2.

### Specify Python version

To specify the Python version when you create a cluster using the UI, select it from the **Python Version** drop-down.




The screenshot shows a portion of the Databricks cluster configuration interface. It includes a text input field for 'Cluster Name'. Below it is a dropdown menu for 'Databricks Runtime Version' with the selected option '3.5 LTS (includes Apache Spark 2.2.1, Scala 2.11)'. Below that is another dropdown menu for 'Python Version' with the selected option '3'. The dropdown menu is open, showing options '2' and '3', with '3' highlighted in blue and a mouse cursor pointing at it.

To specify the Python version when you create a cluster using the API, set the environment variable `PYSPARK_PYTHON` to `/databricks/python/bin/python` or `/databricks/python3/bin/python3`. For an example, see the REST API example [Create a Python 3 cluster \(Databricks Runtime 5.5 and below\)](#).

To validate that the `PYSPARK_PYTHON` configuration took effect, in a Python notebook (or `%python` cell) run:


Python

 Copy

```
import sys
print(sys.version)
```

If you specified `/databricks/python3/bin/python3`, it should print something like:

Console

 Copy

### Important!

For Databricks Runtime 5.5 and below, when you run `%sh python --version` in a notebook, `python` refers to the Ubuntu system Python version, which is Python 2. Use `/databricks/python/bin/python` to refer to the version of Python used by Databricks notebooks and Spark: this path is automatically configured to point to the correct Python executable.

## Frequently asked questions (FAQ)

### Can I use both Python 2 and Python 3 notebooks on the same cluster?

No. The Python version is a cluster-wide setting and is not configurable on a per-notebook basis.

### What libraries are installed on Python clusters?

For details on the specific libraries that are installed, see the [Databricks Runtime Release Notes](#).

### Will my existing PyPI libraries work with Python 3?

It depends on whether the version of the library supports the Python 3 version of a Databricks Runtime version. Databricks Runtime 5.5 and below use Python 3.5. Databricks Runtime 6.0 and above, and Databricks Runtime with Conda use Python 3.7. It is possible that a specific old version of a Python library is not forward compatible with Python 3.7. For this case, you will need to use a newer version of the library.

### Will my existing `.egg` libraries work with Python 3?

It depends on whether your existing egg library is cross-compatible with both Python 2 and 3. If the library does not support Python 3 then either library attachment will fail or runtime errors will occur.

For a comprehensive guide on porting code to Python 3 and writing code compatible with both Python 2 and 3, see [Supporting Python 3](#).

### Can I still install Python libraries using `init` scripts?

A common use case for [Cluster Node Initialization Scripts](#) is to install packages. For Databricks Runtime 5.5 and below, use `/databricks/python/bin/pip` to ensure that Python packages

python and pip.

## Cluster node type

A cluster consists of one driver node and worker nodes. You can pick separate cloud provider instance types for the driver and worker nodes, although by default the driver node uses the same instance type as the worker node. Different families of instance types fit different use cases, such as memory-intensive or compute-intensive workloads.

### Driver node

The driver maintains state information of all notebooks attached to the cluster. The driver node is also responsible for maintaining the SparkContext and interpreting all the commands you run from a notebook or a library on the cluster. The driver node also runs the Apache Spark master that coordinates with the Spark executors.

The default value of the driver node type is the same as the worker node type. You can choose a larger driver node type with more memory if you are planning to `collect()` a lot of data from Spark workers and analyze them in the notebook.

#### Tip

Since the driver node maintains all of the state information of the notebooks attached, make sure to detach unused notebooks from the driver.

### Worker node

Databricks workers run the Spark executors and other services required for the proper functioning of the clusters. When you distribute your workload with Spark, all of the distributed processing happens on workers.

#### Tip

To run a Spark job, you need at least one worker. If a cluster has zero workers, you can run non-Spark commands on the driver, but Spark commands will fail.

## GPU instance types

For computationally challenging tasks that demand high performance, like those associated with deep learning, Databricks supports clusters accelerated with graphics processing units (GPUs). This support is in Beta. For more information, see [GPU-enabled Clusters](#).

## Cluster size and autoscaling

When you create a Databricks cluster, you can either provide a fixed number of workers for the cluster or provide a minimum and maximum number of workers for the cluster.

When you provide a fixed size cluster, Databricks ensures that your cluster has the specified number of workers. When you provide a range for the number of workers, Databricks chooses the appropriate number of workers required to run your job. This is referred to as *autoscaling*.

With autoscaling, Databricks dynamically reallocates workers to account for the characteristics of your job. Certain parts of your pipeline may be more computationally demanding than others, and Databricks automatically adds additional workers during these phases of your job (and removes them when they're no longer needed).

Autoscaling makes it easier to achieve high cluster utilization, because you don't need to provision the cluster to match a workload. This applies especially to workloads whose requirements change over time (like exploring a dataset during the course of a day), but it can also apply to a one-time shorter workload whose provisioning requirements are unknown. Autoscaling thus offers two advantages:

- Workloads can run faster compared to a constant-sized under-provisioned cluster.
- Autoscaling clusters can reduce overall costs compared to a statically-sized cluster.

Depending on the constant size of the cluster and the workload, autoscaling gives you one or both of these benefits at the same time. The cluster size can go below the minimum number of workers selected when the cloud provider terminates instances. In this case, Databricks continuously retries to re-provision instances in order to maintain the minimum number of workers.



- Autoscaling works best with Databricks Runtime 3.4 and above.
- Autoscaling is not available for spark-submit jobs.

Databricks offers two types of cluster node autoscaling: standard and optimized. For a discussion of the benefits of optimized autoscaling, see the blog post on [Optimized Autoscaling](#).

Automated (job) clusters always use optimized autoscaling. The type of autoscaling performed on interactive clusters depends on the workspace configuration.

Standard autoscaling is used by default on interactive clusters. Optimized autoscaling for interactive clusters is available on request. Contact your Databricks account manager to enable this feature.

## How autoscaling behaves

Autoscaling behaves differently depending on whether it is optimized or standard and whether applied to an interactive or an automated cluster.

### Optimized autoscaling

- Scales up from min to max in 2 steps.
- Can scale down even if the cluster is not idle by looking at shuffle file state.
- Scales down based on a percentage of current nodes.
- On automated clusters, scales down if the cluster is underutilized over the last 40 seconds.
- On interactive clusters, scales down if the cluster is underutilized over the last 150 seconds.

### Standard autoscaling

- Starts with adding 8 nodes. Thereafter, scales up exponentially, but can take many steps to reach the max. You can customize the first step by setting the `spark.databricks.autoscaling.standardFirstStepUp` Spark configuration property.
- Scales down only when the cluster is completely idle and it has been underutilized for the last 10 minutes.
- Scales down exponentially, starting with 1 node.

## Enable and configure autoscaling

To allow Databricks to resize your cluster automatically, you enable autoscaling for the cluster and provide the min and max range of workers.

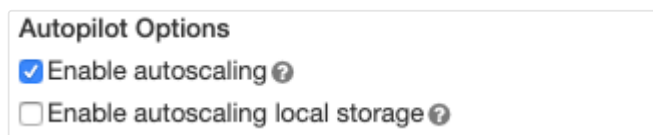
## 1. Enable autoscaling.

- **Interactive cluster** - On the **Create Cluster** page, select the **Enable autoscaling** checkbox



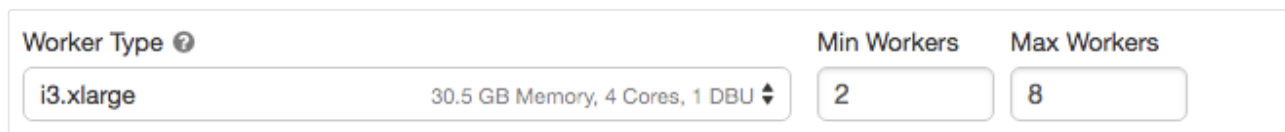
☒ Enable autoscaling ?  
☐ Enable autoscaling local storage ?  
☒ Terminate after  minutes of inactivity ?

- **Automated cluster** - On the **Configure Cluster** page, select the **Enable autoscaling** checkbox in the **Autopilot Options** box:



**Autopilot Options**  
☒ Enable autoscaling ?  
☐ Enable autoscaling local storage ?

## 2. Configure the min and max workers.



**Worker Type** ?  
i3.xlarge 30.5 GB Memory, 4 Cores, 1 DBU ⬆  
**Min Workers**  **Max Workers**

## Autoscaling example

If you reconfigure a static cluster to be an autoscaling cluster, Databricks immediately resizes the cluster within the minimum and maximum bounds and then starts autoscaling. As an example, the table below demonstrates what happens to clusters with a certain initial size if you reconfigure a cluster to autoscale between 5 and 10 nodes.

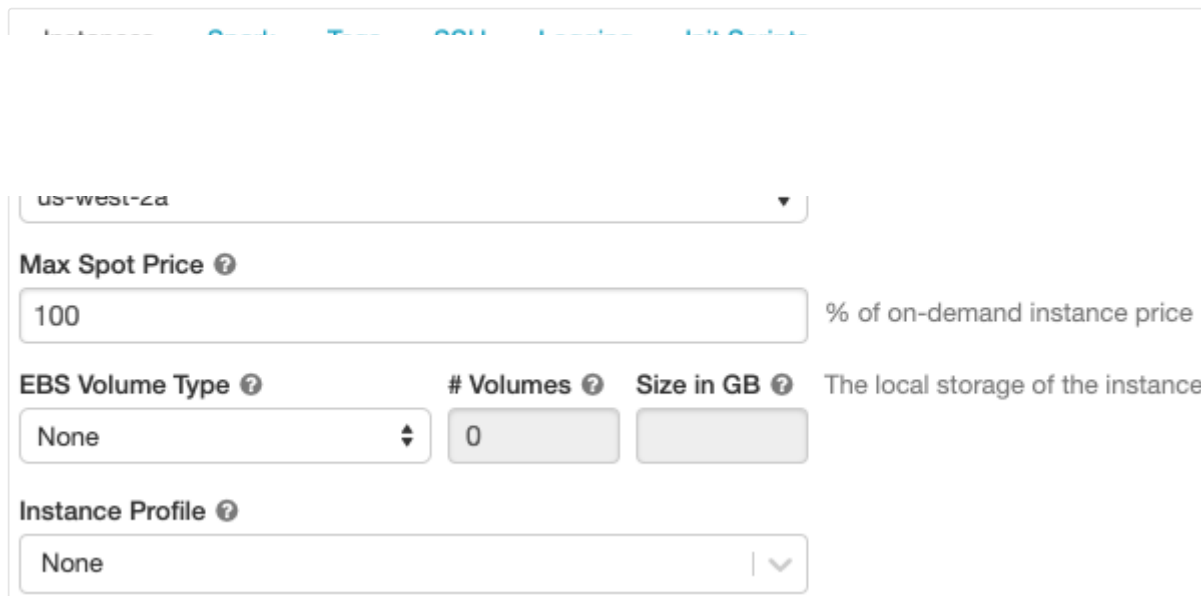
Initial size	Size after reconfiguration
6	6
12	10
3	5

## AWS configurations

When you configure a cluster's AWS instances you can choose the availability zone, the max spot price, EBS volume type and size, and IAM roles. To specify configurations,

1. On the cluster configuration page, click the **Advanced Options** toggle.

2. At the bottom of the page, click the **Instances** tab.



The screenshot shows the configuration panel for AWS instances in the 'Instances' tab. At the top, there is a dropdown menu for the region, currently set to 'us-west-2a'. Below this, the 'Max Spot Price' section includes a text input field with the value '100' and a label '% of on-demand instance price'. The 'EBS Volume Type' section features a dropdown menu set to 'None', a '# Volumes' input field with the value '0', and a 'Size in GB' input field. To the right of these fields is the text 'The local storage of the instance'. At the bottom, the 'Instance Profile' section has a dropdown menu set to 'None'.

## Availability zones

Choosing a specific availability zone for a cluster is useful primarily if your organization has purchased reserved instances in specific availability zones. Read more about [AWS availability zones](#).

## Max spot price

You can specify the max spot price to use when launching spot instances as a percentage of the corresponding on-demand price. By default, the max price is 100% of the on-demand price. Read more about [AWS spot pricing](#).

## EBS volumes

This section describes the default EBS volume settings for worker nodes, how to add shuffle volumes, and how to configure a cluster so that Databricks automatically allocates EBS volumes.

To configure EBS volumes, click the **Instances** tab in the cluster configuration and select an option in the EBS Volume Type drop-down list.

Instances

Spark

Tags

SSH

Logging

100

EBS Volume Type ?  
None

# Volumes ?

Size in GB ?

IAM Role ?  
None

## Default EBS volumes

Databricks provisions EBS volumes for every worker node as follows:

- A 30 GB unencrypted EBS instance root volume used only by the host operating system and Databricks internal services.
- A 150 GB encrypted EBS container root volume used by the Spark worker. This hosts Spark services and logs.
- (HIPAA only) a 75 GB encrypted EBS worker log volume that stores logs for Databricks internal services.

## Add EBS shuffle volumes

To add shuffle volumes, select **General Purpose SSD** in the EBS Volume Type drop-down list:

Instances

Spark

Tags

SSH

Logging

Availability Zone ?  
us-west-2c

Spot Bid Price ?  
100 % of on-demand instance price

EBS Volume Type ?  
General Purpose SSD  
None  
✓ General Purpose SSD  
Throughput Optimized HDD  
Autoscaling Local Storage

# Volumes ?  
3

Size in GB ?  
100

300 GB additional storage per node.

By default, Spark shuffle outputs go to the instance local disk. For instance types that do not have a local disk, or if you want to increase your Spark shuffle storage space, you can specify [Amazon EBS volumes](#) to store the shuffle data.

about [AWS EBS volumes](#).

## AWS EBS limits

Ensure that your AWS EBS limits are high enough to satisfy the runtime requirements for all workers in all clusters. For information on the default EBS limits and how to change them, see [Amazon Elastic Block Store \(EBS\) Limits](#).

## Autoscaling local storage

If you don't want to allocate a fixed number of EBS volumes at cluster creation time, use autoscaling local storage. With autoscaling local storage, Databricks monitors the amount of free disk space available on your cluster's Spark workers. If a worker begins to run too low on disk, Databricks automatically attaches a new EBS volume to the worker before it runs out of disk space. EBS volumes are attached up to a limit of 5 TB of total disk space per instance (including the instance's local storage).

To configure autoscaling storage, select **Enable autoscaling local storage** in the Autopilot Options box:

**Autopilot Options**

- ☒ Enable autoscaling ?
- ☒ Enable autoscaling local storage ⓘ
- ☒ Terminate after  minutes of inactivity ?

The EBS volumes attached to an instance are detached only when the instance is returned to AWS. That is, EBS volumes are never detached from an instance as long as it is part of a running cluster. To scale down EBS usage, Databricks recommends using this feature in a cluster configured with [Cluster size and autoscaling](#) or [Automatic termination](#).

### Note

Databricks uses Throughput Optimized HDD (st1) to extend the local storage of an instance. The [default AWS capacity limit](#) for these volumes is 20 TiB. To avoid hitting this limit, administrators should request an increase in this limit based on their usage requirements.

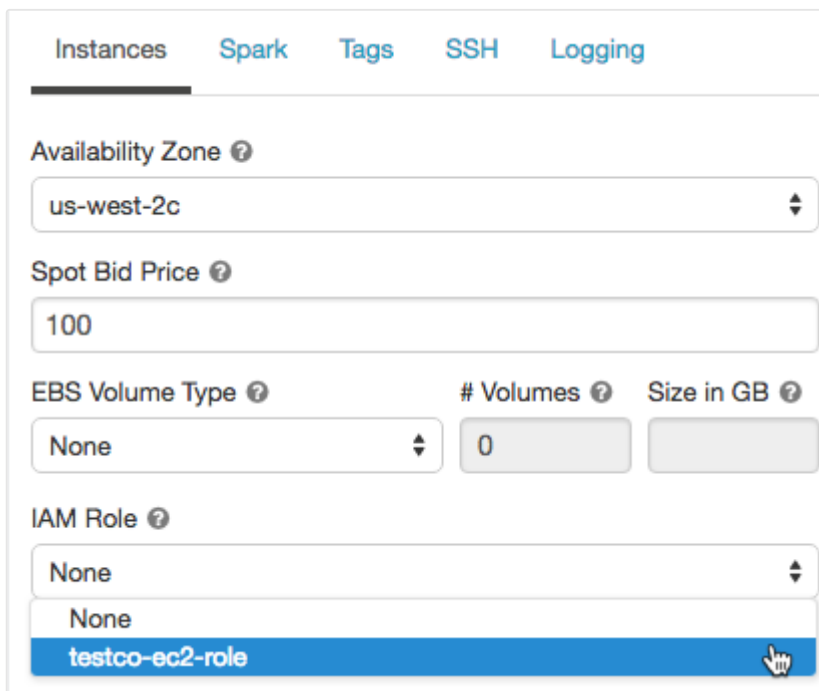
### Note

If you created your Databricks account prior to version 2.44 (that is, before Apr 27, 2017) and want to use autoscaling local storage (enabled by default in [High concurrency](#)

[Configure Your AWS Account.](#)

## IAM roles

To securely access AWS resources without using AWS keys, you can launch Databricks clusters with IAM roles. See [Secure Access to S3 Buckets Using IAM Roles](#) for information about how to create and configure IAM roles. Once you have created an IAM role, you select the role in the IAM Role drop-down list:



The screenshot shows the 'Instances' tab of the Databricks cluster configuration interface. The 'IAM Role' dropdown menu is open, showing three options: 'None', 'None', and 'testco-ec2-role'. The 'testco-ec2-role' option is highlighted in blue. Other configuration options visible include 'Availability Zone' set to 'us-west-2c', 'Spot Bid Price' set to '100', 'EBS Volume Type' set to 'None', '# Volumes' set to '0', and 'Size in GB' set to an empty field.

### Note

Once a cluster launches with an IAM role, anyone who has attach permissions to this cluster can access the underlying resources controlled by this role. To guard against unwanted access, you can use [Cluster Access Control](#) to restrict permissions to the cluster.

## Spark configuration

To fine tune Spark jobs, you can provide custom [Spark configuration properties](#) in a cluster configuration.

1. On the cluster configuration page, click the **Advanced Options** toggle.

2. Click the **Spark** tab.

**Spark Config** ⓘ

Enter your Spark configuration options here. Provide only one key-value pair per line.


Example:

```
spark.speculation true
spark.kryo.registrator my.package.MyRegistrator
```

When you configure a cluster using the [Clusters API](#), set Spark properties in the `spark_conf` field in the [Create cluster request](#) or [Edit cluster request](#).

To set Spark properties for all clusters, create a [global init script](#):

Scala

 Copy

```
dbutils.fs.put("dbfs:/databricks/init/set_spark_params.sh", ""
|#!/bin/bash
|
|cat << 'EOF' > /databricks/driver/conf/00-custom-spark-driver-defaults.conf
|[driver] {
|  "spark.sql.sources.partitionOverwriteMode" = "DYNAMIC"
|}
|EOF
|""".stripMargin, true)
```

## Environment variables

You can set environment variables that you can access from scripts running on a cluster.

1. On the cluster configuration page, click the **Advanced Options** toggle.

2. Click the **Spark** tab.

3. Set the environment variables in the **Environment Variables** field.

#### Environment Variables ⓘ

```
MY_ENVIRONMENT_VARIABLE=true
```

You can also set environment variables using the `spark_env_vars` field in the [Create cluster request](#) or [Edit cluster request](#) Clusters API endpoints.

#### Note

The environment variables you set in this field are not available in [Cluster Node Initialization Scripts](#). Init scripts support only a limited set of predefined [Environment variables](#).

## Cluster tags

Cluster tags allow you to easily monitor the cost of cloud resources used by various groups in your organization. You can specify tags as key-value pairs when you create a cluster, and Databricks applies these tags to cloud resources like VMs and disk volumes.

For convenience, Databricks applies four default tags to each cluster: `Vendor`, `Creator`, `ClusterName`, and `ClusterId`. You can add custom tags when you create a cluster. To configure cluster tags:

1. On the cluster configuration page, click the **Advanced Options** toggle.
2. At the bottom of the page, click the **Tags** tab.



Tags

Key	Value
ClusterName	DataEng
ClusterId	<Generated after creation>

Add Tag


3. Add a key-value pair for each custom tag. You can add up to 45 custom tags.

## Enforce mandatory tags

To ensure that certain tags are always populated when clusters are created, you can apply a specific IAM policy to your account's primary IAM role (the one created during account setup; contact your AWS administrator if you need access). The IAM policy should include explicit [Deny statements](#) for mandatory tag keys and optional values. *Cluster creation will fail* if required tags with one of the allowed values aren't provided.

For example, if you want to enforce **Department** and **Project** tags, with only specified values allowed for the former and a free-form non-empty value for the latter, you could apply an IAM policy like this one:

JSON

 Copy

```

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

  "Action": [
    "ec2:RunInstances",
    "ec2:CreateTags"
  ],
  "Resource": "arn:aws:ec2:region:accountId:instance/*",
  "Condition": {
    "StringNotEqualsIgnoreCase": {
      "aws:RequestTag/Department": [
        "Deptt1", "Deptt2", "Deptt3"
      ]
    }
  },
  {
    "Sid": "MandateLaunchWithTag2",
    "Effect": "Deny",
    "Action": [
      "ec2:RunInstances",
      "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:region:accountId:instance/*",
    "Condition": {
      "StringNotLike": {
        "aws:RequestTag/Project": "?*"
      }
    }
  }
]
}

```

Both `ec2:RunInstances` and `ec2:CreateTags` actions are required for each tag for there to be effective coverage of scenarios in which there are clusters that have only on-demand instances, only spot instances, or both.

### Tip

We recommend that you add a separate policy statement for each tag. The overall policy might become a bit verbose, but it is easier to debug. See the [IAM Policy Condition Operators Reference](#) for a list of operators that can be used in a policy.

## Note

Cloud Provider Launch Failure: A cloud provider error was encountered while setting up the cluster.

The message is encoded because the details of the authorization status can constitute privileged information that the user who requested the action should not see. See [DecodeAuthorizationMessage API](#) (or [CLI](#)) for information about how to decode such messages.

# SSH access to clusters

[SSH](#) allows you to log into Apache Spark clusters remotely for advanced troubleshooting and installing custom software.

This section describes how to configure your AWS account to enable ingress access to your cluster with your public key, and how to open an SSH connection to cluster nodes.

## Configure security group

You must update the Databricks security group in your AWS account to give ingress access to the IP address from which you will initiate the SSH connection. You can set this for a single IP address or provide a range that represents your entire office IP range.

1. In your AWS console, find the Databricks security group. It will have a label similar to `<databricks-instance>-worker-unmanaged`. (Example: `dbc-fb3asdddd3-worker-unmanaged`)
2. Edit the security group and add an inbound TCP rule to allow port `2200` to worker machines. It can be a single IP address or a range.

Edit inbound rules

Type

Protocol

Port Range

Source

Description

Cancel


Save

3. Make sure that your computer and office allow you to send TCP traffic on port 2200.

## Generate SSH key pair

Create an SSH key pair by running this command in a terminal session:

Bash

 Copy

```
ssh-keygen -t rsa -b 4096 -C "email@example.com"
```

You must provide the path to the directory where you want to save the public and private key. The public key is saved with the extension `.pub`.

## Configure a new cluster with your public key

1. Copy the entire contents of the public key file.
2. On the cluster configuration page, click the **Advanced Options** toggle.
3. At the bottom of the page, click the **SSH** tab.
4. Paste the key you copied into the **SSH Public Key** field.

InstancesSparkTagsSSHLogging

SSH Public Key ?

ssh-rsa <public\_key> email@example.com

# Configure an existing cluster with your public key

```
val publicKey = " put your public key here "


def addAuthorizedPublicKey(key: String): Unit = {
  val fw = new java.io.FileWriter("/home/ubuntu/.ssh/authorized_keys", /* append */
true)
  fw.write("\n" + key)
  fw.close()
}

val numExecutors = sc.getExecutorMemoryStatus.keys.size
sc.parallelize(0 until numExecutors, numExecutors).foreach { i =>
  addAuthorizedPublicKey(publicKey)
}
addAuthorizedPublicKey(publicKey)
```

## SSH into the Spark driver

1. On the cluster configuration page, click the **Advanced Options** toggle.
2. Click the **SSH** tab. Note the driver hostname.
3. Run the following command, replacing the hostname and private key file path.

Bash

 Copy

```
ssh ubuntu@<hostname> -p 2200 -i <private-key-file-path>
```

## SSH into Spark workers

You SSH into workers the same way that you SSH into the driver.

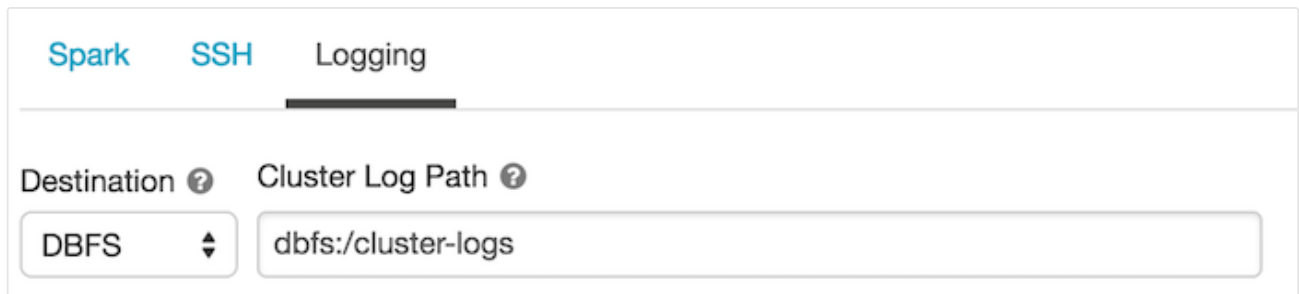
1. On the cluster details page, click the **Spark Cluster UI - Master** tab.
2. In the Workers table, click the worker that you want to SSH into. Copy the Hostname field.

When you create a cluster, you can specify a location to deliver Spark driver and worker logs. Logs are delivered every five minutes to your chosen destination. When a cluster is terminated, Databricks guarantees to deliver all logs generated up until the cluster was terminated.

The destination of the logs depends on the cluster ID. If the specified destination is `dbfs:/cluster-log-delivery`, cluster logs for `0630-191345-leap375` are delivered to `dbfs:/cluster-log-delivery/0630-191345-leap375`.

To configure the log delivery location:

1. On the cluster configuration page, click the **Advanced Options** toggle.
2. At the bottom of the page, click the **Logging** tab.



The screenshot shows the 'Logging' configuration tab in the Databricks UI. It features two input fields: 'Destination' with a dropdown menu currently showing 'DBFS', and 'Cluster Log Path' with a text input field containing 'dbfs:/cluster-logs'. Both fields have a question mark icon to the right of their labels.

3. Select a destination type.
4. Enter the cluster log path.

## S3 bucket destinations

If you choose an S3 destination, you must configure the cluster with an IAM role that can access the bucket. This IAM role must have both the `PutObject` and `PutObjectAcl` permissions. An example IAM role has been included below for your convenience. See [Secure Access to S3 Buckets Using IAM Roles](#) for instructions on how to set up an IAM role.

```

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

  "s3:ListBucket"
],
"Resource": [
  "arn:aws:s3:::<my-s3-bucket>"
],
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject",
    "s3:PutObjectAcl",
    "s3:GetObject",
    "s3:DeleteObject"
  ],
  "Resource": [
    "arn:aws:s3:::<my-s3-bucket>/*"
  ]
}
]
}

```

### Note

This feature is also available in the REST API. See [Clusters API](#) and [Cluster log delivery examples](#).

## Init scripts

A cluster node initialization—or init—script is a shell script that runs during startup for each cluster node *before* the Spark driver or worker JVM starts. You can use init scripts to install packages and libraries not included in the Databricks runtime, modify the JVM system classpath, set system properties and environment variables used by the JVM, or modify Spark configuration parameters, among other configuration tasks.

You can attach init scripts to a cluster by expanding the **Advanced Options** section and clicking the **Init Scripts** tab.

For detailed instructions, see [Cluster Node Initialization Scripts](#).

Was this article helpful?

Yes

No

---

© Databricks 2020. All rights reserved. Apache, Apache Spark, Spark, and the Spark logo are trademarks of the [Apache Software Foundation](#).

[Send us feedback](#) | [Privacy Policy](#) | [Terms of Use](#)