# Table of Contents

- About Amazon EMR Releases ........................................................................................................... ix
  - Applications ................................................................................................................................. 1
  - Components ............................................................................................................................... 3
  - Learn More ................................................................................................................................ 6
- What's New? ................................................................................................................................... 7
  - Release 5.11.1 (Latest) ................................................................................................................ 7
    - Changes, Enhancements, and Resolved Issues .......................................................................... 7
    - Known Issues ......................................................................................................................... 7
  - Release 5.11.0 ........................................................................................................................... 7
    - Upgrades ................................................................................................................................. 7
    - New Features .......................................................................................................................... 8
    - Known Issues ........................................................................................................................ 8
  - Release 5.10.0 ........................................................................................................................... 8
    - Upgrades ................................................................................................................................. 8
    - New Features .......................................................................................................................... 8
    - Changes, Enhancements, and Resolved Issues ...................................................................... 9
    - Known Issues ........................................................................................................................ 9
  - Release 5.9.0 ........................................................................................................................... 9
    - Upgrades ................................................................................................................................. 10
    - New Features .......................................................................................................................... 10
    - Changes, Enhancements, and Resolved Issues .................................................................... 10
    - Known Issues ........................................................................................................................ 10
  - Release 5.8.1 ........................................................................................................................... 11
    - Changes, Enhancements, and Resolved Issues ...................................................................... 11
  - Release 5.8.0 ........................................................................................................................... 11
    - Upgrades ................................................................................................................................. 11
    - New Features .......................................................................................................................... 11
    - Changes, Enhancements, and Resolved Issues .................................................................... 11
    - Known Issues ........................................................................................................................ 12
  - Release 5.7.0 ........................................................................................................................... 12
    - Upgrades ................................................................................................................................. 12
    - New Features .......................................................................................................................... 13
    - Changes, Enhancements, and Resolved Issues .................................................................... 13
  - Release 5.6.0 ........................................................................................................................... 13
    - Upgrades ................................................................................................................................. 13
    - Changes, Enhancements, and Resolved Issues .................................................................... 13
  - Release 5.5.1 ........................................................................................................................... 13
    - Changes, Enhancements, and Resolved Issues .................................................................... 13
  - Release 5.5.0 ........................................................................................................................... 14
    - Upgrades ................................................................................................................................. 14
    - Changes, Enhancements, and Resolved Issues .................................................................... 14
  - Release 5.4.0 ........................................................................................................................... 15
    - Changes and Enhancements ................................................................................................. 15
  - Release 5.3.0 ........................................................................................................................... 15
    - Upgrades ................................................................................................................................. 15
    - Changes and Enhancements ................................................................................................. 16
  - Release 5.2.2 ........................................................................................................................... 16
    - Known Issues Resolved from the Previous Releases .......................................................... 16
  - Release 5.2.1 ........................................................................................................................... 16
    - Upgrades ................................................................................................................................. 16
    - Changes and Enhancements ................................................................................................. 16
    - Known Issues Resolved from the Previous Releases .......................................................... 17
<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>28</td>
</tr>
<tr>
<td>Known Issues Resolved from the Previous Releases</td>
<td>28</td>
</tr>
<tr>
<td>Configuring Applications</td>
<td>30</td>
</tr>
<tr>
<td>Configuring Applications to Use a Specific Java Virtual Machine</td>
<td>35</td>
</tr>
<tr>
<td>Service ports</td>
<td>36</td>
</tr>
<tr>
<td>Application users</td>
<td>37</td>
</tr>
<tr>
<td>Hadoop</td>
<td>39</td>
</tr>
<tr>
<td>Hadoop Release Information for This Release of Amazon EMR</td>
<td>39</td>
</tr>
<tr>
<td>Create or Run a Hadoop Application</td>
<td>39</td>
</tr>
<tr>
<td>Build Binaries Using Amazon EMR</td>
<td>39</td>
</tr>
<tr>
<td>Run a Script in a Cluster</td>
<td>41</td>
</tr>
<tr>
<td>Process Data with Streaming</td>
<td>42</td>
</tr>
<tr>
<td>Process Data with a Custom JAR</td>
<td>45</td>
</tr>
<tr>
<td>Configure Hadoop</td>
<td>47</td>
</tr>
<tr>
<td>Hadoop Daemon Settings</td>
<td>47</td>
</tr>
<tr>
<td>HDFS Configuration</td>
<td>61</td>
</tr>
<tr>
<td>Task Configuration</td>
<td>66</td>
</tr>
<tr>
<td>Ganglia</td>
<td>84</td>
</tr>
<tr>
<td>Ganglia Release Information for This Release of Amazon EMR</td>
<td>84</td>
</tr>
<tr>
<td>Add Ganglia to a Cluster</td>
<td>84</td>
</tr>
<tr>
<td>View Ganglia Metrics</td>
<td>85</td>
</tr>
<tr>
<td>Hadoop and Spark Metrics in Ganglia</td>
<td>86</td>
</tr>
<tr>
<td>HBase</td>
<td>87</td>
</tr>
<tr>
<td>HBase Release Information for This Release of Amazon EMR</td>
<td>87</td>
</tr>
<tr>
<td>Creating a Cluster with HBase</td>
<td>88</td>
</tr>
<tr>
<td>Creating a Cluster with HBase Using the Console</td>
<td>88</td>
</tr>
<tr>
<td>Creating a Cluster with HBase Using the AWS CLI</td>
<td>88</td>
</tr>
<tr>
<td>HBase on Amazon S3 (Amazon S3 Storage Mode)</td>
<td>89</td>
</tr>
<tr>
<td>Enabling HBase on Amazon S3</td>
<td>90</td>
</tr>
<tr>
<td>Using a Read-Replica Cluster</td>
<td>90</td>
</tr>
<tr>
<td>Operational Considerations</td>
<td>91</td>
</tr>
<tr>
<td>Using the HBase Shell</td>
<td>93</td>
</tr>
<tr>
<td>Create a Table</td>
<td>93</td>
</tr>
<tr>
<td>Put a Value</td>
<td>94</td>
</tr>
<tr>
<td>Get a Value</td>
<td>94</td>
</tr>
<tr>
<td>Access HBase Tables with Hive</td>
<td>94</td>
</tr>
<tr>
<td>Using HBase Snapshots</td>
<td>95</td>
</tr>
<tr>
<td>Create a snapshot using a table</td>
<td>95</td>
</tr>
<tr>
<td>Delete a snapshot</td>
<td>96</td>
</tr>
<tr>
<td>View snapshot info</td>
<td>96</td>
</tr>
<tr>
<td>Export a snapshot to Amazon S3</td>
<td>96</td>
</tr>
<tr>
<td>Import snapshot from Amazon S3</td>
<td>96</td>
</tr>
<tr>
<td>Restore a table from snapshots within the HBase shell</td>
<td>97</td>
</tr>
<tr>
<td>Configure HBase</td>
<td>98</td>
</tr>
<tr>
<td>Changes to Memory Allocation in YARN</td>
<td>98</td>
</tr>
<tr>
<td>HBase Port Numbers</td>
<td>98</td>
</tr>
<tr>
<td>HBase Site Settings to Optimize</td>
<td>99</td>
</tr>
<tr>
<td>View the HBase User Interface</td>
<td>100</td>
</tr>
<tr>
<td>View HBase Log Files</td>
<td>101</td>
</tr>
<tr>
<td>Monitor HBase with Ganglia</td>
<td>102</td>
</tr>
<tr>
<td>Migrating from Previous HBase Versions</td>
<td>103</td>
</tr>
<tr>
<td>HCatalog</td>
<td>104</td>
</tr>
<tr>
<td>HCatalog Release Information for This Release of Amazon EMR</td>
<td>104</td>
</tr>
<tr>
<td>Creating a Cluster with HCatalog</td>
<td>104</td>
</tr>
<tr>
<td>Using HCatalog</td>
<td>105</td>
</tr>
<tr>
<td>Disable Direct Write When Using HCatalog HStorer</td>
<td>105</td>
</tr>
<tr>
<td>Create a Table Using the HCat CLI and Use That Data in Pig</td>
<td>105</td>
</tr>
</tbody>
</table>
About Amazon EMR Releases

This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

This document provides information about Amazon EMR 4.x and 5.x software releases. A release is a set of software applications and components that can be installed and configured on an Amazon EMR cluster. Amazon EMR releases are packaged using a system based on Apache BigTop, which is an open source project associated with the Hadoop ecosystem. In addition to Hadoop and Spark ecosystem projects, each Amazon EMR release provides components that enable cluster and resource management, interoperability with other AWS services, and additional configuration optimizations for installed software.

Topics

- Applications (p. 1)
- Components (p. 3)
- Learn More (p. 6)

Applications

Each Amazon EMR release contains several distributed applications available for installation on your cluster. Amazon EMR defines each application as not only the set of the components which comprise that open source project but also a set of associated components which are required for that the application to function. When you choose to install an application using the console, API, or CLI, Amazon EMR installs and configures this set of components across nodes in your cluster. The following applications are supported for this release: Flink, Ganglia, Hadoop, HBase, HCatalog, Hive, Hue, Livy, Mahout, MXNet, Oozie, Phoenix, Pig, Presto, Spark, Sqoop, Tez, Zeppelin, and ZooKeeper.
Components

The Amazon EMR releases include various components that can be installed by specifying an application which uses them. The versions of these components are typically those found in the community. Amazon EMR makes an effort to make community releases available in a timely fashion. However, there may be a need to make changes to specific components. If those components are modified, they have a release version such as the following:

\[ \text{communityVersion-amzn-emrReleaseVersion} \]

As an example, assume that the component, `ExampleComponent1`, has not been modified by Amazon EMR; the version is `1.0`, which is the community version. However, another component, `ExampleComponent2`, is modified and its Amazon EMR release version is `1.0.0-amzn-0`.

There are also components provided exclusively by Amazon EMR. For example, the DynamoDB connector component, `emr-ddb`, is provided by Amazon EMR for use with applications running on Amazon EMR clusters. Amazon components have just one version number. For example, an `emr-ddb` version is `2.1.0`. For more information about using Hive to query DynamoDB and an example, see Amazon EMR Hive Queries to Accommodate Partial DynamoDB Schemas (p. 111).

The following components are included with Amazon EMR:

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws-hm-client</td>
<td>1.3.0</td>
<td>Amazon Glue connector for Hadoop ecosystem applications integrated with the Hive Metastore.</td>
</tr>
<tr>
<td>aws-sagemaker-spark-sdk</td>
<td>1.0</td>
<td>Amazon SageMaker Spark SDK</td>
</tr>
<tr>
<td>emr-ddb</td>
<td>4.5.0</td>
<td>Amazon DynamoDB connector for Hadoop ecosystem applications.</td>
</tr>
<tr>
<td>emr-goodies</td>
<td>2.4.0</td>
<td>Extra convenience libraries for the Hadoop ecosystem.</td>
</tr>
<tr>
<td>emr-kinesis</td>
<td>3.4.0</td>
<td>Amazon Kinesis connector for Hadoop ecosystem applications.</td>
</tr>
<tr>
<td>emr-s3-dist-cp</td>
<td>2.8.0</td>
<td>Distributed copy application optimized for Amazon S3.</td>
</tr>
<tr>
<td>emrfs</td>
<td>2.20.0</td>
<td>Amazon S3 connector for Hadoop ecosystem applications.</td>
</tr>
<tr>
<td>flink-client</td>
<td>1.3.2</td>
<td>Apache Flink command line client scripts and applications.</td>
</tr>
<tr>
<td>ganglia-monitor</td>
<td>3.7.2</td>
<td>Embedded Ganglia agent for Hadoop ecosystem applications along with the Ganglia monitoring agent.</td>
</tr>
<tr>
<td>ganglia-metadata-collector</td>
<td>3.7.2</td>
<td>Ganglia metadata collector for aggregating metrics from Ganglia monitoring agents.</td>
</tr>
<tr>
<td>Component</td>
<td>Version</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ganglia-web</td>
<td>3.7.1</td>
<td>Web application for viewing metrics collected by the Ganglia metadata collector.</td>
</tr>
<tr>
<td>hadoop-client</td>
<td>2.7.3-amzn-6</td>
<td>Hadoop command-line clients such as 'hdfs', 'hadoop', or 'yarn'.</td>
</tr>
<tr>
<td>hadoop-hdfs-datanode</td>
<td>2.7.3-amzn-6</td>
<td>HDFS node-level service for storing blocks.</td>
</tr>
<tr>
<td>hadoop-hdfs-library</td>
<td>2.7.3-amzn-6</td>
<td>HDFS command-line client and library</td>
</tr>
<tr>
<td>hadoop-hdfs-namenode</td>
<td>2.7.3-amzn-6</td>
<td>HDFS service for tracking file names and block locations.</td>
</tr>
<tr>
<td>hadoop-httpfs-server</td>
<td>2.7.3-amzn-6</td>
<td>HTTP endpoint for HDFS operations.</td>
</tr>
<tr>
<td>hadoop-kms-server</td>
<td>2.7.3-amzn-6</td>
<td>Cryptographic key management server based on Hadoop's KeyProvider API.</td>
</tr>
<tr>
<td>hadoop-mapred</td>
<td>2.7.3-amzn-6</td>
<td>MapReduce execution engine libraries for running a MapReduce application.</td>
</tr>
<tr>
<td>hadoop-yarn-nodemanager</td>
<td>2.7.3-amzn-6</td>
<td>YARN service for managing containers on an individual node.</td>
</tr>
<tr>
<td>hadoop-yarn-resourcemanager</td>
<td>2.7.3-amzn-6</td>
<td>YARN service for allocating and managing cluster resources and distributed applications.</td>
</tr>
<tr>
<td>hadoop-yarn-timeline-server</td>
<td>2.7.3-amzn-6</td>
<td>Service for retrieving current and historical information for YARN applications.</td>
</tr>
<tr>
<td>hbase-hmaster</td>
<td>1.3.1</td>
<td>Service for an HBase cluster responsible for coordination of Regions and execution of administrative commands.</td>
</tr>
<tr>
<td>hbase-region-server</td>
<td>1.3.1</td>
<td>Service for serving one or more HBase regions.</td>
</tr>
<tr>
<td>hbase-client</td>
<td>1.3.1</td>
<td>HBase command-line client.</td>
</tr>
<tr>
<td>hbase-rest-server</td>
<td>1.3.1</td>
<td>Service providing a RESTful HTTP endpoint for HBase.</td>
</tr>
<tr>
<td>hbase-thrift-server</td>
<td>1.3.1</td>
<td>Service providing a Thrift endpoint to HBase.</td>
</tr>
<tr>
<td>hcatalog-client</td>
<td>2.3.2-amzn-0</td>
<td>The 'hcat' command line client for manipulating hcatalog-server.</td>
</tr>
<tr>
<td>Component</td>
<td>Version</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hcatalog-server</td>
<td>2.3.2-amzn-0</td>
<td>Service providing HCatalog, a table and storage management layer for distributed applications.</td>
</tr>
<tr>
<td>hcatalog-webhcat-server</td>
<td>2.3.2-amzn-0</td>
<td>HTTP endpoint providing a REST interface to HCatalog.</td>
</tr>
<tr>
<td>hive-client</td>
<td>2.3.2-amzn-0</td>
<td>Hive command line client.</td>
</tr>
<tr>
<td>hive-hbase</td>
<td>2.3.2-amzn-0</td>
<td>Hive-hbase client.</td>
</tr>
<tr>
<td>hive-metastore-server</td>
<td>2.3.2-amzn-0</td>
<td>Service for accessing the Hive metastore, a semantic repository storing metadata for SQL on Hadoop operations.</td>
</tr>
<tr>
<td>hive-server2</td>
<td>2.3.2-amzn-0</td>
<td>Service for accepting Hive queries as web requests.</td>
</tr>
<tr>
<td>hue-server</td>
<td>4.0.1</td>
<td>Web application for analyzing data using Hadoop ecosystem applications</td>
</tr>
<tr>
<td>livy-server</td>
<td>0.4.0-incubating</td>
<td>REST interface for interacting with Apache Spark</td>
</tr>
<tr>
<td>kerberos-ad-joiner</td>
<td>1.15.1</td>
<td>Security component for Active Directory joining</td>
</tr>
<tr>
<td>kerberos-server</td>
<td>1.15.1</td>
<td>Security application</td>
</tr>
<tr>
<td>kerberos-client</td>
<td>1.15.1</td>
<td>Security application</td>
</tr>
<tr>
<td>mahout-client</td>
<td>0.13.0</td>
<td>Library for machine learning.</td>
</tr>
<tr>
<td>mxnet</td>
<td>0.12.0</td>
<td>A flexible, scalable, and efficient library for deep learning.</td>
</tr>
<tr>
<td>mysql-server</td>
<td>5.5.54+</td>
<td>MySQL database server</td>
</tr>
<tr>
<td>oozie-client</td>
<td>4.3.0</td>
<td>Oozie command-line client.</td>
</tr>
<tr>
<td>oozie-server</td>
<td>4.3.0</td>
<td>Service for accepting Oozie workflow requests.</td>
</tr>
<tr>
<td>phoenix-library</td>
<td>4.11.0-HBase-1.3</td>
<td>The phoenix libraries for server and client</td>
</tr>
<tr>
<td>phoenix-query-server</td>
<td>4.11.0-HBase-1.3</td>
<td>A light weight server providing JDBC access as well as Protocol Buffers and JSON format access to the Avatica API</td>
</tr>
<tr>
<td>presto-coordinator</td>
<td>0.187</td>
<td>Service for accepting queries and managing query execution among presto-workers.</td>
</tr>
<tr>
<td>Component</td>
<td>Version</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>presto-worker</td>
<td>0.187</td>
<td>Service for executing pieces of a query.</td>
</tr>
<tr>
<td>pig-client</td>
<td>0.17.0</td>
<td>Pig command-line client.</td>
</tr>
<tr>
<td>spark-client</td>
<td>2.2.1</td>
<td>Spark command-line clients.</td>
</tr>
<tr>
<td>spark-history-server</td>
<td>2.2.1</td>
<td>Web UI for viewing logged events for the lifetime of a completed Spark application.</td>
</tr>
<tr>
<td>spark-on-yarn</td>
<td>2.2.1</td>
<td>In-memory execution engine for YARN.</td>
</tr>
<tr>
<td>spark-yarn-slave</td>
<td>2.2.1</td>
<td>Apache Spark libraries needed by YARN slaves.</td>
</tr>
<tr>
<td>sqoop-client</td>
<td>1.4.6</td>
<td>Apache Sqoop command-line client.</td>
</tr>
<tr>
<td>tez-on-yarn</td>
<td>0.8.4</td>
<td>The tez YARN application and libraries.</td>
</tr>
<tr>
<td>webserver</td>
<td>2.4.25+</td>
<td>Apache HTTP server.</td>
</tr>
<tr>
<td>zeppelin-server</td>
<td>0.7.3</td>
<td>Web-based notebook that enables interactive data analytics.</td>
</tr>
<tr>
<td>zeppelin-user</td>
<td>0.7.3</td>
<td>Zeppelin component to create zeppelin user in slave nodes when Kerberos enabled.</td>
</tr>
<tr>
<td>zookeeper-server</td>
<td>3.4.10</td>
<td>Centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.</td>
</tr>
<tr>
<td>zookeeper-client</td>
<td>3.4.10</td>
<td>ZooKeeper command line client.</td>
</tr>
</tbody>
</table>

**Learn More**

If you are looking for additional information, see the following guides and sites:

- Information about the Amazon EMR service, getting started, and how to launch or manage clusters, specifically for emr-4.0.0 or greater — Amazon EMR Management Guide
- Amazon EMR API Reference
- AWS SDKs and other tools
- AWS CLI Command Reference
- Information about Amazon EMR AMI versions 2.x and 3.x — Amazon EMR Developer Guide
What's New?

This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

This chapter gives an overview of features and issues resolved in the current release of Amazon EMR as well as the historical record of this information for earlier-version releases back to version 4.2.0.

Release 5.11.1 (Latest)

The following release notes include information for the Amazon EMR version 5.11.1 release. Changes are relative to the Amazon EMR 5.11.0 release.

Initial release date: January 22, 2018

Changes, Enhancements, and Resolved Issues


Known Issues

- MXNet does not include OpenCV libraries.
- Hive 2.3.2 sets hive.compute.stats.using.query=true by default. This causes queries to get data from existing statistics rather than directly from data, which could be confusing. For example, if you have a table with hive.compute.stats.using.query=true and upload new files to the table LOCATION, running a SELECT COUNT(*) query on the table returns the count from the statistics, rather than picking up the added rows.

As a workaround, use the ANALYZE TABLE command to gather new statistics, or set hive.compute.stats.using.query=false. For more information, see Statistics in Hive in the Apache Hive documentation.

Release 5.11.0

The following release notes include information for the Amazon EMR version 5.11.0 release. Changes are relative to the Amazon EMR 5.10.0 release.

Upgrades

The following applications and components have been upgraded in this release to include the following versions.
New Features

- Spark
  - Added `spark.decommissioning.timeout.threshold` setting, which improves Spark decommissioning behavior when using Spot instances. For more information, see Configuring Node Decommissioning Behavior (p. 149).
  - Added the `aws-sagemaker-spark-sdk` component to Spark, which installs Amazon SageMaker Spark and associated dependencies for Spark integration with Amazon SageMaker. You can use Amazon SageMaker Spark to construct Spark machine learning (ML) pipelines using Amazon SageMaker stages. For more information, see the SageMaker Spark Readme on GitHub and Using Apache Spark with Amazon SageMaker in the Amazon SageMaker Developer Guide.

Known Issues

- MXNet does not include OpenCV libraries.
- Hive 2.3.2 sets `hive.compute.stats.using.query=true` by default. This causes queries to get data from existing statistics rather than directly from data, which could be confusing. For example, if you have a table with `hive.compute.stats.using.query=true` and upload new files to the table `LOCATION`, running a `SELECT COUNT(*)` query on the table returns the count from the statistics, rather than picking up the added rows.

  As a workaround, use the `ANALYZE TABLE` command to gather new statistics, or set `hive.compute.stats.using.query=false`. For more information, see Statistics in Hive in the Apache Hive documentation.

Release 5.10.0

The following release notes include information for the Amazon EMR version 5.10.0 release. Changes are relative to the Amazon EMR 5.9.0 release.

Upgrades

The following applications and components have been upgraded in this release to include the following versions.

- AWS SDK for Java 1.11.221
- Hive 2.3.1
- Presto 0.187

New Features

- Added support for Kerberos authentication. For more information, see Use Kerberos Authentication in the Amazon EMR Management Guide
- Added support for EMRFS authorization. For more information, see Use EMRFS Authorization for Data in Amazon S3 in the Amazon EMR Management Guide
• Added support for GPU-based P2 and P3 instance types. For more information, see [Amazon EC2 P2 Instances](https://aws.amazon.com/ec2/instance-types/) and [Amazon EC2 P3 Instances](https://aws.amazon.com/ec2/instance-types/). NVIDIA driver 384.81 and CUDA driver 9.0.176 are installed on these instance types by default.

• Added support for [Apache MXNet](https://mxnet.apache.org) (p. 128).

### Changes, Enhancements, and Resolved Issues

#### Presto

• Added support for using the AWS Glue Data Catalog as the default Hive metastore. For more information, see [Using Presto with the AWS Glue Data Catalog](https://aws.amazon.com/rds/).  

• Added support for geospatial functions.

• Added spill to disk support for joins.

• Added support for the Redshift connector.

#### Spark

• Backported SPARK-20640, which makes the rpc timeout and the retries for shuffle registration values configurable using `spark.shuffle.registration.timeout` and `spark.shuffle.registration.maxAttempts` properties.

• Backported SPARK-21549, which corrects an error that occurs when writing custom OutputFormat to non-HDFS locations.

• Backported Hadoop-13270

• The Numpy, Scipy, and Matplotlib libraries have been removed from the base Amazon EMR AMI. If these libraries are required for your application, they are available in the application repository, so you can use a bootstrap action to install them on all nodes using `yum install`.

• The Amazon EMR base AMI no longer has application RPM packages included, so the RPM packages are no longer present on cluster nodes. Custom AMIs and the Amazon EMR base AMI now reference the RPM package repository in Amazon S3.

• Because of the introduction of per-second billing in Amazon EC2, the default Scale down behavior is now **Terminate at task completion** rather than **Terminate at instance hour**. For more information, see [Configure Cluster Scale-Down](https://aws.amazon.com/rds/).

### Known Issues

• MXNet does not include OpenCV libraries.

• Hive 2.3.1 sets `hive.compute.stats.using.query=true` by default. This causes queries to get data from existing statistics rather than directly from data, which could be confusing. For example, if you have a table with `hive.compute.stats.using.query=true` and upload new files to the table location, running a `SELECT COUNT(*)` query on the table returns the count from the statistics, rather than picking up the added rows.

As a workaround, use the `ANALYZE TABLE` command to gather new statistics, or set `hive.compute.stats.using.query=false`. For more information, see [Statistics in Hive](https://aws.amazon.com/rds/) in the [Apache Hive documentation](https://aws.amazon.com/rds/).

### Release 5.9.0

The following release notes include information for the Amazon EMR version 5.9.0 release. Changes are relative to the Amazon EMR 5.8.0 release.

Release date: October 5, 2017
Upgrades

The following applications and components have been upgraded in this release to include the following versions.

- AWS SDK for Java version 1.11.183
- Flink 1.3.2
- Hue 4.0.1
- Pig 0.17.0
- Presto 0.184

New Features

- Added Livy support (version 0.4.0-incubating). For more information, see Apache Livy (p. 126).
- Added support for Hue Notebook for Spark.
- Added support for i3-series Amazon EC2 instances (October 12, 2017).

Changes, Enhancements, and Resolved Issues

- Spark
  - Added a new set of features that help ensure Spark handles node termination because of a manual resize or an automatic scaling policy request more gracefully. For more information, see Configuring Node Decommissioning Behavior (p. 149).
  - SSL is used instead of 3DES for in-transit encryption for the block transfer service, which enhances performance when using Amazon EC2 instance types with AES-NI.
  - Backported SPARK-21494.
- Zeppelin
  - Backported ZEPPELIN-2377.
- HBase
  - Added patch HBASE-18533, which allows additional values for HBase BucketCache configuration using the hbase-site configuration classification.
- Hue
  - Added AWS Glue Data Catalog support for the Hive query editor in Hue.
  - By default, superusers in Hue can access all files that Amazon EMR IAM roles are allowed to access. Newly created users do not automatically have permissions to access the Amazon S3 filebrowser and must have the filebrowser.s3_access permissions enabled for their group.
  - Resolved an issue that caused underlying JSON data created using AWS Glue Data Catalog to be inaccessible.

Known Issues

- Cluster launch fails when all applications are installed and the default Amazon EBS root volume size is not changed. As a workaround, use the aws emr create-cluster command from the AWS CLI and specify a larger --ebs-root-volume-size parameter.
- Hive 2.3.0 sets hive.compute.stats.using.query=true by default. This causes queries to get data from existing statistics rather than directly from data, which could be confusing. For example, if
you have a table with hive.compute.stats.using.query=true and upload new files to the table
LOCATION, running a SELECT COUNT(*) query on the table returns the count from the statistics,
rather than picking up the added rows.

As a workaround, use the ANALYZE TABLE command to gather new statistics, or set
hive.compute.stats.using.query=false. For more information, see Statistics in Hive in the
Apache Hive documentation.

**Release 5.8.1**

The following release notes include information for the Amazon EMR version 5.8.1 release. Changes are
relative to the Amazon EMR 5.8.0 release.

Initial release date: January 22, 2018

**Changes, Enhancements, and Resolved Issues**

- Updated the Amazon Linux kernel of the default Amazon Linux AMI for Amazon EMR to address
vulnerabilities associated with speculative execution (CVE-2017-5715, CVE-2017-5753, and
CVE-2017-5754). For more information, see https://aws.amazon.com/security/security-bulletins/
AWS-2018-013/.

**Release 5.8.0**

The following release notes include information for the Amazon EMR version 5.8.0 release. Changes are
relative to the Amazon EMR 5.7.0 release.

Initial release date: August 10, 2017

Latest feature update: September 25, 2017

**Upgrades**

The following applications and components have been upgraded in this release to include the following
versions:

- AWS SDK 1.11.160
- Flink 1.3.1
- Hive 2.3.0. For more information, see Release Notes on the Apache Hive site.
- Spark 2.2.0. For more information, see Release Notes on the Apache Spark site.

**New Features**

- Added support for viewing application history (September 25, 2017). For more information, see
Viewing Application History in the Amazon EMR Management Guide.

**Changes, Enhancements, and Resolved Issues**

- Integration with AWS Glue Data Catalog
• Added ability for Hive and Spark SQL to use AWS Glue Data Catalog as the Hive metadata store. For more information, see Using the AWS Glue Data Catalog as the Metastore for Hive (p. 113) and Using the AWS Glue Data Catalog as the Metastore for Spark SQL (p. 145).

• Added **Application history** to cluster details, which allows you to view historical data for YARN applications and additional details for Spark applications. For more information, see View Application History in the *Amazon EMR Management Guide*.

• **Oozie**
  • Backported OOZIE-2748.

• **Hue**
  • Backported HUE-5859

• **HBase**
  • Added patch to expose the HBase master server start time through Java Management Extensions (JMX) using `getMasterInitializedTime`.
  • Added patch that improves cluster start time.

### Known Issues

• Cluster launch fails when all applications are installed and the default Amazon EBS root volume size is not changed. As a workaround, use the `aws emr create-cluster` command from the AWS CLI and specify a larger `--ebs-root-volume-size` parameter.

• Hive 2.3.0 sets `hive.compute.stats.using.query=true` by default. This causes queries to get data from existing statistics rather than directly from data, which could be confusing. For example, if you have a table with `hive.compute.stats.using.query=true` and upload new files to the table `LOCATION`, running a `SELECT COUNT(*)` query on the table returns the count from the statistics, rather than picking up the added rows.

  As a workaround, use the `ANALYZE TABLE` command to gather new statistics, or set `hive.compute.stats.using.query=false`. For more information, see Statistics in Hive in the Apache Hive documentation.

• **Spark**—When using Spark, there is a file handler leak issue with the apppusher daemon, which can appear for a long-running Spark job after several hours or days. To fix the issue, connect to the master node and type `sudo /etc/init.d/apppusher stop`. This stops that apppusher daemon, which Amazon EMR will restart automatically.

• **Application history**
  • Historical data for dead Spark executors is not available.
  • Application history is not available for clusters that use a security configuration to enable in-flight encryption.

### Release 5.7.0

The following release notes include information for the Amazon EMR 5.7.0 release. Changes are relative to the Amazon EMR 5.6.0 release.

**Release date:** July 13, 2017

### Upgrades

• Flink 1.3.0
• Phoenix 4.11.0
• Zeppelin 0.7.2

New Features

• Added the ability to specify a custom Amazon Linux AMI when you create a cluster. For more information, see Using a Custom AMI.

Changes, Enhancements, and Resolved Issues

• HBase
  • Added capability to configure HBase read-replica clusters. See Using a Read-Replica Cluster.
  • Multiple bug fixes and enhancements
• Presto—added ability to configure node.properties.
• YARN—added ability to configure container-log4j.properties
• Sqoop—backported SQOOP-2880, which introduces an argument that allows you to set the Sqoop temporary directory.

Release 5.6.0

The following release notes include information for the Amazon EMR 5.6.0 release. Changes are relative to the Amazon EMR 5.5.0 release.

Release date: June 5, 2017

Upgrades

• Flink 1.2.1
• HBase 1.3.1
• Mahout 0.13.0. This is the first version of Mahout to support Spark 2.x in Amazon EMR version 5.0 and later.
• Spark 2.1.1

Changes, Enhancements, and Resolved Issues

• Presto
  • Added the ability to enable SSL/TLS secured communication between Presto nodes by enabling in-transit encryption using a security configuration. For more information, see In-transit Data Encryption.
  • Backported Presto 7661, which adds the VERBOSE option to the EXPLAIN ANALYZE statement to report more detailed, low level statistics about a query plan.

Release 5.5.1

The following release notes include information for the Amazon EMR 5.5.1 release. Changes are relative to the Amazon EMR 5.5.0 release.
Changes, Enhancements, and Resolved Issues


Release 5.5.0

The following release notes include information for the Amazon EMR 5.5.0 release. Changes are relative to the Amazon EMR 5.4.0 release.

Release date: April 26, 2017

Upgrades

- Hue 3.12
- Presto 0.170
- Zeppelin 0.7.1
- ZooKeeper 3.4.10

Changes, Enhancements, and Resolved Issues

- **Spark**
  - Backported Spark Patch (SPARK-20115) Fix DAGScheduler to recompute all the lost shuffle blocks when external shuffle service is unavailable to version 2.1.0 of Spark, which is included in this release.

- **Flink**
  - Flink is now built with Scala 2.11. If you use the Scala API and libraries, we recommend that you use Scala 2.11 in your projects.
  - Addressed an issue where HADOOP_CONF_DIR and YARN_CONF_DIR defaults were not properly set, so start-scala-shell.sh failed to work. Also added the ability to set these values using env.hadoop.conf.dir and env.yarn.conf.dir in /etc/flink/conf/flink-conf.yaml or the flink-conf configuration classification.
  - Introduced a new EMR-specific command, flink-scala-shell as a wrapper for start-scala-shell.sh. We recommend using this command instead of start-scala-shell. The new command simplifies execution. For example, flink-scala-shell -n 2 starts a Flink Scala shell with a task parallelism of 2.
  - Introduced a new EMR-specific command, flink-yarn-session as a wrapper for yarn-session.sh. We recommend using this command instead of yarn-session. The new command simplifies execution. For example, flink-yarn-session -n 2 --d starts a long-running Flink session in a detached state with two task managers.
  - Addressed (FLINK-6125) Commons httpclient is not shaded anymore in Flink 1.2.

- **Presto**
  - Added support for LDAP authentication. Using LDAP with Presto on Amazon EMR requires that you enable HTTPS access for the Presto coordinator (http-server.https.enabled=true in config.properties). For configuration details, see LDAP Authentication in Presto documentation.
  - Added support for SHOW GRANTS.
Release 5.4.0

The following release notes include information for the Amazon EMR 5.4.0 release. Changes are relative to the Amazon EMR 5.3.0 release.

Release date: March 08, 2017

Upgrades

The following upgrades are available in this release:

- Upgraded to Flink 1.2.0
- Upgraded to Hbase 1.3.0
- Upgraded to Phoenix 4.9.0

Note

If you upgrade from an earlier version of Amazon EMR to Amazon EMR version 5.4.0 or later and use secondary indexing, upgrade local indexes as described in the Apache Phoenix documentation. Amazon EMR removes the required configurations from the hbase-site classification, but indexes need to be repopulated. Online and offline upgrade of indexes are supported. Online upgrades are the default, which means indexes are repopulated while initializing from Phoenix clients of version 4.8.0 or greater. To specify offline upgrades, set the phoenix.client.localIndexUpgrade configuration to false in the phoenix-site classification, and then SSH to the master node to run psql [zookeeper] -i.

- Upgraded to Presto 0.166
- Upgraded to Zeppelin 0.7.0

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-5.4.0:

- Added support for r4 instances. See Amazon EC2 Instance Types.

Release 5.3.0

The following release notes include information for the Amazon EMR 5.3.0 release. Changes are relative to the Amazon EMR 5.2.1 release.

Release date: January 26, 2017

Upgrades

The following upgrades are available in this release:

- Upgraded to Hive 2.1.1
Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-5.3.0:

- Added a patch to Hue that allows you to use the interpreters_shown_on_wheel setting to configure what interpreters to show first on the Notebook selection wheel, regardless of their ordering in the hue.ini file.
- Added the hive-parquet-logging configuration classification, which you can use to configure values in Hive's parquet-logging.properties file.

Release 5.2.2

The following release notes include information for the Amazon EMR 5.2.2 release. Changes are relative to the Amazon EMR 5.2.1 release.

Release date: May 2, 2017

Known Issues Resolved from the Previous Releases

- Backported SPARK-194459, which addresses an issue where reading from an ORC table with char/varchar columns can fail.

Release 5.2.1

The following release notes include information for the Amazon EMR 5.2.1 release. Changes are relative to the Amazon EMR 5.2.0 release.

Release date: December 29, 2016

Upgrades

The following upgrades are available in this release:

- Upgraded to Presto 0.157.1. For more information, see Presto Release Notes in the Presto documentation.
- Upgraded to Zookeeper 3.4.9. For more information, see ZooKeeper Release Notes in the Apache ZooKeeper documentation.

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-5.2.1:

- Added support for the Amazon EC2 m4.16xlarge instance type in Amazon EMR version 4.8.3 and later, excluding 5.0.0, 5.0.3, and 5.2.0.
Amazon EMR releases are now based on Amazon Linux 2016.09. For more information, see https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/.

- The location of Flink and YARN configuration paths are now set by default in /etc/default/flink that you don't need to set the environment variables FLINK_CONF_DIR and HADOOP_CONF_DIR when running the flink or yarn-session.sh driver scripts to launch Flink jobs.
- Added support for FlinkKinesisConsumer class.

### Known Issues Resolved from the Previous Releases

- Fixed an issue in Hadoop where the ReplicationMonitor thread could get stuck for a long time because of a race between replication and deletion of the same file in a large cluster.
- Fixed an issue where ControlledJob#toString failed with a null pointer exception (NPE) when job status was not successfully updated.

### Release 5.2.0

The following release notes include information for the Amazon EMR 5.2.0 release. Changes are relative to the Amazon EMR 5.1.0 release.

**Release date: November 21, 2016**

**Changes and enhancements**

The following changes and enhancements are available in this release:

- Added Amazon S3 storage mode for HBase.
- Enables you to specify an Amazon S3 location for the HBase rootdir. For more information, see HBase on Amazon S3.

**Upgrades**

The following upgrades are available in this release:

- Upgraded to Spark 2.0.2

### Known Issues Resolved from the Previous Releases

- Fixed an issue with /mnt being constrained to 2 TB on EBS-only instance types.
- Fixed an issue with instance-controller and logpusher logs being output to their corresponding .out files instead of to their normal log4j-configured .log files, which rotate hourly. The .out files don't rotate, so this would eventually fill up the /emr partition. This issue only affects hardware virtual machine (HVM) instance types.

### Release 5.1.0

The following release notes include information for the Amazon EMR 5.1.0 release. Changes are relative to the Amazon EMR 5.0.0 release.
Changes and enhancements

The following changes and enhancements are available in this release:

- Added support for Flink 1.1.3.
- Presto has been added as an option in the notebook section of Hue.

Upgrades

The following upgrades are available in this release:

- Upgraded to HBase 1.2.3
- Upgraded to Zeppelin 0.6.2

Known Issues Resolved from the Previous Releases

- Fixed an issue with Tez queries on Amazon S3 with ORC files did not perform as well as earlier Amazon EMR 4.x versions.

Release 5.0.3

The following release notes include information for the Amazon EMR 5.0.3 release. Changes are relative to the Amazon EMR 5.0.0 release.

Release date: October 24, 2016

Upgrades

The following upgrades are available in this release:

- Upgraded to Hadoop 2.7.3
- Upgraded to Presto 0.152.3, which includes support for the Presto web interface. You can access the Presto web interface on the Presto coordinator using port 8889. For more information about the Presto web interface, see Web Interface in the Presto documentation.
- Upgraded to Spark 2.0.1
- Amazon EMR releases are now based on Amazon Linux 2016.09. For more information, see https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/.

Release 5.0.0

Release date: July 27, 2016

Upgrades

The following upgrades are available in this release:
• Upgraded to Hive 2.1
• Upgraded to Presto 0.150
• Upgraded to Spark 2.0
• Upgraded to Hue 3.10.0
• Upgraded to Pig 0.16.0
• Upgraded to Tez 0.8.4
• Upgraded to Zeppelin 0.6.1

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-5.0.0 or greater:

• Amazon EMR supports the latest open-source versions of Hive (version 2.1) and Pig (version 0.16.0). If you have used Hive or Pig on Amazon EMR in the past, this may affect some use cases. For more information, see Hive and Pig.
• The default execution engine for Hive and Pig is now Tez. To change this, you would edit the appropriate values in the hive-site and pig-properties configuration classifications, respectively.
• An enhanced step debugging feature was added, which allows you to see the root cause of step failures if the service can determine the cause. For more information, see Enhanced Step Debugging in the Amazon EMR Management Guide.
• Applications that previously ended with "-Sandbox" no longer have that suffix. This may break your automation, for example, if you are using scripts to launch clusters with these applications. The following table shows application names in Amazon EMR 4.7.2 versus Amazon EMR 5.0.0.

<table>
<thead>
<tr>
<th>Application name changes</th>
<th>Amazon EMR 4.7.2</th>
<th>Amazon EMR 5.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oozie-Sandbox</td>
<td>Oozie</td>
<td></td>
</tr>
<tr>
<td>Presto-Sandbox</td>
<td>Presto</td>
<td></td>
</tr>
<tr>
<td>Sqoop-Sandbox</td>
<td>Sqoop</td>
<td></td>
</tr>
<tr>
<td>Zeppelin-Sandbox</td>
<td>Zeppelin</td>
<td></td>
</tr>
<tr>
<td>ZooKeeper-Sandbox</td>
<td>ZooKeeper</td>
<td></td>
</tr>
</tbody>
</table>

• Spark is now compiled for Scala 2.11.
• Java 8 is now the default JVM. All applications run using the Java 8 runtime. There are no changes to any application's byte code target. Most applications continue to target Java 7.
• Zeppelin now includes authentication features. For more information, see Zeppelin.
• Added support for security configurations, which allow you to create and apply encryption options more easily. For more information, see Data Encryption.

Release 4.9.3

The following release notes include information for the Amazon EMR 4.9.3 release. Changes are relative to the Amazon EMR 4.9.2 release.

Initial release date: January 22, 2018
Changes, Enhancements, and Resolved Issues


Release 4.9.2

The following release notes include information for the Amazon EMR 4.9.2 release. Changes are relative to the Amazon EMR 4.9.1 release.

Release date: July 13, 2017

Minor changes, bug fixes, and enhancements were made in this release.

Release 4.9.1

The following release notes include information for the Amazon EMR 4.9.1 release. Changes are relative to the Amazon EMR 4.8.4 release.

Release date: April 10, 2017

Known Issues Resolved from the Previous Releases

- Backports of HIVE-9976 and HIVE-10106
- Fixed an issue in YARN where a large number of nodes (greater than 2,000) and containers (greater than 5,000) would cause an out of memory error, for example: "Exception in thread 'main' java.lang.OutOfMemoryError".

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-4.9.1:

- Amazon EMR releases are now based on Amazon Linux 2017.03. For more information, see https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/.
- Removed Python 2.6 from the Amazon EMR base Linux image. You can install Python 2.6 manually if necessary.

Release 4.8.4

The following release notes include information for the Amazon EMR 4.8.4 release. Changes are relative to the Amazon EMR 4.8.3 release.

Release date: Feb 7, 2017

Minor changes, bug fixes, and enhancements were made in this release.
Release 4.8.3

The following release notes include information for the Amazon EMR 4.8.3 release. Changes are relative to the Amazon EMR 4.8.2 release.

Release date: December 29, 2016

Upgrades

The following upgrades are available in this release:

• Upgraded to Presto 0.157.1. For more information, see Presto Release Notes in the Presto documentation.
• Upgraded to Spark 1.6.3. For more information, see Spark Release Notes in the Apache Spark documentation.
• Upgraded to ZooKeeper 3.4.9. For more information, see ZooKeeper Release Notes in the Apache ZooKeeper documentation.

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-4.8.3:

• Added support for the Amazon EC2 m4.16xlarge instance type in Amazon EMR version 4.8.3 and later, excluding 5.0.0, 5.0.3, and 5.2.0.
• Amazon EMR releases are now based on Amazon Linux 2016.09. For more information, see https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/.

Known Issues Resolved from the Previous Releases

• Fixed an issue in Hadoop where the ReplicationMonitor thread could get stuck for a long time because of a race between replication and deletion of the same file in a large cluster.
• Fixed an issue where ControlledJob#toString failed with a null pointer exception (NPE) when job status was not successfully updated.

Release 4.8.2

The following release notes include information for the Amazon EMR 4.8.2 release. Changes are relative to the Amazon EMR 4.8.0 release.

Release date: October 24, 2016

Upgrades

The following upgrades are available in this release:

• Upgraded to Hadoop 2.7.3
• Upgraded to Presto 0.152.3, which includes support for the Presto web interface. You can access the Presto web interface on the Presto coordinator using port 8889. For more information about the Presto web interface, see Web Interface in the Presto documentation.
Amazon EMR releases are now based on Amazon Linux 2016.09. For more information, see https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/.

Release 4.8.0

Release date: September 7, 2016

Upgrades

The following upgrades are available in this release:

- Upgraded to HBase 1.2.2
- Upgraded to Presto-Sandbox 0.151
- Upgraded to Tez 0.8.4
- Upgraded to Zeppelin-Sandbox 0.6.1

Changes and Enhancements

The following are changes made to Amazon EMR releases for release label emr-4.8.0:

- Fixed an issue in YARN where the ApplicationMaster would attempt to clean up containers that no longer exist because their instances have been terminated.
- Corrected the hive-server2 URL for Hive2 actions in the Oozie examples.
- Added support for additional Presto catalogs.
- Backported patches: HIVE-8948, HIVE-12679, HIVE-13405, PHOENIX-3116, HADOOP-12689
- Added support for security configurations, which allow you to create and apply encryption options more easily. For more information, see Data Encryption.

Release 4.7.2

The following release notes include information for Amazon EMR 4.7.2.

Release date: July 15, 2016

Features

The following features are available in this release:

- Upgraded to Mahout 0.12.2
- Upgraded to Presto 0.148
- Upgraded to Spark 1.6.2
- You can now create an AWSCredentialsProvider for use with EMRFS using a URI as a parameter. For more information, see Create an AWSCredentialsProvider for EMRFS.
- EMRFS now allows users to configure a custom DynamoDB endpoint for their Consistent View metadata using the fs.s3.consistent.dynamodb.endpoint property in emrfs-site.xml.
- Added a script in /usr/bin called spark-example, which wraps /usr/lib/spark/spark/bin/run-example so you can run examples directly. For instance, to run the SparkPi example that comes
with the Spark distribution, you can run `spark-example SparkPi 100` from the command line or using `command-runner.jar` as a step in the API.

**Known Issues Resolved from Previous Releases**

- Fixed an issue where Oozie had the `spark-assembly.jar` was not in the correct location when Spark was also installed, which resulted in failure to launch Spark applications with Oozie.
- Fixed an issue with Spark Log4j-based logging in YARN containers.

**Release 4.7.1**

Release date: June 10, 2016

**Known Issues Resolved from Previous Releases**

- Fixed an issue that extended the startup time of clusters launched in a VPC with private subnets. The bug only impacted clusters launched with the Amazon EMR 4.7.0 release.
- Fixed an issue that improperly handled listing of files in Amazon EMR for clusters launched with the Amazon EMR 4.7.0 release.

**Release 4.7.0**

**Important**

Amazon EMR 4.7.0 is deprecated. Use Amazon EMR 4.7.1 or later instead.

Release date: June 2, 2016

**Features**

The following features are available in this release:

- Added Apache Phoenix 4.7.0
- Added Apache Tez 0.8.3
- Upgraded to HBase 1.2.1
- Upgraded to Mahout 0.12.0
- Upgraded to Presto 0.147
- Upgraded the AWS SDK for Java to 1.10.75
- The final flag was removed from the `mapreduce.cluster.local.dir` property in `mapred-site.xml` to allow users to run Pig in local mode.

**Amazon Redshift JDBC Drivers Available on Cluster**

Amazon Redshift JDBC drivers are now included at `/usr/share/aws/redshiftjdbc`. `/usr/share/aws/redshiftjdbc/RedshiftJDBC41.jar` is the JDBC 4.1-compatible Amazon Redshift driver and `/usr/share/aws/redshiftjdbc/RedshiftJDBC4.jar` is the JDBC 4.0-compatible Amazon Redshift driver. For more information, see Configure a JDBC Connection in the Amazon Redshift Cluster Management Guide.
Java 8

Except for Presto, OpenJDK 1.7 is the default JDK used for all applications. However, both OpenJDK 1.7 and 1.8 are installed. For information about how to set JAVA_HOME for applications, see Configuring Applications to Use Java 8.

Known Issues Resolved from Previous Releases

- Fixed a kernel issue that significantly affected performance on Throughput Optimized HDD (st1) EBS volumes for Amazon EMR in emr-4.6.0.
- Fixed an issue where a cluster would fail if any HDFS encryption zone were specified without choosing Hadoop as an application.
- Changed the default HDFS write policy from RoundRobin to AvailableSpaceVolumeChoosingPolicy. Some volumes were not properly utilized with the RoundRobin configuration, which resulted in failed core nodes and an unreliable HDFS.
- Fixed an issue with the EMRFS CLI, which would cause an exception when creating the default DynamoDB metadata table for consistent views.
- Fixed a deadlock issue in EMRFS that potentially occurred during multipart rename and copy operations.
- Fixed an issue with EMRFS that caused the CopyPart size default to be 5 MB. The default is now properly set at 128 MB.
- Fixed an issue with the Zeppelin upstart configuration that potentially prevented you from stopping the service.
- Fixed an issue with Spark and Zeppelin, which prevented you from using the s3a:// URI scheme because /usr/lib/hadoop/hadoop-aws.jar was not properly loaded in their respective classpath.
- Backported HUE-2484.
- Backported a commit from Hue 3.9.0 (no JIRA exists) to fix an issue with the HBase browser sample.
- Backported HIVE-9073.

Release 4.6.0

Release date: April 21, 2016

Features

The following features are available in this release:

- Added HBase 1.2.0
- Added Zookeeper-Sandbox 3.4.8
- Upgraded to Presto-Sandbox 0.143
- Amazon EMR releases are now based on Amazon Linux 2016.03.0. For more information, see https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/.

Issue Affecting Throughput Optimized HDD (st1) EBS Volume Types

An issue in the Linux kernel versions 4.2 and above significantly affects performance on Throughput Optimized HDD (st1) EBS volumes for EMR. This release (emr-4.6.0) uses kernel version 4.4.5 and hence
is impacted. Therefore, we recommend not using emr-4.6.0 if you want to use st1 EBS volumes. You can use emr-4.5.0 or prior Amazon EMR releases with st1 without impact. In addition, we provide the fix with future releases.

**Python Defaults**

Python 3.4 is now installed by default, but Python 2.7 remains the system default. You may configure Python 3.4 as the system default using either a bootstrap action; you can use the configuration API to set PYSPARK_PYTHON export to `/usr/bin/python3.4` in the `spark-env` classification to affect the Python version used by PySpark.

**Java 8**

Except for Presto, OpenJDK 1.7 is the default JDK used for all applications. However, both OpenJDK 1.7 and 1.8 are installed. For information about how to set JAVA_HOME for applications, see Configuring Applications to Use Java 8.

**Known Issues Resolved from Previous Releases**

- Fixed an issue where application provisioning would sometimes randomly fail due to a generated password.
- Previously, `mysqld` was installed on all nodes. Now, it is only installed on the master instance and only if the chosen application includes `mysql-server` as a component. Currently, the following applications include the `mysql-server` component: HCatalog, Hive, Hue, Presto-Sandbox, and Sqoop-Sandbox.
- Changed `yarn.scheduler.maximum-allocation-vcores` to 80 from the default of 32, which fixes an issue introduced in emr-4.4.0 that mainly occurs with Spark while using the `maximizeResourceAllocation` option in a cluster whose core instance type is one of a few large instance types that have the YARN vcores set higher than 32; namely c4.8xlarge, cc2.8xlarge, hs1.8xlarge, i2.8xlarge, m2.4xlarge, r3.8xlarge, d2.8xlarge, or m4.10xlarge were affected by this issue.
- `s3-dist-cp` now uses EMRFS for all Amazon S3 nominations and no longer stages to a temporary HDFS directory.
- Fixed an issue with exception handling for client-side encryption multipart uploads.
- Added an option to allow users to change the Amazon S3 storage class. By default this setting is `STANDARD`. The `emrfs-site` configuration classification setting is `fs.s3.storageClass` and the possible values are `STANDARD`, `STANDARD_IA`, and `REDUCED_REDUNDANCY`. For more information about storage classes, see Storage Classes in the Amazon Simple Storage Service Developer Guide.

**Release 4.5.0**

Release date: April 4, 2016

**Features**

The following features are available in this release:

- Upgraded to Spark 1.6.1
- Upgraded to Hadoop 2.7.2
- Upgraded to Presto 0.140
- Added AWS KMS support for Amazon S3 server-side encryption.
Known Issues Resolved from Previous Releases

- Fixed an issue where MySQL and Apache servers would not start after a node was rebooted.
- Fixed an issue where IMPORT did not work correctly with non-partitioned tables stored in Amazon S3
- Fixed an issue with Presto where it requires the staging directory to be /mnt/tmp rather than /tmp when writing to Hive tables.

Release 4.4.0

Release date: March 14, 2016

Features

The following features are available in this release:

- Added HCatalog 1.0.0
- Added Sqoop-Sandbox 1.4.6
- Upgraded to Presto 0.136
- Upgraded to Zeppelin 0.5.6
- Upgraded to Mahout 0.11.1
- Enabled dynamicResourceAllocation by default.
- Added a table of all configuration classifications for the release. For more information, see the Configuration Classifications table in Configuring Applications.

Known Issues Resolved from Previous Releases

- Fixed an issue where the maximizeResourceAllocation setting would not reserve enough memory for YARN ApplicationMaster daemons.
- Fixed an issue encountered with a custom DNS. If any entries in resolve.conf precede the custom entries provided, then the custom entries are not resolvable. This behavior was affected by clusters in a VPC where the default VPC name server is inserted as the top entry in resolve.conf.
- Fixed an issue where the default Python moved to version 2.7 and boto was not installed for that version.
- Fixed an issue where YARN containers and Spark applications would generate a unique Ganglia round robin database (rrd) file, which resulted in the first disk attached to the instance filling up. Because of this fix, YARN container level metrics have been disabled and Spark application level metrics have been disabled.
- Fixed an issue in log pusher where it would delete all empty log folders. The effect was that the Hive CLI was not able to log because log pusher was removing the empty user folder under /var/log/hive.
- Fixed an issue affecting Hive imports, which affected partitioning and resulted in an error during import.
- Fixed an issue where EMRFS and s3-dist-cp did not properly handle bucket names that contain periods.
- Changed a behavior in EMRFS so that in versioning-enabled buckets the _$folder$ marker file is not continuously created, which may contribute to improved performance for versioning-enabled buckets.
- Changed the behavior in EMRFS such that it does not use instruction files except for cases where client-side encryption is enabled. If you want to delete instruction files while using client-side encryption, you can set the emrfs-site.xml property, fs.s3.cse.cryptoStorageMode.deleteInstructionFiles.enabled, to true.
• Changed YARN log aggregation to retain logs at the aggregation destination for two days. The default
destination is your cluster's HDFS storage. If you want to change this duration, change the value of
yarn.log-aggregation.retain-seconds using the yarn-site configuration classification when
you create your cluster. As always, you can save your application logs to Amazon S3 using the log-uri
parameter when you create your cluster.

Patches Applied

The following patches from open source projects were included in this release:

• HIVE-9655
• HIVE-9183
• HADOOP-12810

Release 4.3.0

Release date: January 19, 2016

Features

The following features are available in this release:

• Upgraded to Hadoop 2.7.1
• Upgraded to Spark 1.6.0
• Upgraded Ganglia to 3.7.2
• Upgraded Presto to 0.130

Amazon EMR made some changes to spark.dynamicAllocation.enabled when it is set to true; it is
false by default. When set to true, this affects the defaults set by the maximizeResourceAllocation
setting:

• If spark.dynamicAllocation.enabled is set to true, spark.executor.instances is not set by
maximizeResourceAllocation.

• The spark.driver.memory setting is now configured based on the instance types in the cluster in a
similar way to how spark.executors.memory is set. However, because the Spark driver application
may run on either the master or one of the core instances (for example, in YARN client and cluster
modes, respectively), the spark.driver.memory setting is set based on the instance type of the
smaller instance type between these two instance groups.

• The spark.default.parallelism setting is now set at twice the number of CPU cores available for
YARN containers. In previous releases, this was half that value.

• The calculations for the memory overhead reserved for Spark YARN processes was adjusted to be
more accurate, resulting in a small increase in the total amount of memory available to Spark (that is,
spark.executor.memory).

Known Issues Resolved from the Previous Releases

• YARN log aggregation is now enabled by default.
• Fixed an issue where logs would not be pushed to a cluster's Amazon S3 logs bucket when YARN log
aggregation was enabled.
• YARN container sizes now have a new minimum of 32 across all node types.
• Fixed an issue with Ganglia that caused excessive disk I/O on the master node in large clusters.
• Fixed an issue that prevented applications logs from being pushed to Amazon S3 when a cluster is shutting down.
• Fixed an issue in EMRFS CLI that caused certain commands to fail.
• Fixed an issue with Zeppelin that prevented dependencies from being loaded in the underlying SparkContext.
• Fixed an issue that resulted from issuing a resize attempting to add instances.
• Fixed an issue in Hive where CREATE TABLE AS SELECT makes excessive list calls to Amazon S3.
• Fixed an issue where large clusters would not provision properly when Hue, Oozie, and Ganglia are installed.
• Fixed an issue in s3-dist-cp where it would return a zero exit code even if it failed with an error.

Patches Applied

The following patches from open source projects were included in this release:

• OOZIE-2402
• HIVE-12502
• HIVE-10631
• HIVE-12213
• HIVE-10559
• HIVE-12715
• HIVE-10685

Release 4.2.0

Release date: November 18, 2015

Features

The following features are available in this release:

• Added Ganglia support
• Upgraded to Spark 1.5.2
• Upgraded to Presto 0.125
• Upgraded Oozie to 4.2.0
• Upgraded Zeppelin to 0.5.5
• Upgraded the AWS SDK for Java to 1.10.27

Known Issues Resolved from the Previous Releases

• Fixed an issue with the EMRFS CLI where it did not use the default metadata table name.
• Fixed an issue encountered when using ORC-backed tables in Amazon S3.
• Fixed an issue encountered with a Python version mismatch in the Spark configuration.
• Fixed an issue when a YARN node status fails to report because of DNS issues for clusters in a VPC.
• Fixed an issue encountered when YARN decommissioned nodes, resulting in hanged applications or the inability to schedule new applications.
• Fixed an issue encountered when clusters terminated with status TIMED_OUT_STARTING.
• Fixed an issue encountered when including the EMRFS Scala dependency in other builds. The Scala dependency has been removed.
Configuring Applications

You can override the default configurations for applications by supplying a configuration object for applications when you create a cluster. The configuration object is referenced as a JSON file. Configuration objects consist of a classification, properties, and optional nested configurations. Properties are the settings you want to change in that file. You can specify multiple classifications for multiple applications in a single JSON object.

Example JSON for a list of configurations is provided below:

```json
[
  {
    "Classification": "core-site",
    "Properties": {
      "hadoop.security.groups.cache.secs": "250"
    }
  },
  {
    "Classification": "mapred-site",
    "Properties": {
      "mapreduce.map.sort.spill.percent": "0.90",
      "mapreduce.tasktracker.reduce.tasks.maximum": "5"
    }
  }
]
```

A classification typically maps to an application-specific configuration file. An exception to this is the deprecated bootstrap action configure-daemons, which is used to set environment parameters such as --namenode-heap-size. Now, options like this are subsumed into the hadoop-env and yarn-env classifications with their own nested export classifications. If any classification ends in "env", you should use the export sub-classification. Another exception is s3get, which was used to place a customer EncryptionMaterialsProvider object on each node in a cluster for use in client-side encryption. An option was added to the emrfs-site classification for this purpose.

An example of the hadoop-env classification is provided below:

```json
[
  {
    "Classification": "hadoop-env",
    "Properties": {
      "HADOOP_DATANODE_HEAPSIZE": "2048",
      "HADOOP_NAMENODE_OPTS": "-XX:GCTimeRatio=19"
    }
  }
]
```

An example of the yarn-env classification is provided below:

```json
[
  {
    "Classification": "yarn-env",
    "Properties": {
      "yarn.resourcemanager.resource Faythalizer": "true"
    }
  }
]
```
[{
  "Classification": "yarn-env",
  "Properties": {
    "YARN_RESOURCEMANAGER_OPTS": "-Xdebug -Xrunjdwp:transport=dt_socket"
  }
}]

The following settings do not belong to a configuration file but are used by Amazon EMR to potentially set multiple settings on your behalf.

**Amazon EMR-curated Settings**

<table>
<thead>
<tr>
<th>Application</th>
<th>Release label classification</th>
<th>Valid properties</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>spark</td>
<td>maximizeResourceAllocation</td>
<td>Configure executors to utilize the maximum resources of each node.</td>
</tr>
</tbody>
</table>

The following are all configuration classifications for this release:

**Configuration Classifications**

<table>
<thead>
<tr>
<th>Classifications</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity-scheduler</td>
<td>Change values in Hadoop's capacity-scheduler.xml file.</td>
</tr>
<tr>
<td>core-site</td>
<td>Change values in Hadoop's core-site.xml file.</td>
</tr>
<tr>
<td>emrfs-site</td>
<td>Change EMRFS settings.</td>
</tr>
<tr>
<td>flink-conf</td>
<td>Change flink-conf.yaml settings.</td>
</tr>
<tr>
<td>flink-log4j</td>
<td>Change Flink log4j.properties settings.</td>
</tr>
<tr>
<td>flink-log4j-yarn-session</td>
<td>Change Flink log4j-yarn-session.properties settings.</td>
</tr>
<tr>
<td>flink-log4j-cli</td>
<td>Change Flink log4j-cli.properties settings.</td>
</tr>
<tr>
<td>hadoop-env</td>
<td>Change values in the Hadoop environment for all Hadoop components.</td>
</tr>
<tr>
<td>hadoop-log4j</td>
<td>Change values in Hadoop's log4j.properties file.</td>
</tr>
<tr>
<td>hadoop-ssl-server</td>
<td>Change hadoop ssl server configuration.</td>
</tr>
<tr>
<td>Classifications</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hadoop-ssl-client</td>
<td>Change hadoop ssl client configuration</td>
</tr>
<tr>
<td>hbase</td>
<td>Amazon EMR-curated settings for Apache HBase.</td>
</tr>
<tr>
<td>hbase-env</td>
<td>Change values in HBase's environment.</td>
</tr>
<tr>
<td>hbase-log4j</td>
<td>Change values in HBase's hbase-log4j.properties file.</td>
</tr>
<tr>
<td>hbase-metrics</td>
<td>Change values in HBase's hadoop-metrics2-hbase.properties file.</td>
</tr>
<tr>
<td>hbase-policy</td>
<td>Change values in HBase's hbase-policy.xml file.</td>
</tr>
<tr>
<td>hbase-site</td>
<td>Change values in HBase's hbase-site.xml file.</td>
</tr>
<tr>
<td>hdfs-encryption-zones</td>
<td>Configure HDFS encryption zones.</td>
</tr>
<tr>
<td>hdfs-site</td>
<td>Change values in HDFS's hdfs-site.xml file.</td>
</tr>
<tr>
<td>hcatalog-env</td>
<td>Change values in HCatalog's environment.</td>
</tr>
<tr>
<td>hcatalog-server-jndi</td>
<td>Change values in HCatalog's jndi.properties.</td>
</tr>
<tr>
<td>hcatalog-server-proto-hive-site</td>
<td>Change values in HCatalog's proto-hive-site.xml.</td>
</tr>
<tr>
<td>hcatalog-webhcat-env</td>
<td>Change values in HCatalog WebHCat's environment.</td>
</tr>
<tr>
<td>hcatalog-webhcat-log4j2</td>
<td>Change values in HCatalog WebHCat's log4j2.properties.</td>
</tr>
<tr>
<td>hcatalog-webhcat-site</td>
<td>Change values in HCatalog WebHCat's webhcat-site.xml file.</td>
</tr>
<tr>
<td>hive-beeline-log4j2</td>
<td>Change values in Hive's beeline-log4j2.properties file.</td>
</tr>
<tr>
<td>hive-parquet-logging</td>
<td>Change values in Hive's parquet-logging.properties file.</td>
</tr>
<tr>
<td>hive-env</td>
<td>Change values in the Hive environment.</td>
</tr>
<tr>
<td>hive-log4j2</td>
<td>Change values in Hive's hive-log4j2.properties file.</td>
</tr>
<tr>
<td>hive-site</td>
<td>Change values in Hive's hive-site.xml file</td>
</tr>
<tr>
<td>hiveserver2-site</td>
<td>Change values in Hive Server2's hiveserver2-site.xml file</td>
</tr>
<tr>
<td>hue-ini</td>
<td>Change values in Hue's ini file</td>
</tr>
<tr>
<td>httpfs-env</td>
<td>Change values in the HTTPFS environment.</td>
</tr>
<tr>
<td>Classifications</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>httpfs-site</td>
<td>Change values in Hadoop's httpfs-site.xml file.</td>
</tr>
<tr>
<td>hadoop-kms-env</td>
<td>Change values in the Hadoop KMS environment.</td>
</tr>
<tr>
<td>hadoop-kms-log4j</td>
<td>Change values in Hadoop's kms-log4j.properties file.</td>
</tr>
<tr>
<td>hadoop-kms-site</td>
<td>Change values in Hadoop's kms-site.xml file.</td>
</tr>
<tr>
<td>livy-conf</td>
<td>Change values in Livy's livy.conf file.</td>
</tr>
<tr>
<td>livy-env</td>
<td>Change values in the Livy environment.</td>
</tr>
<tr>
<td>livy-log4j</td>
<td>Change Livy log4j.properties settings.</td>
</tr>
<tr>
<td>mapred-env</td>
<td>Change values in the MapReduce application's environment.</td>
</tr>
<tr>
<td>mapred-site</td>
<td>Change values in the MapReduce application's mapred-site.xml file.</td>
</tr>
<tr>
<td>oozie-env</td>
<td>Change values in Oozie's environment.</td>
</tr>
<tr>
<td>oozie-log4j</td>
<td>Change values in Oozie's oozie-log4j.properties file.</td>
</tr>
<tr>
<td>oozie-site</td>
<td>Change values in Oozie's oozie-site.xml file.</td>
</tr>
<tr>
<td>phoenix-hbase-metrics</td>
<td>Change values in Phoenix's hadoop-metrics2-hbase.properties file.</td>
</tr>
<tr>
<td>phoenix-hbase-site</td>
<td>Change values in Phoenix's hbase-site.xml file.</td>
</tr>
<tr>
<td>phoenix-log4j</td>
<td>Change values in Phoenix's log4j.properties file.</td>
</tr>
<tr>
<td>phoenix-metrics</td>
<td>Change values in Phoenix's hadoop-metrics2-phoenix.properties file.</td>
</tr>
<tr>
<td>pig-properties</td>
<td>Change values in Pig's pig.properties file.</td>
</tr>
<tr>
<td>pig-log4j</td>
<td>Change values in Pig's log4j.properties file.</td>
</tr>
<tr>
<td>presto-log</td>
<td>Change values in Presto's log.properties file.</td>
</tr>
<tr>
<td>presto-config</td>
<td>Change values in Presto's config.properties file.</td>
</tr>
<tr>
<td>presto-env</td>
<td>Change values in Presto's presto-env.sh file.</td>
</tr>
<tr>
<td>presto-node</td>
<td>Change values in Presto's node.properties file.</td>
</tr>
<tr>
<td>presto-connector-blackhole</td>
<td>Change values in Presto's blackhole.properties file.</td>
</tr>
<tr>
<td>presto-connector-cassandra</td>
<td>Change values in Presto's cassandra.properties file.</td>
</tr>
<tr>
<td>presto-connector-hive</td>
<td>Change values in Presto's hive.properties file.</td>
</tr>
<tr>
<td>presto-connector-jmx</td>
<td>Change values in Presto's jmx.properties file.</td>
</tr>
</tbody>
</table>
### Classifications

<table>
<thead>
<tr>
<th>Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>presto-connector-kafka</td>
<td>Change values in Presto's kafka.properties file.</td>
</tr>
<tr>
<td>presto-connector-localfile</td>
<td>Change values in Presto's localfile.properties file.</td>
</tr>
<tr>
<td>presto-connector-mongodb</td>
<td>Change values in Presto's mongodb.properties file.</td>
</tr>
<tr>
<td>presto-connector-mysql</td>
<td>Change values in Presto's mysql.properties file.</td>
</tr>
<tr>
<td>presto-connector-postgresql</td>
<td>Change values in Presto's postgresql.properties file.</td>
</tr>
<tr>
<td>presto-connector-raptor</td>
<td>Change values in Presto's raptor.properties file.</td>
</tr>
<tr>
<td>presto-connector-redis</td>
<td>Change values in Presto's redis.properties file.</td>
</tr>
<tr>
<td>presto-connector-tpch</td>
<td>Change values in Presto's tpch.properties file.</td>
</tr>
<tr>
<td>spark</td>
<td>Amazon EMR-curated settings for Apache Spark.</td>
</tr>
<tr>
<td>spark-defaults</td>
<td>Change values in Spark's spark-defaults.conf file.</td>
</tr>
<tr>
<td>spark-env</td>
<td>Change values in the Spark environment.</td>
</tr>
<tr>
<td>spark-hive-site</td>
<td>Change values in Spark's hive-site.xml file</td>
</tr>
<tr>
<td>spark-log4j</td>
<td>Change values in Spark's log4j.properties file.</td>
</tr>
<tr>
<td>spark-metrics</td>
<td>Change values in Spark's metrics.properties file.</td>
</tr>
<tr>
<td>sqoop-env</td>
<td>Change values in Sqoop's environment.</td>
</tr>
<tr>
<td>sqoop-oraoop-site</td>
<td>Change values in Sqoop OraOop's oraoop-site.xml file.</td>
</tr>
<tr>
<td>sqoop-site</td>
<td>Change values in Sqoop's sqoop-site.xml file</td>
</tr>
<tr>
<td>tez-site</td>
<td>Change values in Tez's tez-site.xml file.</td>
</tr>
<tr>
<td>yarn-env</td>
<td>Change values in the YARN environment.</td>
</tr>
<tr>
<td>yarn-site</td>
<td>Change values in YARN's yarn-site.xml file.</td>
</tr>
<tr>
<td>zeppelin-env</td>
<td>Change values in the Zeppelin environment.</td>
</tr>
<tr>
<td>zookeeper-config</td>
<td>Change values in ZooKeeper's zoo.cfg file.</td>
</tr>
<tr>
<td>zookeeper-log4j</td>
<td>Change values in ZooKeeper's log4j.properties file.</td>
</tr>
</tbody>
</table>

### Example Supplying a Configuration in the Console

To supply a configuration, you navigate to the Create cluster page and choose **Edit software settings**. You can then enter the configuration directly (in JSON or using shorthand syntax demonstrated in shadow text) in the console or provide an Amazon S3 URI for a file with a JSON Configurations object.
Example Supplying a Configuration Using the CLI

You can provide a configuration to `create-cluster` by supplying a path to a JSON file stored locally or in Amazon S3. The following example assumes you are using default roles for Amazon EMR and the roles have been created. If you need to create the roles, run `aws emr create-default-roles` first.

```
aws emr create-cluster --use-default-roles --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 2 --applications Name=Hive --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

If your configuration is in your local directory, you can use the following:

```
aws emr create-cluster --use-default-roles --release-label emr-5.11.0 --applications Name=Hive --instance-type m3.xlarge --instance-count 3 --configurations file:///./configurations.json
```

Example Supplying a Configuration Using the Java SDK

The following program excerpt shows how to supply a configuration using the AWS SDK for Java:

```
Application hive = new Application().withName("Hive");
Map<String,String> hiveProperties = new HashMap<String,String>();
hiveProperties.put("hive.join.emit.interval","1000");
hiveProperties.put("hive.merge.mapfiles","true");
Configuration myHiveConfig = new Configuration() .withClassification("hive-site") .withProperties(hiveProperties);
RunJobFlowRequest request = new RunJobFlowRequest() .withName("Create cluster with ReleaseLabel") .withReleaseLabel("emr-5.11.0") .withApplications(hive) .withConfigurations(myHiveConfig) .withServiceRole("EMR_DefaultRole") .withJobFlowRole("EMR_EC2_DefaultRole") .withInstances(new JobFlowInstancesConfig() .withEc2KeyName("myKey") .withInstanceCount(1) .withKeepJobFlowAliveWhenNoSteps(true) .withMasterInstanceType("m3.xlarge") .withSlaveInstanceType("m3.xlarge"));
```

Configuring Applications to Use a Specific Java Virtual Machine

Java 8 is the default Java virtual machine (JVM) for cluster instances created using Amazon EMR version 4.9.2 or later. To override this JVM setting—for example, to use Java 8 with a cluster created using Amazon EMR version 4.8.0—you can set `JAVA_HOME` for an application by supplying the setting to its environment classification, `application-env`. For Hadoop and Hive, this would look like:

```
[...]
```
For Spark, if you are writing a driver for submission in cluster mode, the driver will use Java 7 but setting the environment can ensure that the executors use Java 8. To do this, we recommend setting both Hadoop and Spark classifications:

```json
{
  "Classification": "hadoop-env",
  "Configurations": [
    {
      "Classification": "export",
      "Configurations": [],
      "Properties": {
        "JAVA_HOME": "/usr/lib/jvm/java-1.8.0"
      }
    },
    "Properties": {}
  ],
  "Properties": {}
}
{
  "Classification": "spark-env",
  "Configurations": [
    {
      "Classification": "export",
      "Configurations": [],
      "Properties": {
        "JAVA_HOME": "/usr/lib/jvm/java-1.8.0"
      }
    },
    "Properties": {}
  ],
  "Properties": {}
}
```

Service ports

The following are YARN and HDFS service ports. These settings reflect Hadoop defaults. Other application services are hosted at default ports unless otherwise documented. Please see the application’s project documentation for further information.

### Port Settings for YARN and HDFS

<table>
<thead>
<tr>
<th>Setting</th>
<th>Hostname/Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs.default.name</td>
<td>default (hdfs://emrDeterminedIP:8020)</td>
</tr>
</tbody>
</table>
### Application users

Applications will run processes as their own user. For example, Hive JVMs will run as user `hive`, MapReduce JVMs will run as `mapred`, and so on. The following process status demonstrates this:

```
USER     PID   %CPU  %MEM  VSZ   RSS  TTY      STAT   START   TIME     COMMAND
hive     6452   0.2   0.7 853684 218520  ?       Sl   16:32   0:13       /usr/lib/jvm/java-
hive     6557   0.2   0.6 849508 202396  ?       Sl   16:32   0:09       /usr/lib/jvm/java-
```

### Note

The term `emrDeterminedIP` is an IP address that is generated by the Amazon EMR control plane. In the newer version, this convention has been eliminated except for the `yarn.resourcemanager.hostname` and `fs.default.name` settings.
hbase  6716  0.1  1.0 1755516 336600 ?      Sl   Jun21   2:20 /usr/lib/jvm/java-openjdk/bin/java -Dproc_master -XX:OnOutOfMemoryError=kill -9 %p -Xmx1024m -ea -XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode -Dhbase.log.dir=/var/
hbase  6871  0.0  0.7 1672196 237648 ?      Sl   Jun21   0:46 /usr/lib/jvm/java-openjdk/bin/java -Dproc_thrift -XX:OnOutOfMemoryError=kill -9 %p -Xmx1024m -ea -XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode -Dhbase.log.dir=/var/
hdfs  7491  0.4  1.0 1719476 309820 ?      Sl   16:32   0:22 /usr/lib/jvm/java-openjdk/bin/java -Dproc_namenode -Xmx1000m -Dhadoop.log.dir=/var/log/hadoop-hdfs -Dhadoop.log.file=hadoop-hdfs-namenode-ip-10-71-203-213.log -Dhadoop
yarn  8524  0.1  0.6 1626164 211300 ?      Sl   16:33   0:05 /usr/lib/jvm/java-openjdk/bin/java -Dproc_proxyserver -Xmx1000m -Dhadoop.log.dir=/var/log/hadoop-yarn -Dyarn.log.dir=/var/log/hadoop-yarn -Dhadoop.log.file=yarn-yarn-yarn  8646  1.0  1.2 1876916 385308 ?      Sl   16:33   0:46 /usr/lib/jvm/java-openjdk/bin/java -Dproc_resourcemanager -Xmx1000m -Dhadoop.log.dir=/var/log/hadoop-yarn -Dyarn.log.dir=/var/log/hadoop-yarn -Dhadoop.log.file=yarn-yarn-yarn  9265  0.2  0.8 1666628 260484 ?      Sl   16:33   0:12 /usr/lib/jvm/java-openjdk/bin/java -Dproc_historyserver -Xmx1000m -Dhadoop.log.dir=/usr/lib/hadoop/logs -Dhadoop.log.file=hadoop.log -Dhadoop.home.dir=/usr/lib/hadoop
Apache Hadoop

Apache Hadoop is an open-source Java software framework that supports massive data processing across a cluster of instances. It can run on a single instance, or thousands of instances. Hadoop uses a programming model called MapReduce to distribute processing across multiple instances. It also implements a distributed file system called HDFS that stores data across multiple instances. Hadoop monitors the health of instances in the cluster, and can recover from the failure of one or more nodes. In this way, Hadoop provides increased processing and storage capacity, as well as high availability.

For more information, see http://hadoop.apache.org

Hadoop Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Create or Run a Hadoop Application

Topics
- Build Binaries Using Amazon EMR (p. 39)
- Run a Script in a Cluster (p. 41)
- Process Data with Streaming (p. 42)
- Process Data with a Custom JAR (p. 45)

Build Binaries Using Amazon EMR

You can use Amazon EMR as a build environment to compile programs for use in your cluster. Programs that you use with Amazon EMR must be compiled on a system running the same version of Linux used
by Amazon EMR. For a 32-bit version, you should have compiled on a 32-bit machine or with 32-bit cross compilation options turned on. For a 64-bit version, you need to have compiled on a 64-bit machine or with 64-bit cross compilation options turned. For more information about EC2 instance versions, go to http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-ec2-instances.html. Supported programming languages include C++, Cython, and C#.

The following table outlines the steps involved to build and test your application using Amazon EMR.

**Process for Building a Module**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connect to the master node of your cluster.</td>
</tr>
<tr>
<td>2</td>
<td>Copy source files to the master node.</td>
</tr>
<tr>
<td>3</td>
<td>Build binaries with any necessary optimizations.</td>
</tr>
<tr>
<td>4</td>
<td>Copy binaries from the master node to Amazon S3.</td>
</tr>
</tbody>
</table>

The details for each of these steps are covered in the sections that follow.

**To connect to the master node of the cluster**

- Follow these instructions to connect to the master node: Connect to the Master Node Using SSH in the Amazon EMR Management Guide.

**To copy source files to the master node**

1. Put your source files in an Amazon S3 bucket. To learn how to create buckets and how to move data into Amazon S3, see the Amazon Simple Storage Service Getting Started Guide.
2. Create a folder on your Hadoop cluster for your source files by entering a command similar to the following:

   ```bash
   mkdir SourceFiles
   ```
3. Copy your source files from Amazon S3 to the master node by typing a command similar to the following:

   ```bash
   hadoop fs -get s3://mybucket/SourceFiles SourceFiles
   ```

**Build binaries with any necessary optimizations**

How you build your binaries depends on many factors. Follow the instructions for your specific build tools to setup and configure your environment. You can use Hadoop system specification commands to obtain cluster information to determine how to install your build environment.

**To identify system specifications**

- Use the following commands to verify the architecture you are using to build your binaries.
  
  a. To view the version of Debian, enter the following command:

  ```bash
  master# cat /etc/issue
  ```
  
  The output looks similar to the following.
b. To view the public DNS name and processor size, enter the following command:

```
master$ uname -a
```

The output looks similar to the following.

```
Linux domU-12-31-39-17-29-39.compute-1.internal 2.6.21.7-2.fc8xen #1 SMP Fri Feb 15 12:34:28 EST 2008 x86_64 GNU/Linux
```

c. To view the processor speed, enter the following command:

```
master$ cat /proc/cpuinfo
```

The output looks similar to the following.

```
processor : 0
vendor_id : GenuineIntel
model name : Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
flags : fpu tsc mar pae mce cx8 apic mca cmov pat pse36 clflush dts acpi mmx fxsr
sse sse2 ss ht tm syscall nx lm constant_tsc pni monitor ds_cpl vmx est tm2 ssse3
x86 xtrr cda lahf_lm
```

Once your binaries are built, you can copy the files to Amazon S3.

**To copy binaries from the master node to Amazon S3**

- Type the following command to copy the binaries to your Amazon S3 bucket:

```
hadoop fs -put BinaryFiles s3://mybucket/BinaryDestination
```

---

Run a Script in a Cluster

Amazon EMR enables you to run a script at any time during step processing in your cluster. You specify a step that runs a script either when you create your cluster or you can add a step if your cluster is in the WAITING state. For more information about adding steps, see Submit Work to a Cluster.

To run a script before step processing begins, use a bootstrap action. For more information about bootstrap actions, see (Optional) Create Bootstrap Actions to Install Additional Software.

**Submitting a Custom JAR Step Using the AWS CLI**

**Note**

You can now use command-runner.jar in many cases instead of script-runner.jar. command-runner.jar does not need to have a full path for the JAR. For more information, see Command Runner (p. 199).

This section describes how to add a step to run a script. The script-runner.jar takes arguments to the path to a script and any additional arguments for the script. The JAR file runs the script with the passed arguments.
Important  
script-runner.jar is located at s3://region.elasticmapreduce/libs/script-runner/script-runner.jar where region is the region in which your EMR cluster resides.

The cluster containing a step that runs a script looks similar to the following examples.

To add a step to run a script using the AWS CLI

- To run a script using the AWS CLI, type the following command, replace myKey with the name of your EC2 key pair and replace mybucket with your S3 bucket. This cluster runs the script my_script.sh on the master node when the step is processed.

```bash
aws emr create-cluster --name "Test cluster" --release-label emr-5.11.0 --applications Name=Hive Name=Pig --use-default-roles --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 --steps Type=CUSTOM_JAR,Name=CustomJAR,ActionOnFailure=CONTINUE,Jar=s3://region.elasticmapreduce/libs/script-runner/script-runner.jar,Args=["s3://mybucket/script-path/my_script.sh"]
```

When you specify the instance count without using the --instance-groups parameter, a single master node is launched, and the remaining instances are launched as core nodes. All nodes use the instance type specified in the command.

Note  
If you have not previously created the default Amazon EMR service role and EC2 instance profile, type aws emr create-default-roles to create them before typing the create-cluster subcommand.

For more information on using Amazon EMR commands in the AWS CLI, see http://docs.aws.amazon.com/cli/latest/reference/emr.

Process Data with Streaming

Hadoop Streaming is a utility that comes with Hadoop that enables you to develop MapReduce executables in languages other than Java. Streaming is implemented in the form of a JAR file, so you can run it from the Amazon EMR API or command line just like a standard JAR file.

This section describes how to use Streaming with Amazon EMR.

Note  
Apache Hadoop Streaming is an independent tool. As such, all of its functions and parameters are not described here. For more information about Hadoop Streaming, go to http://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html.

Using the Hadoop Streaming Utility

This section describes how to use Hadoop's Streaming utility.

Hadoop Process

|   | Write your mapper and reducer executable in the programming language of your choice. Follow the directions in Hadoop's documentation to write your streaming executables. The programs should read their input from standard input and output data through standard output. By default, each line of input/output represents a record and the first tab on each line is used as a separator between the key and value. |
2 Test your executables locally and upload them to Amazon S3.

3 Use the Amazon EMR command line interface or Amazon EMR console to run your application.

Each mapper script launches as a separate process in the cluster. Each reducer executable turns the output of the mapper executable into the data output by the job flow.

The `input`, `output`, `mapper`, and `reducer` parameters are required by most Streaming applications. The following table describes these and other, optional parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-input</td>
<td>Location on Amazon S3 of the input data. Type: String Default: None Constraint: URI. If no protocol is specified then it uses the cluster's default file system.</td>
<td>Yes</td>
</tr>
<tr>
<td>-output</td>
<td>Location on Amazon S3 where Amazon EMR uploads the processed data. Type: String Default: None Constraint: URI Default: If a location is not specified, Amazon EMR uploads the data to the location specified by <code>input</code>.</td>
<td>Yes</td>
</tr>
<tr>
<td>-mapper</td>
<td>Name of the mapper executable. Type: String Default: None</td>
<td>Yes</td>
</tr>
<tr>
<td>-reducer</td>
<td>Name of the reducer executable. Type: String Default: None</td>
<td>Yes</td>
</tr>
<tr>
<td>-cacheFile</td>
<td>An Amazon S3 location containing files for Hadoop to copy into your local working directory (primarily to improve performance). Type: String Default: None Constraints: [URI]#[symlink name to create in working directory]</td>
<td>No</td>
</tr>
<tr>
<td>-cacheArchive</td>
<td>JAR file to extract into the working directory Type: String Default: None</td>
<td>No</td>
</tr>
</tbody>
</table>
### Submit a Streaming Step

This section covers the basics of submitting a Streaming step to a cluster. A Streaming application reads input from standard input and then runs a script or executable (called a mapper) against each input. The result from each of the inputs is saved locally, typically on a Hadoop Distributed File System (HDFS) partition. After all the input is processed by the mapper, a second script or executable (called a reducer) processes the mapper results. The results from the reducer are sent to standard output. You can chain together a series of Streaming steps, where the output of one step becomes the input of another step.

The mapper and the reducer can each be referenced as a file or you can supply a Java class. You can implement the mapper and reducer in any of the supported languages, including Ruby, Perl, Python, PHP, or Bash.

### Submit a Streaming Step Using the Console

This example describes how to use the Amazon EMR console to submit a Streaming step to a running cluster.

**To submit a Streaming step**

1. Open the Amazon EMR console at `https://console.aws.amazon.com/elasticmapreduce/`.
2. In the **Cluster List**, select the name of your cluster.
3. Scroll to the **Steps** section and expand it, then choose **Add step**.
4. In the **Add Step** dialog box:
   - For **Step type**, choose **Streaming program**.
Process Data with a Custom JAR

A custom JAR runs a compiled Java program that you upload to Amazon S3. Compile the program against the version of Hadoop you want to launch and submit a CUSTOM_JAR step to your Amazon EMR cluster. For more information about compiling a JAR file, see Build Binaries Using Amazon EMR (p. 39).
Submit a Custom JAR Step

This section covers the basics of submitting a custom JAR step in Amazon EMR. Submitting a custom JAR step enables you to write a script to process your data using the Java programming language.

Submit a Custom JAR Step Using the Console

This example describes how to use the Amazon EMR console to submit a custom JAR step to a running cluster.

To submit a custom JAR step using the console
1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. In the Cluster List, select the name of your cluster.
3. Scroll to the Steps section and expand it, then choose Add step.
4. In the Add Step dialog:
   - For Step type, choose Custom JAR.
   - For Name, accept the default name (Custom JAR) or type a new name.
   - For JAR S3 location, type or browse to the location of your JAR file. The value must be in the form s3://BucketName/path/JARfile.
   - For Arguments, type any required arguments as space-separated strings or leave the field blank.
   - For Action on failure, accept the default option (Continue).
5. Choose Add. The step appears in the console with a status of Pending.
6. The status of the step changes from Pending to Running to Completed as the step runs. To update the status, choose the Refresh icon above the Actions column.

Launching a cluster and submitting a custom JAR step using the AWS CLI

To launch a cluster and submit a custom JAR step using the AWS CLI

To launch a cluster and submit a custom JAR step using the AWS CLI, type the create-cluster subcommand with the --steps parameter.

- To launch a cluster and submit a custom JAR step, type the following command, replace myKey with the name of your EC2 key pair, and replace mybucket with your bucket name.

```bash
aws emr create-cluster --name "Test cluster" --release-label emr-5.11.0 --applications Name=Hue Name=Hive Name=Pig --use-default-roles --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 --steps Type=CUSTOM_JAR,Name="Custom JAR Step",ActionOnFailure=CONTINUE,Jar=pathtojarfile,Args=\"pathtoinputdata\",\"pathtooutputbucket\",\"arg1\",\"arg2\""
```

Note

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

When you specify the instance count without using the --instance-groups parameter, a single master node is launched, and the remaining instances are launched as core nodes. All nodes use the instance type specified in the command.
Note
If you have not previously created the default Amazon EMR service role and EC2 instance profile, type `aws emr create-default-roles` to create them before typing the `create-cluster` subcommand.

For more information on using Amazon EMR commands in the AWS CLI, see `http://docs.aws.amazon.com/cli/latest/reference/emr`.

Third-party dependencies
Sometimes it may be necessary to include in the MapReduce classpath JARs for use with your program. You have two options for doing this:

- Include the `--libjars s3://URI_to_JAR` in the step options for the procedure in Launching a cluster and submitting a custom JAR step using the AWS CLI (p. 46).
- Launch the cluster with a modified `mapreduce.application.classpath` setting in `mapred-site.xml` using the `mapred-site` configuration classification. To create the cluster with the step using AWS CLI, this would look like the following:

```
aws emr create-cluster --release-label "" --applications Name=Hue Name=Hive Name=Pig --use-default-roles --instance-type m3.xlarge --instance-count 2 --ec2-attributes KeyName=myKey --steps Type=CUSTOM_JAR,Name="Custom JAR Step",ActionOnFailure=CONTINUE,Jar=pathtojarfile,Args=["pathtoinputdata","pathtooutputbucket","arg1"] --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

`myConfig.json`:

```
[
  {
    "Classification": "mapred-site",
    "Properties": {
      "mapreduce.application.classpath": "path1,path2"
    }
  }
]
```

The comma-separated list of paths should be appended to the classpath for each task's JVM.

Configure Hadoop

The following sections give default configuration settings for Hadoop daemons, tasks, and HDFS.

Topics
- Hadoop Daemon Settings (p. 47)
- HDFS Configuration (p. 61)
- Task Configuration (p. 66)

Hadoop Daemon Settings

The following tables list the default configuration settings for each EC2 instance type in clusters launched with Amazon EMR.
### m1.medium

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_HEAPSIZE</td>
<td>384</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>192</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>256</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>256</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>384</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>192</td>
</tr>
</tbody>
</table>

### m1.large

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_HEAPSIZE</td>
<td>768</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>384</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>512</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>512</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>768</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### m1.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>512</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>768</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>2304</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### m2.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER HEAPSIZE</td>
<td>1024</td>
</tr>
</tbody>
</table>
## Hadoop Daemon Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3072</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### m2.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6144</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### m2.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>12288</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### m3.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>1740</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>757</td>
</tr>
</tbody>
</table>
### m3.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_heapsize</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_ProxyServer_heapsize</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_Nodemanager_heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>Hadoop_Job_historyserver_heapsize</td>
<td>2703</td>
</tr>
<tr>
<td>Hadoop_Namenode_heapsize</td>
<td>3276</td>
</tr>
<tr>
<td>Hadoop_Datanode_heapsize</td>
<td>1064</td>
</tr>
</tbody>
</table>

### m4.large

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_heapsize</td>
<td>2252</td>
</tr>
<tr>
<td>YARN_ProxyServer_heapsize</td>
<td>2252</td>
</tr>
<tr>
<td>YARN_Nodemanager_heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>Hadoop_Job_historyserver_heapsize</td>
<td>2252</td>
</tr>
<tr>
<td>Hadoop_Namenode_heapsize</td>
<td>1024</td>
</tr>
<tr>
<td>Hadoop_Datanode_heapsize</td>
<td>614</td>
</tr>
</tbody>
</table>

### m4.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_heapsize</td>
<td>2416</td>
</tr>
<tr>
<td>YARN_ProxyServer_heapsize</td>
<td>2416</td>
</tr>
<tr>
<td>YARN_Nodemanager_heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>Hadoop_Job_historyserver_heapsize</td>
<td>2416</td>
</tr>
<tr>
<td>Hadoop_Namenode_heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>Hadoop_Datanode_heapsize</td>
<td>778</td>
</tr>
</tbody>
</table>

### m4.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESourcemanager_heapsize</td>
<td>2744</td>
</tr>
<tr>
<td>YARN_ProxyServer_heapsize</td>
<td>2744</td>
</tr>
<tr>
<td>YARN_Nodemanager_heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>Hadoop_Job_historyserver_heapsize</td>
<td>2744</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3481</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1105</td>
</tr>
</tbody>
</table>

### m4.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>3399</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3399</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3399</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6758</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1761</td>
</tr>
</tbody>
</table>

### m4.10xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>5365</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>5365</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>5365</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>16588</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>3727</td>
</tr>
</tbody>
</table>

### m4.16xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>7331</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>7331</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7331</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>26419</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
</tbody>
</table>
### c1.medium

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>192</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>96</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>128</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>128</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>192</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>96</td>
</tr>
</tbody>
</table>

### c1.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>768</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>384</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>512</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>512</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>768</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### c3.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>972</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>588</td>
</tr>
</tbody>
</table>

### c3.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>1740</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>757</td>
</tr>
</tbody>
</table>

**c3.4xlarge**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3276</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1064</td>
</tr>
</tbody>
</table>

**c3.8xlarge**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>3317</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3317</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3317</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6348</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1679</td>
</tr>
</tbody>
</table>

**c4.large**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>1152</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1152</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1152</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1152</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>576</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>
### c4.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2124</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>972</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>588</td>
</tr>
</tbody>
</table>

### c4.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2396</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>1740</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>757</td>
</tr>
</tbody>
</table>

### c4.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2703</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3276</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1064</td>
</tr>
</tbody>
</table>

### c4.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>3317</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3317</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3317</td>
</tr>
</tbody>
</table>
## Hadoop Daemon Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6348</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1679</td>
</tr>
</tbody>
</table>

### cc2.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>12288</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### cg1.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3840</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

### cr1.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>25190</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
</tbody>
</table>
## Hadoop Daemon Settings

### d2.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARNPROXYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARNNODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3328</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1075</td>
</tr>
</tbody>
</table>

### d2.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARNPROXYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARNNODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6451</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1699</td>
</tr>
</tbody>
</table>

### d2.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARNPROXYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARNNODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>12697</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>2949</td>
</tr>
</tbody>
</table>

### d2.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>7089</td>
</tr>
<tr>
<td>YARNPROXYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARNNODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
</tbody>
</table>
### Parameter Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HADOOP Namenode Heapsize</td>
<td>25190</td>
</tr>
<tr>
<td>HADOOP Datanode Heapsize</td>
<td>4096</td>
</tr>
</tbody>
</table>

#### hi1.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN ResourceManager Heapsize</td>
<td>3328</td>
</tr>
<tr>
<td>YARN ProxyServer Heapsize</td>
<td>3328</td>
</tr>
<tr>
<td>YARN NodeManager Heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP Job HistoryServer Heapsize</td>
<td>3328</td>
</tr>
<tr>
<td>HADOOP Namenode Heapsize</td>
<td>6400</td>
</tr>
<tr>
<td>HADOOP Datanode Heapsize</td>
<td>1689</td>
</tr>
</tbody>
</table>

#### hs1.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN ResourceManager Heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>YARN ProxyServer Heapsize</td>
<td>1024</td>
</tr>
<tr>
<td>YARN NodeManager Heapsize</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP Job HistoryServer Heapsize</td>
<td>1536</td>
</tr>
<tr>
<td>HADOOP Namenode Heapsize</td>
<td>12288</td>
</tr>
<tr>
<td>HADOOP Datanode Heapsize</td>
<td>384</td>
</tr>
</tbody>
</table>

#### i2.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN ResourceManager Heapsize</td>
<td>2713</td>
</tr>
<tr>
<td>YARN ProxyServer Heapsize</td>
<td>2713</td>
</tr>
<tr>
<td>YARN NodeManager Heapsize</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP Job HistoryServer Heapsize</td>
<td>2713</td>
</tr>
<tr>
<td>HADOOP Namenode Heapsize</td>
<td>3328</td>
</tr>
<tr>
<td>HADOOP Datanode Heapsize</td>
<td>1075</td>
</tr>
</tbody>
</table>
### i2.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6451</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1699</td>
</tr>
</tbody>
</table>

### i2.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>12697</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>2949</td>
</tr>
</tbody>
</table>

### i2.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>25190</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
</tbody>
</table>

### g2.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>1536</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>1024</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>2304</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>384</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3328</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1075</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6451</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1699</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESOURCEMANAGER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>12697</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>2949</td>
</tr>
</tbody>
</table>
### r3.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESCOURSEMANAGER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>25190</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
</tbody>
</table>

**Note**

R4 instances are available only in version 5.4.0 and later.

### r4.xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESCOURSEMANAGER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>2713</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>3328</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1075</td>
</tr>
</tbody>
</table>

### r4.2xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESCOURSEMANAGER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>3338</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>6451</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>1699</td>
</tr>
</tbody>
</table>

### r4.4xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN_RESCOURSEMANAGER_HEAPSIZE</td>
<td>4587</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>4587</td>
</tr>
</tbody>
</table>
### HDFS Configuration

The following table describes the default Hadoop Distributed File System (HDFS) parameters and their settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfs.block.size</td>
<td>134217728 (128 MB)</td>
</tr>
<tr>
<td>dfs.replication</td>
<td>1 for clusters &lt; four nodes</td>
</tr>
</tbody>
</table>

The following table describes the default HDFS parameters and their settings for different instance types:

#### r4.8xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>7086</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>25190</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
</tbody>
</table>

#### r4.16xlarge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARNRESOURCEMANAGER_HEAPSIZE</td>
<td>12083</td>
</tr>
<tr>
<td>YARN_PROXYSERVER_HEAPSIZE</td>
<td>12083</td>
</tr>
<tr>
<td>YARN_NODEMANAGER_HEAPSIZE</td>
<td>2048</td>
</tr>
<tr>
<td>HADOOP_JOB_HISTORYSERVER_HEAPSIZE</td>
<td>12083</td>
</tr>
<tr>
<td>HADOOP_NAMENODE_HEAPSIZE</td>
<td>50176</td>
</tr>
<tr>
<td>HADOOP_DATANODE_HEAPSIZE</td>
<td>4096</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>easy to restart in case of data loss. You can change the setting to 1, 2, or 3 as your needs dictate. Amazon EMR automatically calculates the replication factor based on cluster size. To overwrite the default value, use the hdfs-site classification.</td>
</tr>
</tbody>
</table>

### Transparent Encryption in HDFS on Amazon EMR

**Note**
This feature is only available in Amazon EMR release version 4.1.0 and later.

Transparent encryption is implemented through the use of HDFS encryption zones, which are HDFS paths that you define. Each encryption zone has its own key, which is stored in the key server specified by the hdfs-site configuration.

Amazon EMR uses the Hadoop KMS by default; however, you can use another KMS that implements the KeyProvider API operation. Each file in an HDFS encryption zone has its own unique data encryption key, which is encrypted by the encryption zone key. HDFS data is encrypted end-to-end (at-rest and in-transit) when data is written to an encryption zone because encryption and decryption activities only occur in the client.

**Note**
You cannot move files between encryptions zones or from an encryption zone to unencrypted paths.

The NameNode and HDFS client interact with the Hadoop KMS (or an alternate KMS you configured) through the KeyProvider API operation. The KMS is responsible for storing encryption keys in the backing keystore. Also, Amazon EMR includes the JCE unlimited strength policy, so you can create keys at a desired length.

For more information, see **Transparent Encryption in HDFS** in the Hadoop documentation.

**Note**
In Amazon EMR, KMS over HTTPS is not enabled by default with Hadoop KMS. For more information about how to enable KMS over HTTPS, see the Hadoop KMS documentation.

### Configuring HDFS Transparent Encryption

You can configure transparent encryption by creating keys and adding encryption zones. You can do this in several ways:

- Using the Amazon EMR configuration API operation when you create a cluster
- Using a Hadoop JAR step with command-runner.jar
- Logging in to the master node of the Hadoop cluster and using the hadoop key and hdfs crypto command line clients
- Using the REST APIs for Hadoop KMS and HDFS

For more information about the REST APIs, see the respective documentation for Hadoop KMS and HDFS.

**To create encryption zones and their keys at cluster creation using the CLI**

The hdfs-encryption-zones classification in the configuration API operation allows you to specify a key name and an encryption zone when you create a cluster. Amazon EMR creates this key in Hadoop KMS on your cluster and configures the encryption zone.

- Create a cluster with the following command:
aws emr create-cluster --release-label \emr-5.11.0--instance-type m3.xlarge --instance-count 2 \--applications Name=App1 Name=App2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json

**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

myConfig.json:

```
[
  {
    "Classification": "hdfs-encryption-zones",
    "Properties": {
      "/myHDFSPath1": "path1_key",
      "/myHDFSPath2": "path2_key"
    }
  }
]
```

To create encryption zones and their keys manually on the master node

1. Launch your cluster using an Amazon EMR release greater than 4.1.0.
2. Connect to the master node of the cluster using SSH.
3. Create a key within Hadoop KMS:

   ```
   $ hadoop key create path2_key
   path2_key has been successfully created with options Options{cipher='AES/CTR/NoPadding', bitLength=256, description='null', attributes=null}. KMSClientProvider[http://ip-x-x-x-x.ec2.internal:16000/kms/v1/] has been updated.
   ```

**Important**

Hadoop KMS requires your key names to be lowercase. If you use a key that has uppercase characters, then your cluster will fail during launch.

4. Create the encryption zone path in HDFS:

   ```
   $ hadoop fs -mkdir /myHDFSPath2
   ```

5. Make the HDFS path an encryption zone using the key that you created:

   ```
   $ hdfs crypto -createZone -keyName path2_key -path /myHDFSPath2
   Added encryption zone /myHDFSPath2
   ```

To create encryption zones and their keys manually using the AWS CLI

- Add steps to create the KMS keys and encryption zones manually with the following command:

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF --steps Type=CUSTOM_JAR,Name="Create First Hadoop KMS Key",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args=[/bin/bash,-c,\"hadoop key create path1_key\""] Type=CUSTOM_JAR,Name="Create First Hadoop HDFS Path",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args=[/bin/bash,-c,\"hadoop fs -mkdir /myHDFSPath1\""]
```
HDFS Configuration

Type=CUSTOM_JAR,Name="Create First Encryption Zone",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args="/bin/bash,-c,""hdfs crypto -createZone -keyName path1_key -path /myHDFSPath1"
Type=CUSTOM_JAR,Name="Create Second Hadoop KMS Key",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args="/bin/bash,-c,""hadoop key create path2_key"
Type=CUSTOM_JAR,Name="Create Second Hadoop HDFS Path",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args="/bin/bash,-c,""hadoop fs -mkdir /myHDFSPath2"
Type=CUSTOM_JAR,Name="Create Second Encryption Zone",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args="/bin/bash,-c,""hdfs crypto -createZone -keyName path2_key -path /myHDFSPath2"

Note
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

Considerations for HDFS Transparent Encryption

A best practice is to create an encryption zone for each application where they may write files. Also, you can encrypt all of HDFS by using the hdfs-encryption-zones classification in the configuration API and specify the root path (/) as the encryption zone.

Hadoop Key Management Server

Hadoop KMS is a key management server that provides the ability to implement cryptographic services for Hadoop clusters, and can serve as the key vendor for Transparent Encryption in HDFS on Amazon EMR (p. 62). Hadoop KMS in Amazon EMR is installed and enabled by default when you select the Hadoop application while launching an EMR cluster. The Hadoop KMS does not store the keys itself except in the case of temporary caching. Hadoop KMS acts as a proxy between the key provider and the client trustee to a backing keystore—it is not a keystore. The default keystore that is created for Hadoop KMS is the Java Cryptography Extension KeyStore (JCEKS). The JCE unlimited strength policy is also included, so you can create keys with the desired length. Hadoop KMS also supports a range of ACLs that control access to keys and key operations independently of other client applications such as HDFS. The default key length in Amazon EMR is 256 bit.

To configure Hadoop KMS, use the hadoop-kms-site classification to change settings. To configure ACLs, you use the classification kms-acls.

For more information, see the Hadoop KMS documentation. Hadoop KMS is used in Hadoop HDFS transparent encryption. To learn more about HDFS transparent encryption, see the HDFS Transparent Encryption topic in the Apache Hadoop documentation.

Note
In Amazon EMR, KMS over HTTPS is not enabled by default with Hadoop KMS. To learn how to enable KMS over HTTPS, see the Hadoop KMS documentation.

Important
Hadoop KMS requires your key names to be lowercase. If you use a key that has uppercase characters, then your cluster will fail during launch.

Configuring Hadoop KMS in Amazon EMR

Important
The Hadoop KMS port is changed in Amazon EMR release 4.6 or later. kms-http-port is now 9700 and kms-admin-port is 9701.

You can configure Hadoop KMS at cluster creation time using the configuration API for Amazon EMR releases. The following are the configuration object classifications available for Hadoop KMS:
### Hadoop KMS Configuration Classifications

<table>
<thead>
<tr>
<th>Classification</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>hadoop-kms-site</td>
<td>kms-site.xml</td>
</tr>
<tr>
<td>hadoop-kms-acls</td>
<td>kms-acls.xml</td>
</tr>
<tr>
<td>hadoop-kms-env</td>
<td>kms-env.sh</td>
</tr>
<tr>
<td>hadoop-kms-log4j</td>
<td>kms-log4j.properties</td>
</tr>
</tbody>
</table>

### To set Hadoop KMS ACLs using the CLI

- Create a cluster with Hadoop KMS with ACLs using the following command:

```bash
aws emr create-cluster --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

`myConfig.json`:

```json
[
  {
    "Classification": "hadoop-kms-acls",
    "Properties": {
      "hadoop.kms.blacklist.CREATE": "hdfs,foo,myBannedUser",
      "hadoop.kms.acl.ROLLOVER": "myAllowedUser"
    }
  }
]
```

### To disable Hadoop KMS cache using the CLI

- Create a cluster with Hadoop KMS hadoop.kms.cache.enable set to false, using the following command:

```bash
aws emr create-cluster --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

`myConfig.json`:

```json
[
  {
    "Classification": "hadoop-kms-site",
    "Properties": {
```

65
To set environment variables in the `kms-env.sh` script using the CLI

- Change settings in `kms-env.sh` via the `hadoop-kms-env` configuration. Create a cluster with Hadoop KMS using the following command:

  ```bash
  aws emr create-cluster --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 2 --applications Name=App1 Name=App2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
  ```

  **Note**
  
  Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

  `myConfig.json`:

  ```json
  [  
      {  
          "Classification": "hadoop-kms-env",  
          "Properties": {  
          },  
          "Configurations": [  
              {  
                  "Classification": "export",  
                  "Properties": {  
                      "JAVA_LIBRARY_PATH": "/path/to/files",  
                      "KMS_SSL_KEYSTORE_FILE": "/non/Default/Path/.keystore",  
                      "KMS_SSL_KEYSTORE_PASS": "myPass"  
                  },  
                  "Configurations": [  
                  ]  
              }  
          ]  
      }  
  ]
  ```

  For information about configuring Hadoop KMS, see the [Hadoop KMS documentation](https://docs.aws.amazon.com/hadoop-kms/latest/ug/).

**Task Configuration**

**Topics**
- Task JVM Memory Settings (p. 66)

There are a number of configuration variables for tuning the performance of your MapReduce jobs. This section describes some of the important task-related settings.

**Task JVM Memory Settings**

Hadoop 2 uses two parameters to configure memory for map and reduce: `mapreduce.map.java.opts` and `mapreduce.reduce.java.opts`, respectively. These replace the single configuration option from previous Hadoop versions: `mapreduce.map.java.opts`.

```json
"hadoop.kms.cache.enable": "false"
```
The defaults for these settings per instance type are shown in the following tables. HBase is available when using Amazon EMR release version 4.6.0 and later. HBase settings change when HBase is installed and those are also provided along with the initial defaults.

### m1.medium

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx512m</td>
<td>-Xmx512m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx768m</td>
<td>-Xmx768m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>768</td>
<td></td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>2048</td>
<td></td>
</tr>
</tbody>
</table>

### m1.large

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx512m</td>
<td>-Xmx512m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1024m</td>
<td>-Xmx1024m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>768</td>
<td></td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>1536</td>
<td></td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>1536</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>5120</td>
<td></td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>5120</td>
<td></td>
</tr>
</tbody>
</table>

### m1.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx512m</td>
<td>-Xmx512m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1536m</td>
<td>-Xmx1536m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>768</td>
<td></td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>

67
### Configuration Option

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
</tbody>
</table>

#### m2.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx864m</td>
<td>-Xmx864m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1536m</td>
<td>-Xmx1536m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>14336</td>
<td>7168</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>14336</td>
<td>7168</td>
</tr>
</tbody>
</table>

#### m2.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1280m</td>
<td>-Xmx1280m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>30720</td>
<td>15360</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>30720</td>
<td>15360</td>
</tr>
</tbody>
</table>

#### m2.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1280m</td>
<td>-Xmx1280m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
</tbody>
</table>
### Task Configuration

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>61440</td>
<td>30720</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>61440</td>
<td>30720</td>
</tr>
</tbody>
</table>

#### m3.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
</tbody>
</table>

#### m3.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
</tbody>
</table>

#### m4.large

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2458m</td>
<td>-Xmx2458m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx4916m</td>
<td>-Xmx4916m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6144</td>
<td>6144</td>
</tr>
</tbody>
</table>
### Task Configuration

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6144</td>
<td>6144</td>
</tr>
<tr>
<td>yarn.schedualer.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedualer.maximum-allocation-mb</td>
<td>6144</td>
<td>3072</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>6144</td>
<td>3072</td>
</tr>
</tbody>
</table>

#### m4.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1229m</td>
<td>-Xmx1229m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2548m</td>
<td>-Xmx2458m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>yarn.schedualer.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedualer.maximum-allocation-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
</tbody>
</table>

#### m4.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1229m</td>
<td>-Xmx1229m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2458m</td>
<td>-Xmx2458m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>yarn.schedualer.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedualer.maximum-allocation-mb</td>
<td>24576</td>
<td>12288</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>24576</td>
<td>12288</td>
</tr>
</tbody>
</table>

#### m4.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1434m</td>
<td>-Xmx1434m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2868m</td>
<td>-Xmx2868m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1792</td>
<td>1792</td>
</tr>
</tbody>
</table>
## Task Configuration

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3584</td>
<td>3584</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3584</td>
<td>3584</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>57344</td>
<td>28672</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>57344</td>
<td>28672</td>
</tr>
</tbody>
</table>

### m4.10xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1557m</td>
<td>-Xmx1557m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx3114m</td>
<td>-Xmx3114m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1946</td>
<td>1946</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3892</td>
<td>3892</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3892</td>
<td>3892</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>155648</td>
<td>124544</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>155648</td>
<td>124544</td>
</tr>
</tbody>
</table>

### m4.16xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1587m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx3114m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1984</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3968</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3968</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>253952</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>253952</td>
</tr>
</tbody>
</table>

### c1.medium

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.sort.mb</td>
<td>100</td>
</tr>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx288m</td>
</tr>
<tr>
<td>Configuration Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx288m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>512</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>512</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td></td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>512</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>1024</td>
</tr>
</tbody>
</table>

### c1.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.sort.mb</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx864m</td>
<td>-Xmx864m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1536m</td>
<td>-Xmx1536m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>5120</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>5120</td>
<td>2560</td>
</tr>
</tbody>
</table>

### c3.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1126m</td>
<td>-Xmx1126m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2252m</td>
<td>-Xmx2252m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1408</td>
<td>1408</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>5632</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>5632</td>
<td>2816</td>
</tr>
</tbody>
</table>
### c3.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
</tbody>
</table>

### c3.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
</tbody>
</table>

### c3.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1331m</td>
<td>-Xmx1331m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2662m</td>
<td>-Xmx2662m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1664</td>
<td>1664</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>3328</td>
<td>3328</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>3328</td>
<td>3328</td>
</tr>
<tr>
<td>yarn.schedulers.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulers.maximum-allocation-mb</td>
<td>53248</td>
<td>26624</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>53248</td>
<td>26624</td>
</tr>
</tbody>
</table>
### c4.large

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value (Without HBase)</th>
<th>Default Value (With HBase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx717m</td>
<td>-Xmx717m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1434m</td>
<td>-Xmx1434m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>896</td>
<td>896</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>1792</td>
<td>1792</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>1792</td>
<td>1792</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>1792</td>
<td>896</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>1792</td>
<td>896</td>
</tr>
</tbody>
</table>

### c4.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value (Without HBase)</th>
<th>Default Value (With HBase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1126m</td>
<td>-Xmx1126m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2252m</td>
<td>-Xmx2252m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1408</td>
<td>1408</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>5632</td>
<td>2816</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>5632</td>
<td>2816</td>
</tr>
</tbody>
</table>

### c4.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value (Without HBase)</th>
<th>Default Value (With HBase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>11520</td>
<td>5760</td>
</tr>
</tbody>
</table>
### c4.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1152m</td>
<td>-Xmx1152m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1440</td>
<td>1440</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2880</td>
<td>2880</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23040</td>
<td>11520</td>
</tr>
</tbody>
</table>

### c4.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1183m</td>
<td>-Xmx1183m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2366m</td>
<td>-Xmx2366m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1479</td>
<td>1479</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2958</td>
<td>2958</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2958</td>
<td>2958</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>53248</td>
<td>26624</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>53248</td>
<td>26624</td>
</tr>
</tbody>
</table>

### cg1.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1280m</td>
<td>-Xmx1280m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>20480</td>
<td>10240</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>20480</td>
<td>10240</td>
</tr>
</tbody>
</table>
### cc2.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1280m</td>
<td>-Xmx1280m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.schedulerm Minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulerm Maximum-allocation-mb</td>
<td>56320</td>
<td>28160</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>56320</td>
<td>28160</td>
</tr>
</tbody>
</table>

### cr1.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx6042m</td>
<td>-Xmx6042m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx12084m</td>
<td>-Xmx12084m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>15104</td>
<td>15104</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>15104</td>
<td>15104</td>
</tr>
<tr>
<td>yarn.schedulerm Minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulerm Maximum-allocation-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
</tbody>
</table>

### d2.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2342m</td>
<td>-Xmx2342m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx4684m</td>
<td>-Xmx4684m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>2928</td>
<td>2928</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.schedulerm Minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.schedulerm Maximum-allocation-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
</tbody>
</table>
### d2.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2714m</td>
<td>-Xmx2714m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5428m</td>
<td>-Xmx5428m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3392</td>
<td>3392</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
</tbody>
</table>

### d2.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2918m</td>
<td>-Xmx2918m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5836m</td>
<td>-Xmx5836m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3648</td>
<td>3648</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
</tbody>
</table>

### d2.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2417m</td>
<td>-Xmx2417m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx4384m</td>
<td>-Xmx4384m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3021</td>
<td>3021</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6042</td>
<td>6042</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6042</td>
<td>6042</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>241664</td>
<td>211470</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>241664</td>
<td>211470</td>
</tr>
</tbody>
</table>
### g2.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx512m</td>
<td>-Xmx512m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx1536m</td>
<td>-Xmx1536m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>768</td>
<td>768</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>12288</td>
<td>6144</td>
</tr>
</tbody>
</table>

### hi1.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2688m</td>
<td>-Xmx2688m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5376m</td>
<td>-Xmx5376m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3360</td>
<td>3360</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6720</td>
<td>6720</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6720</td>
<td>6720</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>53760</td>
<td>26880</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>53760</td>
<td>26880</td>
</tr>
</tbody>
</table>

### hs1.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx1280m</td>
<td>-Xmx1280m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx2304m</td>
<td>-Xmx2304m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>8192</td>
<td>28160</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>56320</td>
<td>28160</td>
</tr>
</tbody>
</table>
### i2.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2342m</td>
<td>-Xmx2342m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx4684m</td>
<td>-Xmx4684m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>2928</td>
<td>2928</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
</tbody>
</table>

### i2.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2714m</td>
<td>-Xmx2714m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5428m</td>
<td>-Xmx5428m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3392</td>
<td>3392</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
</tbody>
</table>

### i2.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2918m</td>
<td>-Xmx2918m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5836m</td>
<td>-Xmx5836m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3648</td>
<td>3648</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
</tbody>
</table>
### i2.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx3021m</td>
<td>-Xmx3021m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx6042m</td>
<td>-Xmx6042m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3776</td>
<td>3776</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
</tbody>
</table>

### r3.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2342m</td>
<td>-Xmx2342m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx4684m</td>
<td>-Xmx4684m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>2982</td>
<td>2982</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
</tbody>
</table>

### r3.2xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2714m</td>
<td>-Xmx2714m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5428m</td>
<td>-Xmx5428m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3392</td>
<td>3392</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
</tbody>
</table>
## Task Configuration

### r3.4xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2918m</td>
<td>-Xmx2918m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx5836m</td>
<td>-Xmx5836m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3648</td>
<td>3648</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
</tbody>
</table>

### r3.8xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx3021m</td>
<td>-Xmx3021m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx6042m</td>
<td>-Xmx6042m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>3776</td>
<td>3776</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>241664</td>
<td>211456</td>
</tr>
</tbody>
</table>

### Note

R4 instances are available only in version 5.4.0 and later.

### r4.xlarge

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx4685m</td>
<td>-Xmx2342m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx9370m</td>
<td>-Xmx4684m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>5856</td>
<td>5856</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>11712</td>
<td>11712</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>11712</td>
<td>11712</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
<tr>
<td>Configuration Option</td>
<td>Default Value</td>
<td>With HBase Installed</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>23424</td>
<td>11712</td>
</tr>
</tbody>
</table>

**r4.2xlarge**

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx2714m</td>
<td>-Xmx2714m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx10854m</td>
<td>-Xmx10854m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>6784</td>
<td>6784</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>13568</td>
<td>13568</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>13568</td>
<td>13568</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>54272</td>
<td>27136</td>
</tr>
</tbody>
</table>

**r4.4xlarge**

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx5837m</td>
<td>-Xmx5837m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx11674m</td>
<td>-Xmx11674m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>7296</td>
<td>7296</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>14592</td>
<td>14592</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>14592</td>
<td>14592</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>yarn.scheduler.maximum-allocation-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
<tr>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>116736</td>
<td>87552</td>
</tr>
</tbody>
</table>

**r4.8xlarge**

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Default Value</th>
<th>With HBase Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce.map.java.opts</td>
<td>-Xmx6042m</td>
<td>-Xmx6042m</td>
</tr>
<tr>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx12084m</td>
<td>-Xmx12084m</td>
</tr>
<tr>
<td>mapreduce.map.memory.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>mapreduce.reduce.memory.mb</td>
<td>15104</td>
<td>15104</td>
</tr>
<tr>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>7552</td>
<td>7552</td>
</tr>
<tr>
<td>yarn.scheduler.minimum-allocation-mb</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>
Configuration Option | Default Value | With HBase Installed
--- | --- | ---
yarn.schedulerm.maximum-allocation-mb | 241664 | 211456
yarn.nodemanager.resource.memory-mb | 241664 | 211456

**r4.16xlarge**

Configuration Option | Default Value | With HBase Installed
--- | --- | ---
mapreduce.map.java.opts | -Xmx6144m | -Xmx6144m
mapreduce.reduce.java.opts | -Xmx12288m | -Xmx12288m
mapreduce.map.memory.mb | 7680 | 7680
mapreduce.reduce.memory.mb | 15360 | 15360
yarn.app.mapreduce.am.resource.mb | 15360 | 15360
yarn.schedulerm.minimum-allocation-mb | 32 | 32
yarn.schedulerm.maximum-allocation-mb | 491520 | 460800
yarn.nodemanager.resource.memory-mb | 491520 | 460800

Use the `mapred.job.reuse.jvm.num.tasks` option to configure the JVM reuse settings.

If you have not previously created the default EMR service role and EC2 instance profile, type `aws emr create-default-roles` to create them before typing the `create-cluster` subcommand.

**Note**
Amazon EMR sets the value of `mapred.job.reuse.jvm.num.tasks` to 20, but you can override it. A value of -1 means infinite reuse within a single job, and 1 means do not reuse tasks.

For more information, see [Amazon EMR commands in the AWS CLI](https://docs.aws.amazon.com/emr/latest/UG/GettingStarted.html).
Ganglia

The Ganglia open source project is a scalable, distributed system designed to monitor clusters and grids while minimizing the impact on their performance. When you enable Ganglia on your cluster, you can generate reports and view the performance of the cluster as a whole, as well as inspect the performance of individual node instances. Ganglia is also configured to ingest and visualize Hadoop and Spark metrics. For more information about the Ganglia open-source project, go to http://ganglia.info/.

When you view the Ganglia web UI in a browser, you see an overview of the cluster’s performance, with graphs detailing the load, memory usage, CPU utilization, and network traffic of the cluster. Below the cluster statistics are graphs for each individual server in the cluster.

Ganglia Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Add Ganglia to a Cluster

To add Ganglia to a cluster using the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster.
3. In Software configuration, choose either All Applications, Core Hadoop, or Spark.
4. Proceed with creating the cluster with configurations as appropriate.

To add Ganglia to a cluster using the AWS CLI

In the AWS CLI, you can add Ganglia to a cluster by using create-cluster subcommand with the --applications parameter. If you specify only Ganglia using the --applications parameter, Ganglia is the only application installed.
View Ganglia Metrics

Ganglia provides a web-based user interface that you can use to view the metrics Ganglia collects. When you run Ganglia on Amazon EMR, the web interface runs on the master node and can be viewed using port forwarding, also known as creating an SSH tunnel. For more information about viewing web interfaces on Amazon EMR, see View Web Interfaces Hosted on Amazon EMR Clusters in the Amazon EMR Management Guide.

To view the Ganglia web interface

1. Use SSH to tunnel into the master node and create a secure connection. For information about how to create an SSH tunnel to the master node, see Option 2, Part 1: Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding in the Amazon EMR Management Guide.

2. Install a web browser with a proxy tool, such as the FoxyProxy plug-in for Firefox, to create a SOCKS proxy for domains of the type *ec2*.amazonaws.com*. For more information, see Option 2, Part 2: Configure Proxy Settings to View Websites Hosted on the Master Node in the Amazon EMR Management Guide.

3. With the proxy set and the SSH connection open, you can view the Ganglia UI by opening a browser window with http://master-public-dns-name/ganglia/, where master-public-dns-name is the public DNS address of the master server in the EMR cluster.
Hadoop and Spark Metrics in Ganglia

Ganglia reports Hadoop metrics for each instance. The various types of metrics are prefixed by category: distributed file system (dfs.*), Java virtual machine (jvm.*), MapReduce (mapred.*), and remote procedure calls (rpc.*).

Ganglia metrics for Spark generally have prefixes for YARN application ID and Spark DAGScheduler. So prefixes follow this form:

- DAGScheduler.*
- applicationxxxxxxxx_xxxx.driver.*
- applicationxxxxxxxx_xxxx.executor.*
Apache HBase

This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

HBase is an open source, non-relational, distributed database developed as part of Apache Software Foundation’s Hadoop project. HBase runs on top of Hadoop Distributed File System (HDFS) to provide non-relational database capabilities for the Hadoop ecosystem. HBase works seamlessly with Hadoop, sharing its file system and serving as a direct input and output to the MapReduce framework and execution engine. HBase also integrates with Apache Hive, enabling SQL-like queries over HBase tables, joins with Hive-based tables, and support for Java Database Connectivity (JDBC). For more information about HBase, see Apache HBase and HBase documentation on the Apache website. For an example of how to use HBase with Hive, see the AWS Big Data Blog post Combine NoSQL and Massively Parallel Analytics Using Apache HBase and Apache Hive on Amazon EMR.

Amazon EMR offers options to integrate with Amazon Simple Storage Service (Amazon S3) for data persistence and disaster recovery.

- **HBase on Amazon S3**—With Amazon EMR version 5.2.0 and later, you can use HBase on Amazon S3 to store a cluster’s HBase root directory and metadata directly to Amazon S3. You can subsequently start a new cluster, pointing it to the root directory location in Amazon S3. Only one cluster at a time can use the HBase location in Amazon S3, with the exception of a read-replica cluster. For more information, see HBase on Amazon S3 (Amazon S3 Storage Mode) (p. 89).

- **HBase read-replicas**—Amazon EMR version 5.7.0 and later with HBase on Amazon S3 supports read-replica clusters. A read-replica cluster provides read-only access to a primary cluster’s store files and metadata for read-only operations. For more information, see Using a Read-Replica Cluster (p. 90).

- **HBase Snapshots**—As an alternative to HBase on Amazon S3, with EMR version 4.0 and later you can create snapshots of your HBase data directly to Amazon S3 and then recover data using the snapshots. For more information, see Using HBase Snapshots (p. 95).

HBase Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label `emr-5.11.1`). This component information applies to both `emr-5.11.1` and `emr-5.11.0`.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBase 1.3.1</td>
<td>emr-5.11.0</td>
<td>emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpsfs-server, hadoop-kms-server, hadoop-mapred, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server,</td>
</tr>
</tbody>
</table>
Creating a Cluster with HBase

The procedures in this section cover the basics of launching a cluster using the AWS Management Console and the AWS CLI. For detailed information about how to plan, configure, and launch EMR clusters, see Plan and Configure Clusters in the Amazon EMR Management Guide.

Creating a Cluster with HBase Using the Console

For quick steps to launch clusters with the console, see Step 3: Launch an Amazon EMR Cluster in the Amazon EMR Management Guide.

To launch a cluster with HBase installed using the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster and Go to advanced options.
3. For Software Configuration, choose an Amazon Release Version of 4.6.0 or later (we recommend the latest version). Choose HBase and other applications as desired.
4. With Amazon EMR version 5.2.0 and later, under HBase Storage Settings, select HDFS or S3. For more information, see HBase on Amazon S3.
5. Select other options as necessary and then choose Create cluster.

Creating a Cluster with HBase Using the AWS CLI

Use the following command to create a cluster with HBase installed:

```
aws emr create-cluster --name "Test cluster" --release-label emr-5.11.0 \ 
--applications Name=HBase --use-default-roles --ec2-attributes KeyName=myKey \ 
--instance-type m3.xlarge --instance-count 3
```

**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

If you use HBase on Amazon S3, specify the --configurations option with a reference to a JSON configuration object. The configuration object must contain an hbase-site classification that specifies the location in Amazon S3 where HBase data is stored using the hbase.rootdir property. It also must contain an hbase classification, which specifies s3 using the hbase.emr.storageMode property. The following example demonstrates a JSON snippet with these configuration settings.

```
{
    "Classification": "hbase-site",
    "Properties": {
        "hbase.rootdir": "s3://MyBucket/MyHBaseStore",
    }
}
```
For more information about HBase on Amazon S3, see HBase on Amazon S3 (Amazon S3 Storage Mode) (p. 89). For more information about classifications, see Configuring Applications (p. 30).

HBase on Amazon S3 (Amazon S3 Storage Mode)

When you run HBase on Amazon EMR version 5.2.0 or later, you can enable HBase on Amazon S3, which offers the following advantages:

- The HBase root directory is stored in Amazon S3, including HBase store files and table metadata. This data is persistent outside of the cluster, available across Amazon EC2 Availability Zones, and you don’t need to recover using snapshots or other methods.
- With store files in Amazon S3, you can size your Amazon EMR cluster for your compute requirements instead of data requirements, with 3x replication in HDFS.
- Using Amazon EMR version 5.7.0 or later, you can set up a read-replica cluster, which allows you to maintain read-only copies of data in Amazon S3. You can access the data from the read-replica cluster to perform read operations simultaneously, and in the event that the primary cluster becomes unavailable.

The following illustration shows the HBase components relevant to HBase on Amazon S3.
Enabling HBase on Amazon S3

You can enable HBase on Amazon S3 using the Amazon EMR console, the AWS CLI, or the Amazon EMR API. The configuration is an option during cluster creation. When you use the console, you choose the setting using Advanced options. When you use the AWS CLI, use the --configurations option to provide a JSON configuration object. Properties of the configuration object specify the storage mode and the root directory location in Amazon S3. The Amazon S3 location that you specify should be in the same region as your Amazon EMR cluster. Only one active cluster at a time can use the same HBase root directory in Amazon S3. For console steps and a detailed create-cluster example using the AWS CLI, see Creating a Cluster with HBase (p. 88). An example configuration object is shown in the following JSON snippet.

Important
We strongly recommend that you use EMRFS consistent view when you enable HBase on Amazon S3 for production workloads. Not using consistent view may result in performance impacts for specific operations. For more information about configuring consistent view, see Consistent View in the Amazon EMR Management Guide.

```json
{
    "Classification": "hbase-site",
    "Properties": {
        "hbase.rootdir": "s3://my-bucket/my-hbase-rootdir"
    }
},
{
    "Classification": "hbase",
    "Properties": {
        "hbase.emr.storageMode": "s3",
        "hbase.emr.readreplica.enabled": "true"
    }
}
```

Using a Read-Replica Cluster

After you set up a primary cluster using HBase on Amazon S3, you can create and configure a read-replica cluster that provides read-only access to the same data as the primary cluster. This is useful when you need simultaneous access to query data or uninterrupted access if the primary cluster becomes unavailable. The read-replica feature is available with Amazon EMR version 5.7.0 and later.

The primary cluster and the read-replica cluster are set up the same way with one important difference. Both point to the same hbase.rootdir location. However, the hbase classification for the read-replica cluster includes the "hbase.emr.readreplica.enabled": "true" property.

For example, given the JSON classification for the primary cluster as shown earlier in the topic, the configuration for a read-replica cluster is as follows:

```json
{
    "Classification": "hbase-site",
    "Properties": {
        "hbase.rootdir": "s3://my-bucket/my-hbase-rootdir"
    }
},
{
    "Classification": "hbase",
    "Properties": {
        "hbase.emr.storageMode": "s3",
        "hbase.emr.readreplica.enabled": "true",
        "hbase.emr.readreplica.rootdir": "s3://my-bucket/my-hbase-rootdir"
    }
}
```
Synchronizing the Read Replica When You Add Data

Because the read-replica uses HBase StoreFiles and metadata that the primary cluster writes to Amazon S3, the read-replica is only as current as the Amazon S3 data store. The following guidance can help minimize the lag time between the primary cluster and the read-replica when you write data.

- Bulk load data on the primary cluster whenever possible. For more information, see Bulk Loading in Apache HBase documentation.
- A flush that writes store files to Amazon S3 should occur as soon as possible after data is added. Either flush manually or tune flush settings to minimize lag time.
- If compactions might run automatically, run a manual compaction to avoid inconsistencies when compactions are triggered.
- On the read-replica cluster, when any metadata has changed—for example, when HBase region split or compactions occur, or when tables are added or removed—run the refresh_meta command.
- On the read-replica cluster, run the refresh_hfiles command when records are added to or changed in a table.

Operational Considerations

HBase region servers use BlockCache to store data reads in memory and BucketCache to store data reads on local disk. In addition, region servers use MemStore to store data writes in-memory, and use write-ahead logs to store data writes in HDFS before the data is written to HBase StoreFiles in Amazon S3. The read performance of your cluster relates to how often a record can be retrieved from the in-memory or on-disk caches. A cache miss results in the record being read from the StoreFile in Amazon S3, which has significantly higher latency and higher standard deviation than reading from HDFS. In addition, the maximum request rates for Amazon S3 are lower than what can be achieved from the local cache, so caching data may be important for read-heavy workloads. For more information about Amazon S3 performance, see Performance Optimization in the Amazon Simple Storage Service Developer Guide.

To improve performance, we recommend that you cache as much of your dataset as possible in EC2 instance storage. Because the BucketCache uses the region server's EC2 instance storage, you can choose an EC2 instance type with a sufficient instance store and add Amazon EBS storage to accommodate the required cache size. You can also increase the BucketCache size on attached instance stores and EBS volumes using the hbase.bucketcache.size property. The default setting is 8,192 MB.

For writes, the frequency of MemStore flushes and the number of StoreFiles present during minor and major compactions can contribute significantly to an increase in region server response times. For
optimal performance, consider increasing the size of the MemStore flush and HRegion block multiplier, which increases the elapsed time between major compactions, but also increases the lag in consistency if you use a read-replica. In some cases, you may get better performance using larger file block sizes (but less than 5 GB) to trigger Amazon S3 multipart upload functionality in EMRFS. Additionally, HBase compactions and region servers perform optimally when fewer StoreFiles need to be compacted.

Tables can take a significant amount of time to drop on Amazon S3 because large directories need to be renamed. Consider disabling tables instead of dropping.

There is an HBase cleaner process that cleans up old WAL files and store files. The cleaner operation can affect query performance when running heavy workloads, so we recommend you enable the cleaner only during off-peak times. The cleaner has the following HBase shell commands:

- `cleaner_enabled` queries whether the cleaner is enabled.
- `cleaner_run` manually runs the cleaner to remove files.
- `cleaner_switch` enables or disables the cleaner and returns the previous state of the cleaner. For example, `cleaner-switch true` enables the cleaner.

### Properties for HBase on Amazon S3 Performance Tuning

The following parameters can be adjusted to tune the performance of your workload when you use HBase on Amazon S3.

<table>
<thead>
<tr>
<th>Configuration Property</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hbase.bucketcache.size</code></td>
<td>8,192</td>
<td>The amount of disk space, in MB, reserved on region server Amazon EC2 instance stores and EBS volumes for BucketCache storage. The setting applies to all region server instances. Larger BucketCache sizes generally correspond to improved performance.</td>
</tr>
<tr>
<td><code>hbase.hregion.memstore.flush.size</code></td>
<td>134217728</td>
<td>The data limit, in bytes, at which a memstore flush to Amazon S3 is triggered.</td>
</tr>
<tr>
<td><code>hbase.hregion.memstore.block.multiplier</code></td>
<td></td>
<td>A multiplier that determines the MemStore upper limit at which updates are blocked. If the MemStore exceeds <code>hbase.hregion.memstore.flush.size</code> multiplied by this value, updates are blocked. MemStore flushes and compaction may happen to unblock updates.</td>
</tr>
<tr>
<td><code>hbase.hstore.blockingStoreFiles</code></td>
<td></td>
<td>The maximum number of StoreFiles that can exist in a store before updates are blocked.</td>
</tr>
<tr>
<td><code>hbase.hregion.max.filesize</code></td>
<td>10737418240</td>
<td>The maximum size of a region before the region is split.</td>
</tr>
</tbody>
</table>
Shutting Down and Restoring a Cluster Without Data Loss

To shut down an Amazon EMR cluster without losing data that hasn’t been written to Amazon S3, the MemStore cache needs to flush to Amazon S3 to write new store files. To do this, you can run a shell script provided on the EMR cluster. You can either add it as a step or run it directly using the on-cluster CLI. The script disables all HBase tables, which causes the MemStore in each region server to flush to Amazon S3. If the script completes successfully, the data persists in Amazon S3 and the cluster can be terminated.

The following step configuration can be used when you add a step to the cluster. For more information, see Work with Steps Using the CLI and Console in the Amazon EMR Management Guide.

```
Name="Disable all tables",Jar="command-runner.jar",Args=["/bin/bash","/usr/lib/hbase/bin/disable_all_tables.sh"]
```

Alternatively, you can run the following bash command directly.

```
bash /usr/lib/hbase/bin/disable_all_tables.sh
```

To restart a cluster with the same HBase data, specify the same Amazon S3 location as the previous cluster either in the AWS Management Console or using the `hbase.rootdir` configuration property.

Using the HBase Shell

After you create an HBase cluster, the next step is to connect to HBase so you can begin reading and writing data (data writes are not supported on a read-replica cluster). You can use the HBase shell to test commands.

**To open the HBase shell**

1. Use SSH to connect to the master server in the HBase cluster. For information about how to connect to the master node using SSH, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide.
2. Run `hbase shell`. The HBase shell opens with a prompt similar to the following example:

```
hbase(main):001:0>
```

You can issue HBase shell commands from the prompt. For more information about the shell commands and how to call them, type `help` at the HBase prompt and press Enter.

**Create a Table**

The following command creates a table named ‘t1’ that has a single column family named ‘f1’:

```
hbase(main):001:0>create 't1', 'f1'
```
Put a Value

The following command puts value 'v1' for row 'r1' in table 't1' and column 'f1':

```
  hbase(main):001:0> put 't1', 'r1', 'f1:col1', 'v1'
```

Get a Value

The following command gets the values for row 'r1' in table 't1':

```
  hbase(main):001:0> get 't1', 'r1'
```

Access HBase Tables with Hive

HBase and Apache Hive (p. 109) are tightly integrated, allowing you run massively parallel processing workloads directly on data stored in HBase. To use Hive with HBase, you can usually launch them on the same cluster. You can, however, launch Hive and HBase on separate clusters. Running HBase and Hive separately on different clusters can improve performance because this allows each application to use cluster resources more efficiently.

The following procedures show how to connect to HBase on a cluster using Hive.

**Note**

You can only connect a Hive cluster to a single HBase cluster.

To connect Hive to HBase

1. Create separate clusters with Hive and HBase installed or create a single cluster with both HBase and Hive installed.
2. If you are using separate clusters, modify your security groups so that HBase and Hive ports are open between these two master nodes.
3. Use SSH to connect to the master node for the cluster with Hive installed. For more information, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide.
4. Launch the Hive shell with the following command.

```
hive
```

5. (Optional) You do not need to do this if HBase and Hive are located on the same cluster. Connect the HBase client on your Hive cluster to the HBase cluster that contains your data. In the following example, `public-DNS-name` is replaced by the public DNS name of the master node of the HBase cluster, for example: ec2-50-19-76-67.compute-1.amazonaws.com.

```
  set hbase.zookeeper.quorum=public-DNS-name;
```
6. Proceed to run Hive queries on your HBase data as desired or see the next procedure.

To access HBase data from Hive

- After the connection between the Hive and HBase clusters has been made (as shown in the previous procedure), you can access the data stored on the HBase cluster by creating an external table in Hive.

The following example, when run from the Hive prompt, creates an external table that references data stored in an HBase table called `inputTable`. You can then reference `inputTable` in Hive statements to query and modify data stored in the HBase cluster.

```
set hbase.zookeeper.quorum=ec2-107-21-163-157.compute-1.amazonaws.com ;
create external table inputTable (key string, value string) 
    stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    with serdeproperties ("hbase.columns.mapping" = ":key,f1:col1")
    tblproperties ("hbase.table.name" = "t1");
select count(*) from inputTable ;
```

For a more advanced use case and example combining HBase and Hive, see the AWS Big Data Blog post, Combine NoSQL and Massively Parallel Analytics Using Apache HBase and Apache Hive on Amazon EMR.

Using HBase Snapshots

HBase uses a built-in snapshot functionality to create lightweight backups of tables. In EMR clusters, these backups can be exported to Amazon S3 using EMRFS. You can create a snapshot on the master node using the HBase shell. This topic shows you how to run these commands interactively with the shell or through a step using `command-runner.jar` with either the AWS CLI or AWS SDK for Java. For more information about other types of HBase backups, see HBase Backup in the HBase documentation.

Create a snapshot using a table

```
hbase snapshot create -n snapshotName -t tableName
```

Using command-runner.jar from the AWS CLI:

```
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF \
--steps Name="HBase Shell Step",Jar="command-runner.jar",\ 
Args=[ "hbase", "snapshot", "create","-n","snapshotName","-t","tableName"]
```

AWS SDK for Java

```
HadoopJarStepConfig hbaseSnapshotConf = new HadoopJarStepConfig() 
    .withJar("command-runner.jar") 
    .withArgs("hbase","snapshot","create","-n","snapshotName","-t","tableName");
```

Note

If your snapshot name is not unique, the create operation fails with a return code of -1 or 255 but you may not see an error message that states what went wrong. To use the same snapshot name, delete it and then re-create it.
Delete a snapshot

hbase shell
>> delete_snapshot 'snapshotName'

View snapshot info

hbase snapshot info -snapshot snapshotName

Export a snapshot to Amazon S3

Important
If you do not specify a -mappers value when exporting a snapshot, HBase uses an arbitrary calculation to determine the number of mappers. This value can be very large depending on your table size, which negatively affects running jobs during the export. For this reason, we recommend that you specify the -mappers parameter, the -bandwidth parameter (which specifies the bandwidth consumption in megabytes per second), or both to limit the cluster resources used by the export operation. Alternatively, you can run the export snapshot operation during a period of low usage.

hbase snapshot export -snapshot snapshotName \\ -copy-to s3://bucketName/folder -mappers 2

Using command-runner.jar from the AWS CLI:

aws emr add-steps --cluster-id j-2AXXXXXXGAPLF \\ --steps Name="HBase Shell Step",Jar="command-runner.jar",\ Abs=[ "hbase", "snapshot", "export", "-snapshot","snapshotName","-copy-to","s3://bucketName/folder","-mappers","2","-bandwidth","50"]

AWS SDK for Java:

HadoopJarStepConfig hbaseImportSnapshotConf = new HadoopJarStepConfig() \\ .withJar("command-runner.jar") \\ .withArgs("hbase", "snapshot", "export", \\ "-snapshot","snapshotName","-copy-to", \\ "s3://bucketName/folder", \\ "-mappers","2","-bandwidth","50");

Import snapshot from Amazon S3

Although this is an import, the HBase option used here is still export.

sudo -u hbase hbase snapshot export \\ -D hbase.rootdir=s3://bucketName/folder \\ -snapshot snapshotName \\ -copy-to hdfs://masterPublicDNSName:8020/user/hbase \\ -mappers 2

Using command-runner.jar from the AWS CLI:

aws emr add-steps --cluster-id j-2AXXXXXXGAPLF \\
--steps Name="HBase Shell Step",Jar="command-runner.jar", \
Args=['sudo','-u','hbase','hbase snapshot export','-snapshot','"snapshotName"', \
'-D','hbase.rootdir=s3://bucketName/folder', \
'-copy-to','hdfs://masterPublicDNSName:8020/user/hbase','-mappers','2','-chmod','700']

AWS SDK for Java:

HadoopJarStepConfig hbaseImportSnapshotConf = new HadoopJarStepConfig() 
.withJar("command-runner.jar") 
.withArgs("sudo","-u","hbase","hbase","snapshot","export","-D","hbase.rootdir=s3://bucketName/folder", 
"-snapshot",""snapshotName"","-copy-to", 
"hdfs://masterPublicDNSName:8020/user/hbase","-mappers","2","-chmod","700")

Restore a table from snapshots within the HBase shell

hbase shell
>> disable tableName
>> restore_snapshot snapshotName
>> enable tableName

HBase currently does not support all snapshot commands found in the HBase shell. For example, there is no HBase command-line option to restore a snapshot, so you must restore it within a shell. This means that command-runner.jar must run a Bash command.

Note
Because the command used here is echo, it is possible that your shell command will still fail even if the command run by Amazon EMR returns a 0 exit code. Check the step logs if you choose to run a shell command as a step.

Here is the step using the AWS CLI. First, create the following snapshot.json file:

```json
[ 
  { 
    "Name": "restore", 
    "Args": ["bash","-c","echo \"disable \"tableName\"\"; restore_snapshot \"snapshotName\"\"; enable \"tableName\"\" | hbase shell\"]
    "Jar": "command-runner.jar", 
    "ActionOnFailure": "CONTINUE", 
    "Type": "CUSTOM_JAR"
  }
]
```

aws emr add-steps --cluster-id j-2AXXXXXXGAPLF \
--steps file:///./snapshot.json

AWS SDK for Java:

HadoopJarStepConfig hbaseRestoreSnapshotConf = new HadoopJarStepConfig() 
 .withJar("command-runner.jar")
Configure HBase

Although the default settings should work for most applications, you have the flexibility to modify your HBase configuration settings. To do this, use the configuration API when you create the cluster:

The following example creates a cluster with an alternate HBase root directory:

Note
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

```
aws emr create-cluster --release-label emr-5.11.0 --applications Name=HBase \
--instance-type m3.xlarge --instance-count 2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

```
myConfig.json:

```
[
  {
    "Classification":"hbase-site",
    "Properties": {
      "hbase.rootdir": "hdfs://ip-XXX-XX-XX-XXX.ec2.internal:8020/user/myCustomHBaseDir"
    }
  }
]
```

Note
If you plan to store your configuration in Amazon S3, you must specify the URL location of the object. For example:

```
aws emr create-cluster --release-label emr-5.11.0 --applications Name=HBase \
--instance-type m3.xlarge --instance-count 3 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Changes to Memory Allocation in YARN

HBase is not running as a YARN application, thus it is necessary to recalculate the memory allocated to YARN and its applications, which results in a reduction in overall memory available to YARN if HBase is installed. You should take this into account when planning to co-locate YARN applications and HBase on the same clusters. The instance types with less than 64 GB of memory have half the memory available to NodeManager, which is then allocated to the HBase RegionServer. For instance types with memory greater than 64 GB, HBase RegionServer memory is capped at 32 GB. As a general rule, YARN setting memory is some multiple of MapReduce reducer task memory.

The tables in Task JVM Memory Settings (p. 66) show changes to YARN settings based on the memory needed for HBase.

HBase Port Numbers

Some port numbers chosen for HBase are different from the default. The following are interfaces and ports for HBase on Amazon EMR.
HBase Ports

<table>
<thead>
<tr>
<th>Interface</th>
<th>Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMaster</td>
<td>16000</td>
<td>TCP</td>
</tr>
<tr>
<td>HMaster UI</td>
<td>16010</td>
<td>HTTP</td>
</tr>
<tr>
<td>RegionServer</td>
<td>16020</td>
<td>TCP</td>
</tr>
<tr>
<td>RegionServer Info</td>
<td>16030</td>
<td>HTTP</td>
</tr>
<tr>
<td>REST server</td>
<td>8070</td>
<td>HTTP</td>
</tr>
<tr>
<td>REST UI</td>
<td>8085</td>
<td>HTTP</td>
</tr>
<tr>
<td>Thrift server</td>
<td>9090</td>
<td>TCP</td>
</tr>
<tr>
<td>Thrift server UI</td>
<td>9095</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

**Important**

The Hadoop KMS port is changed in Amazon EMR release 4.6 or later. `kms-http-port` is now 9700 and `kms-admin-port` is 9701.

HBase Site Settings to Optimize

You can set any or all of the HBase site settings to optimize the HBase cluster for your application’s workload. We recommend the following settings as a starting point in your investigation.

**zookeeper.session.timeout**

The default timeout is three minutes (180000 ms). If a region server crashes, this is how long it takes the master server to notice the absence of the region server and start recovery. To help the master server recover faster, you can reduce this value to a shorter time period. The following example uses one minute, or 60000 ms:

```
[
  {
    "Classification":"hbase-site",
    "Properties": {
      "zookeeper.session.timeout": "60000"
    }
  }
]
```

**hbase.regionserver.handler.count**

This defines the number of threads the region server keeps open to serve requests to tables. The default of 10 is low, in order to prevent users from killing their region servers when using large write buffers with a high number of concurrent clients. The rule of thumb is to keep this number low when the payload per request approaches the MB range (big puts, scans using a large cache) and high when the payload is small (gets, small puts, ICVs, deletes). The following example raises the number of open threads to 30:

```
[
  {
    "Classification":"hbase-site",
    "Properties": {
```
hbase.regionserver.handler.count

This parameter governs the number of handlers per region server. Each handler is responsible for handling client requests. By default, it is set to 30. If you are running a lot of traffic, you can increase this limit to reduce the blocking time for client requests.

```
[
  {  
    "Classification": "hbase-site",
    "Properties": {
      "hbase.regionserver.handler.count": "30"
    }
  }
]
```

hbase.hregion.max.filesize

This parameter governs the size, in bytes, of the individual regions. By default, it is set to 256 MB. If you are writing a lot of data into your HBase cluster and it's causing frequent splitting, you can increase this size to make individual regions bigger. It reduces splitting but takes more time to load balance regions from one server to another.

```
[
  {  
    "Classification": "hbase-site",
    "Properties": {
      "hbase.hregion.max.filesize": "1073741824"
    }
  }
]
```

hbase.hregion.memstore.flush.size

This parameter governs the maximum size of memstore, in bytes, before it is flushed to disk. By default, it is 64 MB. If your workload consists of short bursts of write operations, you might want to increase this limit so all writes stay in memory during the burst and get flushed to disk later. This can boost performance during bursts.

```
[
  {  
    "Classification": "hbase-site",
    "Properties": {
      "hbase.hregion.memstore.flush.size": "134217728"
    }
  }
]
```

View the HBase User Interface

HBase provides a web-based user interface that you can use to monitor your HBase cluster. When you run HBase on Amazon EMR, the web interface runs on the master node and can be viewed using port forwarding, also known as creating an SSH tunnel.

To view the HBase User Interface

1. Use SSH to tunnel into the master node and create a secure connection. For more information, see Option 2, Part 1: Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding in the Amazon EMR Management Guide.
2. Install a web browser with a proxy tool, such as the FoxyProxy plug-in for Firefox, to create a SOCKS proxy for AWS domains. For more information, see Option 2, Part 2: Configure Proxy Settings to View Websites Hosted on the Master Node in the Amazon EMR Management Guide.
3. With the proxy set and the SSH connection open, you can view the HBase UI by opening a browser window with `http://master-public-dns-name:16010/master-status`, where `master-public-dns-name` is the public DNS address of the master server in the HBase cluster.
You can also view HBase in Hue. For example, the following shows the table, t1, created in Using the HBase Shell (p. 93):

For more information about Hue, see Hue (p. 119).

View HBase Log Files

As part of its operation, HBase writes log files with details about configuration settings, daemon actions, and exceptions. These log files can be useful for debugging issues with HBase as well as for tracking performance.

If you configure your cluster to persist log files to Amazon S3, you should know that logs are written to Amazon S3 every five minutes, so there may be a slight delay before the latest log files are available.

To view HBase logs on the master node

- You can view the current HBase logs by using SSH to connect to the master node, and navigating to the /var/log/hbase directory. These logs are not available after the cluster is terminated unless
you enable logging to Amazon S3 when the cluster is launched. For more information, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide. After you have connected to the master node using SSH, you can navigate to the log directory using a command like the following:

```
$ cd /var/log/hbase
```

To view HBase logs on Amazon S3

- To access HBase logs and other cluster logs on Amazon S3, and to have them available after the cluster ends, you must specify an Amazon S3 bucket to receive these logs when you create the cluster. This is done using the `--log-uri` option. For more information about enabling logging for your cluster, see Configure Logging and Debugging (Optional) in the Amazon EMR Management Guide.

Monitor HBase with Ganglia

The Ganglia open-source project is a scalable, distributed system designed to monitor clusters and grids while minimizing the impact on their performance. When you enable Ganglia on your cluster, you can generate reports and view the performance of the cluster as a whole, as well as inspect the performance of individual node instances. For more information about the Ganglia open-source project, see http://ganglia.info/. For more information about using Ganglia with Amazon EMR clusters, see Ganglia (p. 84).

After the cluster is launched with Ganglia configured, you can access the Ganglia graphs and reports using the graphical interface running on the master node.

Ganglia also stores log files on the server at `/var/log/ganglia/rrds`. If you configured your cluster to persist log files to an Amazon S3 bucket, the Ganglia log files are persisted there as well.

To configure a cluster for Ganglia and HBase using the AWS CLI

- Create the cluster with HBase and Ganglia installed using the AWS CLI:

  ```
  aws emr create-cluster --name "Test cluster" --release-label emr-5.11.0 \
  --applications Name=HBase Name=Ganglia --use-default-roles \
  --ec2-attributes KeyName=myKey --instance-type c1.xlarge \
  --instance-count 3 --use-default-roles
  ```

  When you specify the instance count without using the `--instance-groups` parameter, a single master node is launched, and the remaining instances are launched as core nodes. All nodes use the instance type specified in the command.

  **Note**
  
  If you have not previously created the default Amazon EMR service role and Amazon EC2 instance profile, type `aws emr create-default-roles` to create them before typing the `create-cluster` subcommand.

  For more information, see Amazon EMR commands in the AWS CLI.

To view HBase metrics in the Ganglia web interface

1. Use SSH to tunnel into the master node and create a secure connection. For more information, see Option 2, Part 1: Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding in the Amazon EMR Management Guide.
2. Install a web browser with a proxy tool, such as the FoxyProxy plug-in for Firefox, to create a SOCKS proxy for AWS domains. For more information, see Option 2, Part 2: Configure Proxy Settings to View Websites Hosted on the Master Node in the Amazon EMR Management Guide.

3. With the proxy set and the SSH connection open, you can view the Ganglia metrics by opening a browser window with http://master-public-dns-name/ganglia/, where master-public-dns-name is the public DNS address of the master server in the HBase cluster.

To view Ganglia log files on the master node

- If the cluster is still running, you can access the log files by using SSH to connect to the master node and navigating to the /var/log/ganglia/rrds directory. For more information, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide.

To view Ganglia log files on Amazon S3

- If you configured the cluster to persist log files to Amazon S3 when you launched it, the Ganglia log files are written there as well. Logs are written to Amazon S3 every five minutes, so there may be a slight delay before the latest log files are available. For more information, see View HBase Log Files (p. 101).

Migrating from Previous HBase Versions

To migrate data from a previous HBase version, see Upgrading and HBase version number and compatibility in the Apache HBase Reference Guide. You may need to pay special attention to the requirements for upgrading from pre-1.0 versions of HBase.
Apache HCatalog

HCatalog is a tool that allows you to access Hive metastore tables within Pig, Spark SQL, and/or custom MapReduce applications. HCatalog has a REST interface and command line client that allows you to create tables or do other operations. You then write your applications to access the tables using HCatalog libraries. For more information, see Using HCatalog.

HCatalog on Amazon EMR release 5.8.0 and later supports using AWS Glue Data Catalog as the metastore for Hive. For more information, see Using AWS Glue Data Catalog as the Metastore for Hive

Note

HCatalog is included only in Amazon EMR versions 4.4.0 and later.

HCatalog Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Creating a Cluster with HCatalog

Although HCatalog is included in the Hive project, you still must install it on EMR clusters as its own application.

To launch a cluster with HCatalog installed using the console

The following procedure creates a cluster with HCatalog installed. For more information about creating clusters using the console, including Advanced Options see Plan and Configure Clusters in the Amazon EMR Management Guide.

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster to use Quick Create.
To launch a cluster with HCatalog using the AWS CLI

- Create the cluster with the following command:

```bash
aws emr create-cluster --name "Cluster with Hcat" --release-label emr-5.11.0 \
--applications Name=HCatalog --ec2-attributes KeyName=myKey \ 
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

Using HCatalog

You can use HCatalog within various applications that use the Hive metastore. The examples in this section show how to create a table and use it in the context of Pig and Spark SQL.

Disable Direct Write When Using HCatalog HStorer

Whenever an application uses HCatStorer (for example, a Pig script `STORE` command) to write to an HCatalog table stored in Amazon S3, disable the direct write feature of Amazon EMR. You can do this by setting the `mapred.output.direct.NativeS3FileSystem` and the `mapred.output.direct.EmrFileSystem` configurations to `false`. See the Pig example below for an example of setting these configurations from within a Pig script before running subsequent Pig commands. The following example demonstrates how to set these configurations using Java.

```java
Configuration conf = new Configuration();
conf.set("mapred.output.direct.NativeS3FileSystem", "false");
conf.set("mapred.output.direct.EmrFileSystem", "false");
```

Create a Table Using the HCat CLI and Use That Data in Pig

Create the following script, `impressions.q`, on your cluster:

```sql
CREATE EXTERNAL TABLE impressions (requestBeginTime string, adId string, impressionId string, referrer string, userAgent string, userCookie string, ip string) PARTITIONED BY (dt string) ROW FORMAT serde 'org.apache.hive.hcatalog.data.JsonSerDe' with serdeproperties ('paths'='requestBeginTime, adId, impressionId, referrer, userAgent, userCookie, ip') LOCATION 's3://[your region].elasticmapreduce/samples/hive-ads/tables/impressions/';
```

Execute the script using the HCat CLI:
% hcat -f impressions.q
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
OK
Time taken: 4.001 seconds
OK
Time taken: 0.519 seconds

Open the Grunt shell and access the data in impressions:

% pig -useHCatalog -e "A = LOAD 'impressions' USING org.apache.hive.hcatalog.pig.HCatLoader();
B = LIMIT A 5;
dump B;"

---

Accessing the Table using Spark SQL

This example creates a Spark DataFrame from the table created in the first example and shows the first 20 lines:

% spark-shell --jars /usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.0.0-amzn-3.jar
<snip>
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc);
scala> val df = hiveContext.sql("SELECT * FROM impressions")
scala> df.show()
Example: Create an HCatalog Table and Write to it Using Pig

You can create an HCatalog table and use Apache Pig to write to it by way of HCatStorer using a data source in Amazon S3. HCatalog requires that you disable direct write, or the operation fails silently. Set both the mapred.output.direct.NativeS3FileSystem and the mapred.output.direct.EmrFileSystem configurations to false either using the mapred-site classification, or manually from within the Grunt shell. The following example shows a table created using the HCat CLI, followed by commands executed in the Grunt shell to populate the table from a sample data file in Amazon S3.

To run this example, connect to the Master node using SSH.

Create an HCatalog script file, `wikicount.q`, with the following contents, which creates an HCatalog table named `wikicount`.

```scala
CREATE EXTERNAL TABLE IF NOT EXISTS wikicount(
    col1 string,
    col2 string
)
```

Example: Create an HCatalog Table and Write to it Using Pig

```sql
col2 bigint
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
STORED AS ORC
LOCATION 's3://MyBucket/hcat/wikicount';

Use an HCat CLI command to execute the script from the file.

hcat -f wikicount.q

Next, start the Grunt shell with the -useHCatalog option, set configurations to disable direct write, load data from an S3 location, and then write the results to the wikicount table.

pig -useHCatalog
SET mapred.output.direct.NativeS3FileSystem false;
SET mapred.output.direct.EmrFileSystem false;
A = LOAD 's3://support.elasticmapreduce/training/datasets/wikistats_tiny/' USING PigStorage(' ') AS (Site:chararray, page:chararray, views:int, total_bytes:long);
B = GROUP A BY Site;
C = FOREACH B GENERATE group as col1, COUNT(A) as col2;
STORE C INTO 'wikicount' USING org.apache.hive.hcatalog.pig.HCatStorer();
```
Apache Hive

Hive is an open-source, data warehouse, and analytic package that runs on top of a Hadoop cluster. Hive scripts use an SQL-like language called Hive QL (query language) that abstracts programming models and supports typical data warehouse interactions. Hive enables you to avoid the complexities of writing Tez jobs based on directed acyclic graphs (DAGs) or MapReduce programs in a lower level computer language, such as Java.

Hive extends the SQL paradigm by including serialization formats. You can also customize query processing by creating table schema that match your data, without touching the data itself. In contrast to SQL (which only supports primitive value types such as dates, numbers, and strings), values in Hive tables are structured elements, such as JSON objects, any user-defined data type, or any function written in Java.

For more information about Hive, see http://hive.apache.org/.

Amazon EMR sample applications are included with each release. You can view these samples by logging into the master node of your cluster at /usr/share/aws/emr/samples.

Hive Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Differences for Hive on Amazon EMR Versions and Default Apache Hive

Topics
Differences between Apache Hive on Amazon EMR and Apache Hive

This section describes the differences between Hive on Amazon EMR and the default versions of Hive available at http://svn.apache.org/viewvc/hive/branches/.

Hive Live Long and Process (LLAP) not Supported

LLAP functionality added in version 2.0 of default Apache Hive is not supported in Hive 2.1.0 on Amazon EMR release 5.0.

Differences in Hive Between Amazon EMR Release 4.x and 5.x

This section covers differences to consider before you migrate a Hive implementation from Hive version 1.0.0 on Amazon EMR release 4.x to Hive 2.x on Amazon EMR release 5.x.

Operational Differences and Considerations

- **Support added for ACID (Atomicity, Consistency, Isolation, and Durability) transactions**: This difference between Hive 1.0.0 on Amazon EMR 4.x and default Apache Hive has been eliminated.
- **Direct writes to Amazon S3 eliminated**: This difference between Hive 1.0.0 on Amazon EMR and the default Apache Hive has been eliminated. Hive 2.1.0 on Amazon EMR release 5.x now creates, reads from, and writes to temporary files stored in Amazon S3. As a result, to read from and write to the same table you no longer have to create a temporary table in the cluster's local HDFS file system as a workaround. If you use versioned buckets, be sure to manage these temporary files as described below.
- **Manage temp files when using Amazon S3 versioned buckets**: When you run Hive queries where the destination of generated data is Amazon S3, many temporary files and directories are created. This is new behavior as described earlier. If you use versioned S3 buckets, these temp files clutter Amazon S3 and incur cost if they’re not deleted. Adjust your lifecycle rules so that data with a /_tmp prefix is deleted after a short period, such as five days. See Specifying a Lifecycle Configuration for more information.
- **Log4j updated to Log4j 2**: If you use log4j, you may need to change your logging configuration because of this upgrade. See Apache log4j 2 for details.

Performance differences and considerations

- **Performance differences with Tez**: With Amazon EMR release 5.x, Tez is the default execution engine for Hive instead of MapReduce. Tez provides improved performance for most workflows.
- **ORC file performance**: Query performance may be slower than expected for ORC files.
- **Tables with many partitions**: Queries that generate a large number of dynamic partitions may fail, and queries that select from tables with many partitions may take longer than expected to execute. For example, a select from 100,000 partitions may take 10 minutes or more.
Additional Features of Hive on Amazon EMR

Amazon EMR extends Hive with new features that support Hive integration with other AWS services, such as the ability to read from and write to Amazon Simple Storage Service (Amazon S3) and DynamoDB.

Topics
- Amazon EMR Hive Queries to Accommodate Partial DynamoDB Schemas (p. 111)
- Copy Data Between DynamoDB Tables in Different AWS Regions (p. 112)
- Set DynamoDB Throughput Values Per Table (p. 112)

Amazon EMR Hive Queries to Accommodate Partial DynamoDB Schemas

Amazon EMR Hive provides maximum flexibility when querying DynamoDB tables by allowing you to specify a subset of columns on which you can filter data, rather than requiring your query to include all columns. This partial schema query technique is effective when you have a sparse database schema and want to filter records based on a few columns, such as filtering on time stamps.

The following example shows how to use a Hive query to:
- Create a DynamoDB table.
- Select a subset of items (rows) in DynamoDB and further narrow the data to certain columns.
- Copy the resulting data to Amazon S3.

```
DROP TABLE dynamodb;
DROP TABLE s3;

CREATE EXTERNAL TABLE dynamodb(hashKey STRING, recordTimeStamp BIGINT, fullColumn map<String, String>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
   "dynamodb.table.name" = "myTable",
   "dynamodb.throughput.read.percent" = ".1000",
   "dynamodb.column.mapping" = "hashKey:HashKey,recordTimeStamps:RangeKey");

CREATE EXTERNAL TABLE s3(map<String, String>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3 SELECT item fullColumn FROM dynamodb WHERE recordTimeStamps < "2012-01-01";
```

The following table shows the query syntax for selecting any combination of items from DynamoDB.

<table>
<thead>
<tr>
<th>Query example</th>
<th>Result description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM table_name;</td>
<td>Selects all items (rows) from a given table and includes data from all columns available for those items.</td>
</tr>
<tr>
<td>SELECT * FROM table_name WHERE field_name =value;</td>
<td>Selects some items (rows) from a given table and includes data from all columns available for those items.</td>
</tr>
</tbody>
</table>
### Query example

<table>
<thead>
<tr>
<th>Query</th>
<th>Result description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT <code>column1_name</code>, <code>column2_name</code>, <code>column3_name</code> FROM <code>table_name</code>;</td>
<td>Selects all items (rows) from a given table and includes data from some columns available for those items.</td>
</tr>
<tr>
<td>SELECT <code>column1_name</code>, <code>column2_name</code>, <code>column3_name</code> FROM <code>table_name</code> WHERE <code>field_name</code> = <code>value</code>;</td>
<td>Selects some items (rows) from a given table and includes data from some columns available for those items.</td>
</tr>
</tbody>
</table>

### Copy Data Between DynamoDB Tables in Different AWS Regions

Amazon EMR Hive provides a `dynamodb.region` property you can set per DynamoDB table. When `dynamodb.region` is set differently on two tables, any data you copy between the tables automatically occurs between the specified regions.

The following example shows you how to create a DynamoDB table with a Hive script that sets the `dynamodb.region` property:

**Note**

Per-table region properties override the global Hive properties.

```sql
CREATE EXTERNAL TABLE dynamodb(hashKey STRING, recordTimeStamp BIGINT, map<String, String> fullColumn) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES (  "dynamodb.table.name" = "myTable",  "dynamodb.region" = "eu-west-1",  "dynamodb.throughput.read.percent" = ".1000",  "dynamodb.column.mapping" = "hashKey:HashKey,recordTime:RangeKey");
```

### Set DynamoDB Throughput Values Per Table

Amazon EMR Hive enables you to set the DynamoDB `readThroughputPercent` and `writeThroughputPercent` settings on a per table basis in the table definition. The following Amazon EMR Hive script shows how to set the throughput values. For more information about DynamoDB throughput values, see [Specifying Read and Write Requirements for Tables](#).

```sql
CREATE EXTERNAL TABLE dynamodb(hashKey STRING, recordTimeStamp BIGINT, map<String, String> fullColumn) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES (  "dynamodb.table.name" = "myTable",  "dynamodb.throughput.read.percent" = ".4",  "dynamodb.throughput.write.percent" = "1.0",  "dynamodb.column.mapping" = "hashKey:HashKey,recordTime:RangeKey");
```

### Configuring an External Metastore for Hive

By default, Hive records metastore information in a MySQL database on the master node's file system. The metastore contains a description of the table and the underlying data on which it is built, including the partition names, data types, and so on. When a cluster terminates, all cluster nodes shut down, including the master node. When this happens, local data is lost because node file systems use ephemeral storage. If you need the metastore to persist, you must create an external metastore that exists outside the cluster.
You have two options for an external metastore:

- AWS Glue Data Catalog (Amazon EMR version 5.8.0 or later only).
  
  For more information, see Using the AWS Glue Data Catalog as the Metastore for Hive (p. 113).

- Amazon RDS or Amazon Aurora.

  For more information, see Using an External MySQL Database or Amazon Aurora (p. 115).

### Using the AWS Glue Data Catalog as the Metastore for Hive

Using Amazon EMR version 5.8.0 or later, you can configure Hive to use the AWS Glue Data Catalog as its metastore. We recommend this configuration when you require a persistent metastore or a metastore shared by different clusters, services, and applications.

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores. The AWS Glue Data Catalog provides a unified metadata repository across a variety of data sources and data formats, integrating with Amazon EMR as well as Amazon RDS, Amazon Redshift, Redshift Spectrum, Athena, and any application compatible with the Apache Hive metastore. AWS Glue crawlers can automatically infer schema from source data in Amazon S3 and store the associated metadata in the Data Catalog. For more information about the Data Catalog, see Populating the AWS Glue Data Catalog in the AWS Glue Developer Guide.

Separate charges apply for AWS Glue. There is a monthly rate for storing and accessing the metadata in the Data Catalog, an hourly rate billed per minute for AWS Glue ETL jobs and crawler runtime, and an hourly rate billed per minute for each provisioned development endpoint. The Data Catalog allows you to store up to a million objects at no charge. If you store more than a million objects, you are charged USD$1 for each 100,000 objects over a million. An object in the Data Catalog is a table, partition, or database. For more information, see Glue Pricing.

**Important**

If you created tables using Amazon Athena or Amazon Redshift Spectrum before August 14, 2017, databases and tables are stored in an Athena-managed catalog, which is separate from the AWS Glue Data Catalog. To integrate Amazon EMR with these tables, you must upgrade to the AWS Glue Data Catalog. For more information, see Upgrading to the AWS Glue Data Catalog in the Amazon Athena User Guide.

### Specifying AWS Glue Data Catalog as the Metastore

You can specify the AWS Glue Data Catalog as the metastore using the AWS Management Console, AWS CLI, or Amazon EMR API. When you create a cluster using the CLI or API, you use the hive-site configuration classification to specify the Data Catalog. When you create a cluster using the console, you can specify the Data Catalog using Advanced Options or Quick Options.

**Note**

The option to use the Data Catalog is also available with HCatalog because Hive is installed with HCatalog.

**To specify AWS Glue Data Catalog as the metastore using the console**

1. Open the Amazon EMR console at [https://console.aws.amazon.com/elasticmapreduce/](https://console.aws.amazon.com/elasticmapreduce/).
2. Choose Create cluster, Go to advanced options.
3. For Release, choose emr-5.8.0 or later.
4. Under Release, select Hive or HCatalog.
5. Under AWS Glue Data Catalog settings select Use for Hive table metadata.
6. Choose other options for your cluster as appropriate, choose Next, and then configure other cluster options as appropriate for your application.

To specify the AWS Glue Data Catalog as the metastore using the AWS CLI or Amazon EMR API

- Specify the value for hive.metastore.client.factory.class using the hive-site classification as shown in the following example. For more information, see Configuring Applications.

Example Configuration JSON for Using the AWS Glue Data Catalog:

```json
[
  {
    "Classification": "hive-site",
    "Properties": {
      "hive.metastore.client.factory.class": "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
    }
  }
]
```

IAM Permissions

The EMR_EC2_DefaultRole must be allowed IAM permissions for AWS Glue actions. This is only a concern if you don’t use the default AmazonElasticMapReduceforEC2Role managed policy and you attach a customer-managed policy to the role. In this case, you need to configure the policy to allow permission to perform AWS Glue actions. Open the IAM console (https://console.aws.amazon.com/iam/) and view the contents of the AmazonElasticMapReduceforEC2Role managed policy to see the required AWS Glue actions to allow.

Unsupported Configurations, Functions, and Known Issues

The limitations listed below apply when using the AWS Glue Data Catalog as a metastore:

- Adding auxiliary JARs using the Hive shell is not supported. As a workaround, add auxiliary JARs into the Hive classpath (specified using hive.aux.jars.path).
- The VALUES keyword is not supported.
- Hive transactions are not supported.
- Renaming tables from within AWS Glue is not supported.
- Partition values containing quotes and apostrophes are not supported (for example, PARTITION (owner="Doe's").
- Table and partition statistics are not supported.
- Using Hive authorization is not supported.
- Hive constraints are not supported.
- Setting hive.metastore.partition.inherit.table.properties is not supported.
- Using the following metastore constants is not supported: BUCKET_COUNT, BUCKET_FIELD_NAME, DDL_TIME, FIELD_TO_DIMENSION, FILE_INPUT_FORMAT, FILE_OUTPUT_FORMAT, HIVE_FILTER_FIELD_LAST_ACCESS, HIVE_FILTER_FIELD_OWNER, HIVE_FILTER_FIELD_PARAMS, IS_ARCHIVED, META_TABLE_COLUMNS,
META_TABLE_COLUMN_TYPES, META_TABLE_DB, META_TABLE_LOCATION, META_TABLE_NAME, META_TABLE_PARTITION_COLUMNS, META_TABLE_SERDE, META_TABLE_STORAGE, ORIGINAL_LOCATION.

- When you use a predicate expression, explicit values must be on the right side of the comparison operator, or queries might fail.
  - Correct: SELECT * FROM mytable WHERE time > 11
  - Incorrect: SELECT * FROM mytable WHERE 11 > time
- We do not recommend using user-defined functions (UDFs) in predicate expressions. Queries may fail because of the way Hive tries to optimize query execution.
- Temporary tables are not supported.

Using an External MySQL Database or Amazon Aurora

To use an external MySQL database or Amazon Aurora as your Hive metastore, you override the default configuration values for the metastore in Hive to specify the external database location, either on an Amazon RDS MySQL instance or an Amazon Aurora instance.

**Note**

Hive neither supports nor prevents concurrent write access to metastore tables. If you share metastore information between two clusters, you must ensure that you do not write to the same metastore table concurrently, unless you are writing to different partitions of the same metastore table.

The following procedure shows you how to override the default configuration values for the Hive metastore location and start a cluster using the reconfigured metastore location.

To create a metastore located outside of the EMR cluster

1. Create a MySQL or Aurora database.
   
   For information about how to create an Amazon RDS database, see [https://aws.amazon.com/rds/](https://aws.amazon.com/rds/).

2. Modify your security groups to allow JDBC connections between your database and the ElasticMapReduce-Master security group.
   
   For information about how to modify your security groups for access, see [https://aws.amazon.com/rds/faqs/#security](https://aws.amazon.com/rds/faqs/#security).

3. Set JDBC configuration values in `hive-site.xml`:

   - **Important**
     If you supply sensitive information, such as passwords, to the Amazon EMR configuration API, this information is displayed for those accounts that have sufficient permissions. If you are concerned that this information could be displayed to other users, create the cluster with an administrative account and limit other users (IAM users or those with delegated credentials) to accessing services on the cluster by creating a role which explicitly denies permissions to the elasticmapreduce:DescribeCluster API key.

   Create a configuration file called `hiveConfiguration.json` containing edits to `hive-site.xml`:

   ```json
   [
     {
       "Classification": "hive-site",
   ```
"Properties": {
  "javax.jdo.option.ConnectionURL": "jdbc:mysql:\/\slashhostname:3306\slashhive?createDatabaseIfNotExist=true",
  "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
  "javax.jdo.option.ConnectionUserName": "username",
  "javax.jdo.option.ConnectionPassword": "password"
}
]

**Note**
For Amazon EMR versions 4.0.0 or below, the driver used is org.mysql.jdbc.Driver for `javax.jdo.option.ConnectionDriverName`.

Use `hiveConfiguration.json` with the following AWS CLI command to create the cluster:

```
aws emr create-cluster --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 2 \ --applications Name=Hive --configurations ./hiveConfiguration.json --use-default-roles
```

**Note**
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

**Note**
If you plan to store your configuration in Amazon S3, you must specify the URL location of the object. For example:

```
aws emr create-cluster --release-label emr-5.11.0 --instance-type m3.xlarge --instance-count 3 \ --applications Name=Hive --configurations https://s3.amazonaws.com/mybucket/myfolder/hiveConfiguration.json --use-default-roles
```

`hostname` is the DNS address of the Amazon RDS instance running the database. `<username>` and `<password>` are the credentials for your database. For more information about connecting to MySQL and Aurora database instances, see [Connecting to a DB Instance Running the MySQL Database Engine](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RDS-MySQL.Connecting.html) and [Connecting to an Athena DB Cluster in the Amazon Relational Database Service User Guide](https://docs.aws.amazon.com/AmazonRDS/latest/RDS-AthenaUserGuide/CreatingAnAthenaDBCluster.html). `javax.jdo.option.ConnectionURL` is the JDBC connect string for a JDBC metastore. `javax.jdo.option.ConnectionDriverName` is the driver class name for a JDBC metastore.

The MySQL JDBC drivers are installed by Amazon EMR.

**Note**
The value property should not contain any spaces or carriage returns. It should appear all on one line.

4. Connect to the master node of your cluster.

Instructions on how to connect to the master node are available at [Connect to the Master Node Using SSH in the Amazon EMR Management Guide](https://docs.aws.amazon.com/emr/latest/ug/ssh-commands.html).

5. Create your Hive tables specifying the location on Amazon S3 by entering a command similar to the following:

```
CREATE EXTERNAL TABLE IF NOT EXISTS table_name
(
  key int,
  value int
)  
```
6. Add your Hive script to the running cluster.

Your Hive cluster runs using the metastore located in Amazon RDS. Launch all additional Hive clusters that share this metastore by specifying the metastore location.

Use the Hive JDBC Driver

You can use popular business intelligence tools like Microsoft Excel, MicroStrategy, QlikView, and Tableau with Amazon EMR to explore and visualize your data. Many of these tools require Java Database Connectivity (JDBC) driver or an Open Database Connectivity (ODBC) driver. Amazon EMR supports both JDBC and ODBC connectivity.

The example below demonstrates using SQL Workbench/J as a SQL client to connect to a Hive cluster in Amazon EMR. For additional drivers, see Use Business Intelligence Tools with Amazon EMR.

Before you install and work with SQL Workbench/J, download the driver package and install the driver. The drivers included in the package support the Hive versions available in Amazon EMR release versions 4.0 and later. For detailed release notes and documentation, see the PDF documentation included in the package.

- **JDBC Driver Package Download**

  http://awssupportdatasvcs.com/bootstrap-actions/Simba/AmazonHiveJDBC-1.0.9.1060.zip

**To install and configure SQL Workbench**


3. - Linux, Unix, Mac OS X users: In a terminal session, create an SSH tunnel to the master node of your cluster using the following command. Replace `master-public-dns-name` with the public DNS name of the master node and `path-to-key-file` with the location and file name of your Amazon EC2 private key (.pem) file.

   ```
   ssh -o ServerAliveInterval=10 -i path-to-key-file -N -L 10000:localhost:10000 hadoop@master-public-dns-name
   ```

   - Windows users: In a PuTTY session, create an SSH tunnel to the master node of your cluster (using local port forwarding) with the following settings. Replace `master-public-dns-name` with the public DNS name of the master node. For more information about creating an SSH tunnel to the master node, see Option 1: Set Up an SSH Tunnel to the Master Node Using Local Port Forwarding in the Amazon EMR Management Guide.

   ```
   Source port: 10000 Destination: master-public-dns-name:10000
   ```
4. Add the JDBC driver to SQL Workbench.
   
   a. In the **Select Connection Profile** dialog box, click **Manage Drivers**.
   
   b. Click the **Create a new entry** (blank page) icon.
   
   c. In the **Name** field, type **Hive JDBC**.
   
   d. For **Library**, click the **Select the JAR file(s)** icon.
   
   e. Browse to the location containing the extracted drivers, select the following JAR files and click **Open**.

   ```
   hive_metastore.jar
   hive_service.jar
   HiveJDBC41.jar
   libfb303-0.9.0.jar
   libthrift-0.9.0.jar
   log4j-1.2.14.jar
   ql.jar
   slf4j-api-1.5.11.jar
   slf4j-log4j12-1.5.11.jar
   TCLIServiceClient.jar
   zookeeper-3.4.6.jar
   ```

   f. In the **Please select one driver** dialog box, select the following driver and click **OK**.

   ```
   com.amazon.hive.jdbc41.HS2Driver
   ```

5. When you return to the **Manage Drivers** dialog box, verify that the **Classname** field is populated and click **OK**.

6. When you return to the **Select Connection Profile** dialog box, verify that the **Driver** field is set to **Hive JDBC** and provide the JDBC connection string in the **URL** field.

   `jdbc:hive2://localhost:10000/default`

7. Click **OK** to connect. After the connection is complete, connection details appear at the top of the SQL Workbench/J window.

For more information about using Hive and the JDBC interface, see **HiveClient** and **HiveJDBCInterface** in Apache Hive documentation.
Hue

Hue (Hadoop User Experience) is an open-source, web-based, graphical user interface for use with Amazon EMR and Apache Hadoop. Hue groups together several different Hadoop ecosystem projects into a configurable interface for your Amazon EMR cluster. Amazon has also added customizations specific to Hue in Amazon EMR releases. Launch your cluster using the Amazon EMR console and you can interact with Hadoop and related components on your cluster using Hue. For more information about Hue, see http://gethue.com.

Cluster administrators usually use the AWS Management Console or AWS CLI to launch and administer clusters, including a cluster with Hue installed. After a cluster launches, you might interact entirely with your Amazon EMR cluster using an application such as Hue. Hue acts as a front end for the applications, allowing you to interact with clusters using an interface that may be more familiar or user-friendly. The applications in Hue, such as the Hive and Pig editors, replace the need to log in to the cluster to run scripts interactively using each application's respective shell.

Supported and unsupported features of Hue on Amazon EMR

- Amazon S3 and Hadoop File System (HDFS) Browser
  - With the appropriate permissions, you can browse and move data between the ephemeral HDFS storage and S3 buckets belonging to your account.
  - By default, superusers in Hue can access all files that Amazon EMR IAM roles are allowed to access. Newly created users do not automatically have permissions to access the Amazon S3 filebrowser and must have the `filebrowser.s3_access` permissions enabled for their group.
- Hive—Run interactive queries on your data. This is also a useful way to prototype programmatic or batched querying.
- Pig—Run scripts on your data or issue interactive commands.
- Oozie—Create and monitor Oozie workflows.
- Metastore Manager—View and manipulate the contents of the Hive metastore (import/create, drop, and so on).
- Job browser—See the status of your submitted Hadoop jobs.
- User management—Manage Hue user accounts and integrate LDAP users with Hue.
- AWS Samples—There are several "ready-to-run" examples that process sample data from various AWS services using applications in Hue. When you log in to Hue, you are taken to the Hue Home application where the samples are pre-installed.
- Livy Server is supported only in Amazon EMR version 5.9.0 and later.
- To use the Hue Notebook for Spark, you must install Hue with Livy and Spark.
- The Hue Dashboard is not supported.

Hue Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label `emr-5.11.1`). This component information applies to both `emr-5.11.1` and `emr-5.11.0`.

119
Create a Cluster with Hue Installed

To launch a cluster with Hue installed using the console

1. Choose Go to Advanced Options.
2. Navigate to Software Configuration and choose Amazon for Hadoop distribution and 4.1.0 or later for the Release label.
3. In Software Configuration > Applications to be installed, Hue should appear in the list by default.
4. In the Hardware Configuration section, accept the default EC2 instance types: m3.xlarge for instance types. You can change the instance types to suit your needs. If you will have more than 20 concurrent users accessing Hue, we recommend an instance type of m3.2xlarge or greater for the master node. We also recommend that you have a minimum of two core nodes for clusters running Hue.
5. In Security and Access, select a key pair for connecting to your cluster. You will need to use a key pair to open an SSH tunnel to connect to the Hue Web interface on the master node.
6. Click Create cluster.

To launch a cluster with Hue installed using the AWS CLI

To launch a cluster with Hue installed using the AWS CLI, type the create-cluster subcommand with the --applications parameter.

**Note**
You will need to install the current version of the AWS CLI. To download the latest release, see https://aws.amazon.com/cli/.

1. If you have not previously created the default EMR role and EC2 instance profile, type the following command to create them. Alternatively, you can specify your own roles. For more information on using your own roles, see http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-iam-roles.html.

   ```bash
   aws emr create-default-roles
   ```

2. To launch an Amazon EMR cluster with Hue installed using the default roles, type the following command and replace myKey with the name of your EC2 key pair.

   ```bash
   aws emr create-cluster --name "Hue cluster" --release-label emr-5.11.0
   --applications Name=Hue Name=Hive Name=Pig --use-default-roles --ec2-attributes
   KeyName=myKey --instance-type m3.xlarge --instance-count 3
   ```
Launch the Hue Web Interface

Launching Hue is the same as connecting to any HTTP interface hosted on the master node of a cluster. The following procedure describes how to access the Hue interface. For more information on accessing web interfaces hosted on the master node, see: http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-web-interfaces.html.

To launch the Hue web interface

1. Follow these instructions to create an SSH tunnel to the master node and to configure an HTTP proxy add-in for your browser: http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-ssh-tunnel.html.
2. Type the following address in your browser to open the Hue web interface: http://master public DNS:8888.
3. At the Hue login screen, if you are the administrator logging in for the first time, enter a username and password to create your Hue superuser account and then click Create account. Otherwise, type your username and password and click Create account or enter the credentials provided by your administrator.

Use Hue with a Remote Database in Amazon RDS

By default, Hue user information and query histories are stored in a local MySQL database on the master node. However, you can create one or more Hue-enabled clusters using a configuration stored in Amazon S3 and a MySQL database in Amazon RDS. This allows you to persist user information and query history created by Hue without keeping your Amazon EMR cluster running. We recommend using Amazon S3 server-side encryption to store the configuration file.

First create the remote database for Hue.

To create the external MySQL database

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Click Launch a DB Instance.
3. Choose MySQL and click Select.
4. Leave the default selection of Multi-AZ Deployment and Provisioned IOPS Storage and click Next.
5. Leave the Instance Specifications at their defaults, specify Settings, and click Next.
6. On the Configure Advanced Settings page, choose a proper security group and database name. The security group you use must at least allow ingress TCP access for port 3306 from the master node of your cluster. If you have not created your cluster at this point, you can allow all hosts to connect.
to port 3306 and adjust the security group after you have launched the cluster. Click **Launch DB Instance**.

7. From the RDS Dashboard, click on **Instances** and select the instance you have just created. When your database is available, you can open a text editor and copy the following information: dbname, username, password, and RDS instance hostname. You will use information when you create and configure your cluster.

**To specify an external MySQL database for Hue when launching a cluster using the AWS CLI**

To specify an external MySQL database for Hue when launching a cluster using the AWS CLI, use the information you noted when creating your RDS instance for configuring `hue.ini` with a configuration object

**Note**

You can create multiple clusters that use the same external database, but each cluster will share query history and user information.

- Create a cluster with Hue installed, using the external database you created:

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Hue Name=Spark Name=Hive \\
--instance-type m3.xlarge --instance-count 2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

**Note**

Linux line continuation characters (`\`) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (`^`).

`myConfig.json`:

```json
{
    "Classification": "hue-ini",
    "Properties": {},
    "Configurations": [
        {
            "Classification": "desktop",
            "Properties": {},
            "Configurations": [
                {
                    "Classification": "database",
                    "Properties": {
                        "name": "dbname",
                        "user": "username",
                        "password": "password",
                        "host": "hueinstance.c3b8apyyzi.us-east-1.rds.amazonaws.com",
                        "port": "3306",
                        "engine": "mysql"
                    },
                    "Configurations": []
                }
            ]
        }
    ]
}
```

**Note**

If you have not previously created the default EMR service role and EC2 instance profile, type `aws emr create-default-roles` to create them before typing the `create-cluster` subcommand.
For more information on using Amazon EMR commands in the AWS CLI, see [http://docs.aws.amazon.com/cli/latest/reference/emr](http://docs.aws.amazon.com/cli/latest/reference/emr).

**Troubleshooting**

**In the event of Amazon RDS failover**

It is possible users may encounter delays when running a query because the Hue database instance is non-responsive or is in the process of failover. The following are some facts and guidelines for this issue:

- If you login to the Amazon RDS console, you can search for failover events. For example, to see if a failover is in process or has occurred, look for events such as "Multi-AZ instance failover started" and "Multi-AZ instance failover completed."
- It takes about 30 seconds for an RDS instance to complete a failover.
- If you are experiencing longer-than-normal responses for queries in Hue, try to re-execute the query.

**Advanced Configurations for Hue**

This section includes the following topics.

**Topics**
- Configure Hue for LDAP Users (p. 123)

**Configure Hue for LDAP Users**

Integration with LDAP allows users to log into Hue using existing credentials stored in an LDAP directory. When you integrate Hue with LDAP, you do not need to independently manage user information in Hue. The information below demonstrates Hue integration with Microsoft Active Directory, but the configuration options are analogous to any LDAP directory.

LDAP authentication first binds to the server and establishes the connection. Then, the established connection is used for any subsequent queries to search for LDAP user information. Unless your Active Directory server allows anonymous connections, a connection needs to be established using a bind distinguished name and password. The bind distinguished name (or DN) is defined by the `bind_dn` configuration setting. The bind password is defined by the `bind_password` configuration setting. Hue has two ways to bind LDAP requests: search bind and direct bind. The preferred method for using Hue with Amazon EMR is search bind.

When search bind is used with Active Directory, Hue uses the user name attribute (defined by `user_name_attr config`) to find the attribute that needs to be retrieved from the base distinguished name (or DN). Search bind is useful when the full DN is not known for the Hue user.

For example, you may have `user_name_attr config` set to use the common name (or CN). In that case, the Active Directory server uses the Hue username provided during login to search the directory tree for a common name that matches, starting at the base distinguished name. If the common name for the Hue user is found, the user's distinguished name is returned by the server. Hue then constructs a distinguished name used to authenticate the user by performing a bind operation.

**Note**

Search bind searches usernames in all directory subtrees beginning at the base distinguished name. The base distinguished name specified in the Hue LDAP configuration should be the closest parent of the username, or your LDAP authentication performance may suffer.

When direct bind is used with Active Directory, the exact `nt_domain` or `ldap_username_pattern` must be used to authenticate. When direct bind is used, if the `nt_domain` (defined by the `nt_domain` configuration setting) attribute is defined, a user distinguished name template is created using the form:
Configure Hue for LDAP Users

To Launch an Amazon EMR cluster with LDAP properties for Hue using AWS CLI

- To specify LDAP properties for hue-ini, create a cluster with Hue installed using the following commands with AWS CLI:

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Hue Name=Spark Name=Hive 
--instance-type m3.xlarge --instance-count 2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

myConfig.json:

```json
[
  {
    "Classification": "hue-ini",
    "Properties": {},
    "Configurations": [
      {
        "Classification": "desktop",
        "Properties": {},
        "Configurations": [
          {
            "Classification": "ldap",
            "Properties": {},
            "Configurations": [
              {
                "Classification": "ldap_servers",
                "Properties": {},
                "Configurations": [
                  {
                    "Classification": "yourcompany",
                    "Properties": {
                      "base_dn": "DC=yourcompany,DC=hue,DC=com",
                      "ldap_url": "ldap://ldapurl",
                      "search_bind_authentication": "true",
                      "bind_dn": "CN=hue,CN=users,DC=yourcompany,DC=hue,DC=com",
                      "bind_password": "password"
                    },
                    "Configurations": []
                  }
                ],
                "Configurations": []
              }
            ],
            "Configurations": []
          }
        ],
        "Configurations": []
      }
    ],
    "Configurations": []
  }
]
```

To View LDAP Settings in Hue

1. Verify you have an active VPN connection or SSH tunnel to the Amazon EMR cluster’s master node. Then, in your browser, type `master-public-dns:8888` to open the Hue web interface.
2. Log in using your Hue administrator credentials. If the Did you know? window opens, click Got it, prof! to close it.
3. Click the Hue icon in the toolbar.
4. On the About Hue page, click Configuration.
5. In the Configuration Sections and Variables section, click Desktop.
6. Scroll to the ldap section to view your settings.

Metastore Manager Restrictions

The Metastore Manager default database exists in HDFS. You cannot import a table from Amazon S3 using a database stored in HDFS. To import a table from Amazon S3, create a new database, change the default location of the database to Amazon S3, create a table in the database, and then import your table from Amazon S3.
Apache Livy

Livy is available on Amazon EMR version 5.9.0 and later. Livy is a service that enables interaction over a REST interface with an EMR cluster running Spark. You can use the REST interface or an RPC client library to submit Spark jobs or snippets of Spark code, retrieve results synchronously or asynchronously, and manage Spark Context. For more information, see the Apache Livy website.

Livy Release Information

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Livy 0.4.0</td>
<td>emr-5.11.0</td>
<td>aws-hm-client, aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resource manager, hadoop-yarn-timeline-server, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave, livy-server</td>
</tr>
</tbody>
</table>
Apache Mahout

Amazon EMR supports Apache Mahout, a machine learning framework for Apache Hadoop. For more information about Mahout, go to [http://mahout.apache.org/](http://mahout.apache.org/).

Mahout is a machine learning library with tools for clustering, classification, and several types of recommenders, including tools to calculate most-similar items or build item recommendations for users. Mahout employs the Hadoop framework to distribute calculations across a cluster, and now includes additional work distribution methods, including Spark.

For more information and an example of how to use Mahout with Amazon EMR, see the [Building a Recommender with Apache Mahout on Amazon EMR](https://aws.amazon.com/blogs/big-data/) post on the AWS Big Data blog.

**Note**
Only Mahout version 0.13.0 and later are compatible with Spark version 2.x in Amazon EMR version 5.0 and later.

Mahout Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label `emr-5.11.1`). This component information applies to both `emr-5.11.1` and `emr-5.11.0`.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>
Apache MXNet

Apache MXNet, available in Amazon EMR release version 5.10.0 and later, is an acceleration library designed for building neural networks and other deep learning applications. MXNet automates common work flows and optimizes numerical computations. Using it helps you design neural network architectures without having to focus on implementing low-level computations, such as linear algebra operations. For more information, see the Apache MXNet web site.

MXNet Release Information

We strongly recommend Amazon EMR release 5.11.1 (release label `emr-5.11.1`). This component information applies to both `emr-5.11.1` and `emr-5.11.0`.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>
Apache Oozie

Use the Apache Oozie Workflow Scheduler to manage and coordinate Hadoop jobs. For more information about Apache Oozie, see the http://oozie.apache.org/.

Important
The Oozie native web interface is not supported on Amazon EMR. To use a front-end interface for Oozie, try the Hue Oozie application. For more information, see Hue (p. 119).

Oozie Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>
Apache Phoenix

Apache Phoenix is used for OLTP and operational analytics, allowing you to use standard SQL queries and JDBC APIs to work with an Apache HBase backing store. For more information, see Phoenix in 15 minutes or less.

Note
If you upgrade from an earlier version of Amazon EMR to Amazon EMR version 5.4.0 or later and use secondary indexing, upgrade local indexes as described in the Apache Phoenix documentation. Amazon EMR removes the required configurations from the hbase-site classification, but indexes need to be repopulated. Online and offline upgrade of indexes are supported. Online upgrades are the default, which means indexes are repopulated while initializing from Phoenix clients of version 4.8.0 or greater. To specify offline upgrades, set the phoenix.client.localIndexUpgrade configuration to false in the phoenix-site classification, and then SSH to the master node to run psql [zookeeper] -l.

Phoenix Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix 4.11.0</td>
<td>emr-5.11.0</td>
<td>emrfs, emr-ddb, emr-goodies,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>emr-kinesis, emr-s3-dist-cp,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-client, hadoop-hdfs-datanode,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-hdfs-library, hadoop-hdfs-namenode,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-httpfs-server, hadoop-kms-server,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-mapred, hadoop-yarn-nodemanager,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-yarn-resourcemanager,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hadoop-yarn-timeline-server,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hbase-hmaster, hbase-client,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hbase-region-server, phoenix-library,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>phoenix-query-server, zookeeper-client,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>zookeeper-server</td>
</tr>
</tbody>
</table>

Creating a Cluster with Phoenix

You install Phoenix by choosing the application when you create a cluster in the console or using the AWS CLI. The following procedures and examples show how to create a cluster with Phoenix and HBase. For more information about creating clusters using the console, including Advanced Options see Plan and Configure Clusters in the Amazon EMR Management Guide.
To launch a cluster with Phoenix installed using Quick Options for creating a cluster in the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster to use Quick Create.
3. For Software Configuration, choose the most recent release appropriate for your application. Phoenix appears as an option only when Amazon Release Version `emr-4.7.0` or later is selected.
4. For Applications, choose the second option, HBase: HBase ver, Hadoop ver, Hive ver, Hue ver, Phoenix ver, and ZooKeeper ver.
5. Select other options as necessary and then choose Create cluster.

**Note**
Linux line continuation characters (`\`) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (`^`).

The following example launches a cluster with Phoenix installed using default configuration settings.

To launch a cluster with Phoenix and HBase using the AWS CLI

- Create the cluster with the following command:

```bash
aws emr create-cluster --name "Cluster with Phoenix" --release-label emr-5.11.0 \
--applications Name=Phoenix Name=HBase --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Customizing Phoenix Configurations When Creating a Cluster

When creating a cluster, you configure Phoenix by setting values in `hbase-site.xml` using the `hbase-site` configuration classification.

For more information, see Configuration and Tuning in the Phoenix documentation.

The following example demonstrates using a JSON file stored in Amazon S3 to specify the value of `false` for the `phoenix.schema.dropMetaData` property. Multiple properties can be specified for a single classification. For more information, see Configuring Applications (p. 30). The create cluster command then references the JSON file as the --configurations parameter.

The contents of the JSON file saved to `/mybucket/myfolder/myconfig.json` is the following.

```json
[
  {
    "Classification": "hbase-site",
    "Properties": {
      "phoenix.schema.dropMetaData": "false"
    }
  }
]
```

The create_cluster command that references the JSON file is shown in the following example.

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Phoenix \
```

Phoenix Clients

You connect to Phoenix using either a JDBC client built with full dependencies or using the "thin client" that uses the Phoenix Query Server and can only be run on a master node of a cluster (e.g. by using an SQL client, a step, command line, SSH port forwarding, etc.). When using the "fat" JDBC client, it still needs to have access to all nodes of the cluster because it connects to HBase services directly. The "thin" Phoenix client only needs access to the Phoenix Query Server at a default port 8765. There are several scripts within Phoenix that use these clients.

Use an Amazon EMR step to query using Phoenix

The following procedure restores a snapshot from HBase and uses that data to run a Phoenix query. You can extend this example or create a new script that leverages Phoenix's clients to suit your needs.

1. Create a cluster with Phoenix installed, using the following command:

```bash
aws emr create-cluster --name "Cluster with Phoenix" --log-uri s3://myBucket/myLogFolder --release-label emr-5.11.0 --applications Name=Phoenix Name=HBase --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 --use-default-roles
```

2. Create then upload the following files to Amazon S3:

```bash
copySnapshot.sh
sudo su hbase -s /bin/sh -c 'hbase snapshot export
-D hbase.rootdir=s3://us-east-1.elasticmapreduce.samples/hbase-demo-customer-data/
snapshot/
-snapshot customer_snapshot1 
-copy-to hdfs://masterDNSName:8020/user/hbase 
-mappers 2 -chuser hbase -chmod 700''
```

```bash
runQuery.sh
aws s3 cp s3://myBucket/phoenixQuery.sql /home/hadoop/
/usr/lib/phoenix/bin/sqlline-thin.py http://localhost:8765 /home/hadoop/
phoenixQuery.sql
```

```sql
phoenixQuery.sql
CREATE VIEW "customer" (pk VARCHAR PRIMARY KEY, "address"."state" VARCHAR, "address"."street" VARCHAR, "address"."city" VARCHAR, "address"."zip" VARCHAR, "cc"."number" VARCHAR, "cc"."expire" VARCHAR, "cc"."type" VARCHAR, "contact"."phone" VARCHAR);
CREATE INDEX my_index ON "customer" ("customer"."state") INCLUDE("PK", "customer"."city", "customer"."expire", "customer"."type");
```
SELECT "customer"."type" AS credit_card_type, count(*) AS num_customers FROM "customer"
WHERE "customer"."state" = 'CA' GROUP BY "customer"."type";

Use the AWS CLI to submit the files to the S3 bucket:

```bash
aws s3 cp copySnapshot.sh s3://myBucket/
aws s3 cp runQuery.sh s3://myBucket/
aws s3 cp phoenixQuery.sql s3://myBucket/
```

3. Create a table using the following step submitted to the cluster that you created in Step 1:

```json
[  
  {   
    "Name": "Create HBase Table",   
    "Args": ["bash", "-c", "echo #create "customer","address","cc","contact" | hbase shell"],   
    "Jar": "command-runner.jar",   
    "ActionOnFailure": "CONTINUE",   
    "Type": "CUSTOM_JAR"   
  }  
]
```

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF 
--steps file://./createTable.json
```

4. Use script-runner.jar to run the copySnapshot.sh script that you previously uploaded to your S3 bucket:

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF 
--steps Type=CUSTOM_JAR,Name="HBase Copy Snapshot",ActionOnFailure=CONTINUE, 
Jar=s3://region.elasticmapreduce/libs/script-runner/script-runner.jar,Args=["s3://myBucket/copySnapshot.sh"]
```

This runs a MapReduce job to copy your snapshot data to the cluster HDFS.

5. Restore the snapshot that you copied to the cluster using the following step:

```json
[  
  {   
    "Name": "restore",   
    "Args": ["bash", "-c", "echo #disable "customer"; restore_snapshot 
"customer_snapshot1"; enable "customer" | hbase shell"],   
    "Jar": "command-runner.jar",   
    "ActionOnFailure": "CONTINUE",   
    "Type": "CUSTOM_JAR"   
  }  
]
```

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF 
--steps file://./restoreSnapshot.json
```

6. Use script-runner.jar to run the runQuery.sh script that you previously uploaded to your S3 bucket:

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF 
```
Phoenix Clients

```bash
--steps Type=CUSTOM_JAR,Name="Phoenix Run Query",ActionOnFailure=CONTINUE,\ 
    Jar=s3://region.elasticmapreduce/libs/script-runner/script-runner.jar,Args=["s3://myBucket/runQuery.sh"]
```

The query runs and returns the results to the step's stdout. It may take a few minutes for this step to complete.

7. Inspect the results of the step's stdout at the log URI that you used when you created the cluster in Step 1. The results should look like the following:

<table>
<thead>
<tr>
<th>CREDIT_CARD_TYPE</th>
<th>NUM_CUSTOMERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>american_express</td>
<td>5728</td>
</tr>
<tr>
<td>dankort</td>
<td>5782</td>
</tr>
<tr>
<td>diners_club</td>
<td>5795</td>
</tr>
<tr>
<td>discover</td>
<td>5715</td>
</tr>
<tr>
<td>forbrugsforeningen</td>
<td>5691</td>
</tr>
<tr>
<td>jcb</td>
<td>5762</td>
</tr>
<tr>
<td>laser</td>
<td>5769</td>
</tr>
<tr>
<td>maestro</td>
<td>5816</td>
</tr>
<tr>
<td>mastercard</td>
<td>5697</td>
</tr>
<tr>
<td>solo</td>
<td>5586</td>
</tr>
<tr>
<td>switch</td>
<td>5781</td>
</tr>
<tr>
<td>visa</td>
<td>5659</td>
</tr>
</tbody>
</table>
Apache Pig

Amazon EMR supports Apache Pig, a programming framework you can use to analyze and transform large data sets. For more information about Pig, go to http://pig.apache.org/.

Pig is an open-source, Apache library that runs on top of Hadoop. The library takes SQL-like commands written in a language called Pig Latin and converts those commands into Tez jobs based on directed acyclic graphs (DAGs) or MapReduce programs. You do not have to write complex code using a lower level computer language, such as Java.

You can execute Pig commands interactively or in batch mode. To use Pig interactively, create an SSH connection to the master node and submit commands using the Grunt shell. To use Pig in batch mode, write your Pig scripts, upload them to Amazon S3, and submit them as cluster steps. For more information on submitting work to a cluster, see Submit Work to a Cluster in the Amazon EMR Management Guide.

Note

When using an Apache Pig script to write output to an Hcatalog table in Amazon S3, disable Amazon EMR direct write within your Pig script using the `SET mapred.output.direct.NativeS3FileSystem false` and `SET mapred.output.direct.EmrFileSystem false` commands. For more information, see Using HCatalog (p. 105).

Pig Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pig 0.17.0</td>
<td>emr-5.11.0</td>
<td>emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resource-manager, hadoop-yarn-timeline-server, pig-client, tez-on-yarn</td>
</tr>
</tbody>
</table>

Submit Pig Work

This section demonstrates submitting Pig work to an Amazon EMR cluster. The examples that follow generate a report containing the total bytes transferred, a list of the top 50 IP addresses, a list of the top 50 external referrers, and the top 50 search terms using Bing and Google. The Pig script is located in the
Amazon S3 bucket `s3://elasticmapreduce/samples/pig-apache/do-reports2.pig`. Input data is located in the Amazon S3 bucket `s3://elasticmapreduce/samples/pig-apache/input`. The output is saved to an Amazon S3 bucket.

**Important**
For EMR 4.x or greater, you must copy and modify the Pig script do-reports.pig to make it work. In your modified script, replace the following line

```python
register file:/home/hadoop/lib/pig/piggybank.jar
```

with this:

```python
register file:/usr/lib/pig/lib/piggybank.jar
```

Then replace this script in your own bucket in Amazon S3.

## Submit Pig Work Using the Amazon EMR Console

This example describes how to use the Amazon EMR console to add a Pig step to a cluster.

### To submit a Pig step

1. Open the Amazon EMR console at [https://console.aws.amazon.com/elasticmapreduce/](https://console.aws.amazon.com/elasticmapreduce/).
2. In the **Cluster List**, select the name of your cluster.
3. Scroll to the **Steps** section and expand it, then choose **Add step**.
4. In the **Add Step** dialog:
   - For **Step type**, choose **Pig program**.
   - For **Name**, accept the default name (Pig program) or type a new name.
   - For **Script S3 location**, type the location of the Pig script. For example: `s3://elasticmapreduce/samples/pig-apache/do-reports2.pig`.
   - For **Input S3 location**, type the location of the input data. For example: `s3://elasticmapreduce/samples/pig-apache/input`.
   - For **Output S3 location**, type or browse to the name of your Amazon S3 output bucket.
   - For **Arguments**, leave the field blank.
   - For **Action on failure**, accept the default option (**Continue**).
5. Choose **Add**. The step appears in the console with a status of Pending.
6. The status of the step changes from Pending to Running to Completed as the step runs. To update the status, choose the **Refresh** icon above the **Actions** column.

## Submit Pig Work Using the AWS CLI

### To submit a Pig step using the AWS CLI

When you launch a cluster using the AWS CLI, use the `--applications` parameter to install Pig. To submit a Pig step, use the `--steps` parameter.

- To launch a cluster with Pig installed and to submit a Pig step, type the following command, replace `myKey` with the name of your EC2 key pair, and replace `mybucket` with the name of your Amazon S3 bucket.

```
aws emr create-cluster --name "Test cluster" --release-label emr-5.11.0 --applications Name=Pig \
```
Call User Defined Functions from Pig

Pig provides the ability to call user-defined functions (UDFs) from within Pig scripts. You can do this to implement custom processing to use in your Pig scripts. The languages currently supported are Java, Python/Jython, and JavaScript (though JavaScript support is still experimental.)

The following sections describe how to register your functions with Pig so you can call them either from the Pig shell or from within Pig scripts. For more information about using UDFs with Pig, see for your version of Pig.

Call JAR files from Pig

You can use custom JAR files with Pig using the `REGISTER` command in your Pig script. The JAR file is local or a remote file system such as Amazon S3. When the Pig script runs, Amazon EMR downloads the JAR file automatically to the master node and then uploads the JAR file to the Hadoop distributed cache. In this way, the JAR file is automatically used as necessary by all instances in the cluster.

To use JAR files with Pig

1. Upload your custom JAR file into Amazon S3.
2. Use the `REGISTER` command in your Pig script to specify the bucket on Amazon S3 of the custom JAR file.

```
REGISTER s3://mybucket/path/mycustomjar.jar;
```

Call Python/Jython Scripts from Pig

You can register Python scripts with Pig and then call functions in those scripts from the Pig shell or in a Pig script. You do this by specifying the location of the script with the `register` keyword.

Because Pig in written in Java, it uses the Jython script engine to parse Python scripts. For more information about Jython, go to [http://www.jython.org/](http://www.jython.org/).
To call a Python/Jython script from Pig

1. Write a Python script and upload the script to a location in Amazon S3. This should be a bucket owned by the same account that creates the Pig cluster, or that has permissions set so the account that created the cluster can access it. In this example, the script is uploaded to s3://mybucket/pig/python.

2. Start a Pig cluster. If you are accessing Pig from the Grunt shell, run an interactive cluster. If you are running Pig commands from a script, start a scripted Pig cluster. This example starts an interactive cluster. For more information about how to create a Pig cluster, see Submit Pig Work (p. 135).

3. For an interactive cluster, use SSH to connect into the master node and run the Grunt shell. For more information, see SSH into the Master Node.

4. Run the Grunt shell for Pig by typing `pig` at the command line:

   ```
pig
   ```

5. Register the Jython library and your Python script with Pig using the `register` keyword at the Grunt command prompt, as shown in the following command, where you would specify the location of your script in Amazon S3:

   ```
grunt> register 'lib/jython.jar';
grunt> register 's3://mybucket/pig/python/myscript.py' using jython as myfunctions;
   ```

6. Load the input data. The following example loads input from an Amazon S3 location:

   ```
grunt> input = load 's3://mybucket/input/data.txt' using TextLoader as (line:chararray);
   ```

7. You can now call functions in your script from within Pig by referencing them using `myfunctions`:

   ```
grunt> output=foreach input generate myfunctions.myfunction($1);
   ```
Presto

Use Presto as a fast SQL query engine for large data sources. For more information, see the Presto website.

Presto Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Limitations and Known Issues with Presto on Amazon EMR

- Certain Presto properties or properties that pertain to Presto cannot be configured directly with the configuration API. You can configure log.properties and config.properties. However, the following properties cannot be configured:
  - node.properties (configurable in Amazon EMR version 5.6.0 and later)
  - jvm.config

  For more information about these configuration files, see the Presto documentation.
- Presto is not configured to use EMRFS. Instead, it uses PrestoS3FileSystem.
- You can access the Presto web interface on the Presto coordinator using port 8889.

Using Presto with the AWS Glue Data Catalog

Using Amazon EMR release version 5.10.0 and later, you can specify the AWS Glue Data Catalog as the default Hive metastore for Presto. You can specify this option when you create a cluster using the AWS
Management Console, or using the `presto-connector-hive` configuration classification when using the AWS CLI or Amazon EMR API. For more information, see Configuring Applications (p. 30).

**To specify the AWS Glue Data Catalog as the default Hive metastore using the console**

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose **Create cluster**, **Go to advanced options**.
3. Under **Software Configuration** choose a **Release** of `emr-5.10-0` or later and select **Presto**.
4. Select **Use for Presto table metadata**, choose **Next**, and then complete other settings for your cluster as appropriate for your application.

**To specify the AWS Glue Data Catalog as the default Hive metastore using the CLI or API**

- Set the `hive.metastore.glue.datacatalog.enabled` property to `true`, as shown in the following JSON example.

```json
[
  {
    "Classification": "presto-connector-hive",
    "Properties": {
      "hive.metastore.glue.datacatalog.enabled": "true"
    }
  }
]
```

Optionally, you can manually set `hive.metastore.glue.datacatalog.enabled=true` in the `/etc/presto/conf/catalog/hive.properties` file on the master node. If you use this method, make sure that `hive.table-statistics-enabled=false` in the properties file is set because the Data Catalog does not support Hive table and partition statistics. If you change the value on a long-running cluster to switch metastores, you must restart the Presto server on the master node (`sudo restart presto-server`).

**Unsupported Configurations, Functions, and Known Issues**

The limitations listed below apply when using the AWS Glue Data Catalog as a metastore:

- Renaming tables from within AWS Glue is not supported.
- Partition values containing quotes and apostrophes are not supported (for example, `PARTITION (owner="Doe's")`).
- **Table and partition statistics** are not supported.
- Using **Hive authorization** is not supported.

**Adding Database Connectors**

You can add JDBC connectors at cluster launch using the configuration classifications. For more information about connectors, see https://prestodb.io/docs/current/connector.html.

These classifications are named as follows:

- `presto-connector-blackhole`
- `presto-connector-cassandra`
Using LDAP Authentication with Presto

Amazon EMR version 5.5.0 and later supports using Lightweight Directory Access Protocol (LDAP) authentication with Presto. To use LDAP, you must enable HTTPS access for the Presto coordinator (set http-server.https.enabled=true in config.properties on the master node). For configuration details, see LDAP Authentication in Presto documentation.

Enabling SSL/TLS for Internal Communication Between Nodes

With Amazon EMR version 5.6.0 and later, you can enable SSL/TLS secured communication between Presto nodes by using a security configuration to enable in-transit encryption. For more information, see Specifying Amazon EMR Encryption Options Using a Security Configuration. The default port for
internal HTTPS is 8446. The port used for internal communication must be the same port used for HTTPS access to the Presto coordinator. The `http-server.https.port=port_num` parameter in the Presto `config.properties` file specifies the port.

When in-transit encryption is enabled, Amazon EMR does the following for Presto:

- Distributes the artifacts you specify for in-transit encryption throughout the Presto cluster. For more information about encryption artifacts, see Providing Certificates for In-Transit Data Encryption.
- Modifies the `config.properties` file for Presto as follows:
  - Sets `http-server.http.enabled=false` on core and task nodes, which disables HTTP in favor of HTTPS.
  - Sets `http-server.https.*`, `internal-communication.https.*`, and other values to enable HTTPS and specify implementation details, including LDAP parameters if you have enabled and configured LDAP.
Apache Spark

This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

Apache Spark is a cluster framework and programming model that helps you do machine learning, stream processing, or graph analytics using Amazon EMR clusters. Similar to Apache Hadoop, Spark is an open-source, distributed processing system commonly used for big data workloads. However, Spark has several notable differences from Hadoop MapReduce. Spark has an optimized directed acyclic graph (DAG) execution engine and actively caches data in-memory, which can boost performance especially for certain algorithms and interactive queries.

Spark natively supports applications written in Scala, Python, and Java and includes several tightly integrated libraries for SQL (Spark SQL), machine learning (MLlib), stream processing (Spark Streaming), and graph processing (GraphX). These tools make it easier to leverage the Spark framework for a wide variety of use cases.

Spark can be installed alongside the other Hadoop applications available in Amazon EMR, and it can also leverage the EMR file system (EMRFS) to directly access data in Amazon S3. Hive is also integrated with Spark. So you can use a HiveContext object to run Hive scripts using Spark. A Hive context is included in the spark-shell as sqlContext.

To view an end-to-end example using Spark on Amazon EMR, see the New — Apache Spark on Amazon EMR post on the AWS official blog.

To view a machine learning example using Spark on Amazon EMR, see the Large-Scale Machine Learning with Spark on Amazon EMR post on the AWS Big Data blog.

Spark Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark 2.2.1</td>
<td>emr-5.11.0</td>
<td>aws-hm-client, aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, livy-server, spark-client, spark-</td>
</tr>
</tbody>
</table>
Create a Cluster With Spark

The following procedure creates a cluster with Spark installed using Quick Options in the EMR console. Use Advanced Options to further customize your cluster setup, and use Step execution mode to programmatically install applications and then execute custom applications that you submit as steps. With either of these advanced options, you can choose to use AWS Glue as your Spark SQL metastore. See Using the AWS Glue Data Catalog as the Metastore for Spark SQL (p. 145) for more information.

To launch a cluster with Spark installed

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster to use Quick Create.
3. For Software Configuration, choose Amazon Release Version emr-5.11.0 or later.
4. For Select Applications, choose either All Applications or Spark.
5. Select other options as necessary and then choose Create cluster.

Note
To configure Spark when you are creating the cluster, see Configure Spark (p. 147).

To launch a cluster with Spark installed using the AWS CLI

- Create the cluster with the following command:

```
aws emr create-cluster --name "Spark cluster" --release-label emr-5.11.0 --applications Name=Spark "
   --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

To launch a cluster with Spark installed using the SDK for Java

Specify Spark as an application with SupportedProductConfig used in RunJobFlowRequest.

- The following Java program excerpt shows how to create a cluster with Spark:

```java
AmazonElasticMapReduceClient emr = new AmazonElasticMapReduceClient( credentials );

Application sparkApp = new Application()
    .withName("Spark")
;Applications myApps = new Applications();
myApps.add(sparkApp);

RunJobFlowRequest request = new RunJobFlowRequest()
    .withName("Spark Cluster")
    .withApplications(myApps)
    .withReleaseLabel("emr-5.11.0")
```
Using the AWS Glue Data Catalog as the Metastore for Spark SQL

Using Amazon EMR version 5.8.0 or later, you can configure Spark SQL to use the AWS Glue Data Catalog as its metastore. We recommend this configuration when you require a persistent metastore or a metastore shared by different clusters, services, and applications.

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores. The AWS Glue Data Catalog provides a unified metadata repository across a variety of data sources and data formats, integrating with Amazon EMR as well as Amazon RDS, Amazon Redshift, Redshift Spectrum, Athena, and any application compatible with the Apache Hive metastore. AWS Glue crawlers can automatically infer schema from source data in Amazon S3 and store the associated metadata in the Data Catalog. For more information about the Data Catalog, see Populating the AWS Glue Data Catalog in the AWS Glue Developer Guide.

Separate charges apply for AWS Glue. There is a monthly rate for storing and accessing the metadata in the Data Catalog, an hourly rate billed per minute for AWS Glue ETL jobs and crawler runtime, and an hourly rate billed per minute for each provisioned development endpoint. The Data Catalog allows you to store up to a million objects at no charge. If you store more than a million objects, you are charged USD$1 for each 100,000 objects over a million. An object in the Data Catalog is a table, partition, or database. For more information, see Glue Pricing.

Important
If you created tables using Amazon Athena or Amazon Redshift Spectrum before August 14, 2017, databases and tables are stored in an Athena-managed catalog, which is separate from the AWS Glue Data Catalog. To integrate Amazon EMR with these tables, you must upgrade to the AWS Glue Data Catalog. For more information, see Upgrading to the AWS Glue Data Catalog in the Amazon Athena User Guide.

Specifying AWS Glue Data Catalog as the Metastore

You can specify the AWS Glue Data Catalog as the metastore using the AWS Management Console, AWS CLI, or Amazon EMR API. When you create a cluster using the CLI or API, you use the spark-hive-site configuration classification to specify the Data Catalog. When you create a cluster using the console, you can specify the Data Catalog using Advanced Options or Quick Options.

Note
The option to use AWS Glue Data Catalog is also available with Zeppelin because Zeppelin is installed with Spark SQL components.

To specify the AWS Glue Data Catalog as the metastore for Spark SQL using the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster, Go to advanced options.
3. For Release, choose emr-5.8.0 or later.
5. Under AWS Glue Data Catalog settings, select Use for Hive table metadata.
6. Choose other options for your cluster as appropriate, choose Next, and then configure other cluster options as appropriate for your application.

To specify the AWS Glue Data Catalog as the metastore using the AWS CLI or Amazon EMR API

- Specify the value for hive.metastore.client.factory.class using the spark-hive-site classification as shown in the following example. For more information, see Configuring Applications.

Example Example Configuration JSON for Using the AWS Glue Data Catalog

```
[
    {
        "Classification": "spark-hive-site",
        "Properties": {
            "hive.metastore.client.factory.class": "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
        }
    }
]
```

IAM Permissions

The EMR_EC2_DefaultRole must be allowed IAM permissions for AWS Glue actions. This is only a concern if you don’t use the default AmazonElasticMapReduceforEC2Role managed policy and you attach a customer-managed policy to the role. In this case, you need to configure the policy to allow permission to perform AWS Glue actions. Open the IAM console (https://console.aws.amazon.com/iam/) and view the contents of the AmazonElasticMapReduceforEC2Role managed policy to see the required AWS Glue actions to allow.

Unsupported Configurations, Functions, and Known Issues

The limitations listed below apply when using the AWS Glue Data Catalog as a metastore:

- Having a default database without a location URI causes failures when you create a table. As a workaround, use the LOCATION clause to specify a bucket location, such as s3://mybucket, when you use CREATE TABLE. Alternatively create tables within a database other than the default database.
- Renaming tables from within AWS Glue is not supported.
- Partition values containing quotes and apostrophes are not supported (for example, PARTITION (owner="Doe's").
- Table and partition statistics are not supported.
- Using Hive authorization is not supported.
- Hive constraints are not supported.
- Setting hive.metastore.partition.inherit.table.properties is not supported.
- Using the following metastore constants is not supported: BUCKET_COUNT, BUCKET_FIELD_NAME, DDL_TIME, FIELD_TO_DIMENSION, FILE_INPUT_FORMAT, FILE_OUTPUT_FORMAT, HIVE_FILTER_FIELD_LAST_ACCESS, HIVE_FILTER_FIELD_OWNER,
HIVE_FILTER_FIELD_PARAMS, IS_ARCHIVED, META_TABLE_COLUMNS, 
META_TABLE_COLUMN_TYPES, META_TABLE_DB, META_TABLE_LOCATION, META_TABLE_NAME, 
META_TABLE_PARTITION_COLUMNS, META_TABLE_SERDE, META_TABLE_STORAGE, 
ORIGINAL_LOCATION.

- When you use a predicate expression, explicit values must be on the right side of the comparison 
operator, or queries might fail.
  - Correct: SELECT * FROM mytable WHERE time > 11
  - Incorrect: SELECT * FROM mytable WHERE 11 > time
- We do not recommend using user-defined functions (UDFs) in predicate expressions. Queries may fail 
because of the way Hive tries to optimize query execution.
- Temporary tables are not supported.

Configure Spark

You can configure Spark on Amazon EMR using the spark-defaults configuration classification. 
For more information about the options, see the Spark Configuration topic in the Apache Spark 
documentation. It is also possible to configure Spark dynamically at the time of each application 
submission. For more information, see Enabling Dynamic Allocation of Executors (p. 148).

Topics
- Spark Defaults Set By Amazon EMR (p. 147)
- Enabling Dynamic Allocation of Executors (p. 148)
- Configuring Node Decommissioning Behavior (p. 149)
- Spark ThriftServer Environment Variable (p. 150)
- Changing Spark Default Settings (p. 150)

Spark Defaults Set By Amazon EMR

The following defaults are set by Amazon EMR regardless of whether other settings are set to true, such 
as spark.dynamicAllocation.enabled or maximizeResourceAllocation.

- spark.executor.memory
- spark.executor.cores

Note
In releases 4.4.0 or greater, spark.dynamicAllocation.enabled is set to true by default.

The following table shows how Spark defaults that affect applications are set.

Spark defaults set by Amazon EMR

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.executor.memory</td>
<td>Amount of memory to use per executor process. (for example, 1g, 2g)</td>
<td>Setting is configured based on the slave instance types in the cluster.</td>
</tr>
<tr>
<td>spark.executor.cores</td>
<td>The number of cores to use on each executor.</td>
<td>Setting is configured based on the slave instance types in the cluster.</td>
</tr>
</tbody>
</table>
Enabling Dynamic Allocation of Executors

You can configure your executors to utilize the maximum resources possible on each node in a cluster by enabling the `maximizeResourceAllocation` option when creating the cluster. This option calculates the maximum compute and memory resources available for an executor on a node in the core node group and sets the corresponding `spark-defaults` settings with this information.

### Spark defaults set when `maximizeResourceAllocation` is enabled

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spark.default.parallelism</code></td>
<td>Default number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize when not set by user.</td>
<td>2X number of CPU cores available to YARN containers.</td>
</tr>
<tr>
<td><code>spark.driver.memory</code></td>
<td>Amount of memory to use for the driver process, i.e. where SparkContext is initialized. (for example, 1g, 2g).</td>
<td>Setting is configured based on the instance types in the cluster. However, because the Spark driver application may run on either the master or one of the core instances (for example, in YARN client and cluster modes, respectively), this is set based on the smaller of the instance types in these two instance groups.</td>
</tr>
<tr>
<td><code>spark.executor.memory</code></td>
<td>Amount of memory to use per executor process. (for example, 1g, 2g)</td>
<td>Setting is configured based on the slave instance types in the cluster.</td>
</tr>
<tr>
<td><code>spark.executor.cores</code></td>
<td>The number of cores to use on each executor.</td>
<td>Setting is configured based on the slave instance types in the cluster.</td>
</tr>
<tr>
<td><code>spark.executor.instances</code></td>
<td>The number of executors.</td>
<td>Setting is configured based on the slave instance types in the cluster. Set unless <code>spark.dynamicAllocation.enabled</code> explicitly set to true at the same time.</td>
</tr>
</tbody>
</table>

### Enabling Dynamic Allocation of Executors

Spark on YARN has the ability to scale the number of executors used for a Spark application dynamically. In releases 4.4.0 or greater, this is the default behavior. To learn more about dynamic allocation, see the [Dynamic Allocation](#) topic in the Apache Spark documentation.
Configuring Node Decommissioning Behavior

When using Amazon EMR release version 5.9.0 or later, Spark on Amazon EMR includes a set of features to help ensure that Spark handles node termination because of a manual resize or an automatic scaling policy request gracefully. Amazon EMR implements a blacklisting mechanism in Spark that is built on top of YARN's decommissioning mechanism. This mechanism helps ensure that no new tasks are scheduled on a node that is decommissioning, while at the same time allowing tasks that are already running to complete. In addition, there are features to help recover Spark jobs faster if shuffle blocks are lost when a node terminates. The recomputation process is triggered sooner and optimized to recompute faster with fewer stage retries, and jobs can be prevented from failing because of fetch failures that are caused by missing shuffle blocks.

**Important**
The `spark.decommissioning.timeout.threshold` setting was added in Amazon EMR release version 5.11.0 to improve Spark resiliency when you use Spot instances. In earlier release versions, when a node uses a Spot instance, and the instance is terminated because of bid price, Spark may not be able to handle the termination gracefully. Jobs may fail, and shuffle recomputations could take a significant amount of time. For this reason, we recommend using release version 5.11.0 or later if you use Spot instances.

**Spark Node Decommissioning Settings**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spark.blacklist.decommissioning.enabled</code></td>
<td>When set to <code>true</code>, Spark blacklists nodes that are in the decommissioning state in YARN. Spark does not schedule new tasks on executors running on that node. Tasks already running are allowed to complete.</td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>spark.blacklist.decommissioning.timeout</code></td>
<td>The amount of time that a node in the decommissioning state is blacklisted. By default, this value is set to one hour, which is also the default for <code>yarn.resourcemanager.decommissioning.timeout</code>. To ensure that a node is blacklisted for its entire decommissioning period, set this value equal to or greater than <code>yarn.resourcemanager.decommissioning.timeout</code>. After the decommissioning timeout expires, the node transitions to a decommissioned state, and Amazon EMR can terminate the node's EC2 instance. If any tasks are still running after the timeout expires, they are lost or killed and rescheduled on executors running on other nodes.</td>
<td><code>1h</code></td>
</tr>
<tr>
<td><code>spark.decommissioning.timeout.threshold</code></td>
<td>Available in Amazon EMR release version 5.11.0 or later.</td>
<td><code>20s</code></td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>later. Specified in seconds. When a node transitions to the decommissioning state, if the host will decommission within a time period equal to or less than this value, Amazon EMR not only blacklists the node, but also cleans up the host state (as specified by spark.resourceManager.cleanupExpiredHost) without waiting for the node to transition to a decommissioned state. This allows Spark to handle Spot instance terminations better because Spot instances decommission within a 20-second timeout regardless of the value of yarn.resourcemanager.decommissioning.timeout, which may not provide other nodes enough time to read shuffle files.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.resourceManager.cleanupExpiredHost</td>
<td>When set to true, Spark unregisters all cached data and shuffle blocks that are stored in executors on nodes that are in the decommissioned state. This speeds up the recovery process.</td>
<td>true</td>
</tr>
<tr>
<td>spark.stage.attempt.ignoreOnDecommissionFetchFailure</td>
<td>When set to true, helps prevent Spark from failing stages and eventually failing the job because of too many failed fetches from decommissioned nodes. Failed fetches of shuffle blocks from a node in the decommissioned state will not count toward the maximum number of consecutive fetch failures.</td>
<td>true</td>
</tr>
</tbody>
</table>

### Spark ThriftServer Environment Variable

Spark sets the Hive Thrift Server Port environment variable, HIVE_SERVER2_THRIFT_PORT, to 10001.

### Changing Spark Default Settings

You change the defaults in spark-defaults.conf using the spark-defaults configuration classification when you create the cluster or the maximizeResourceAllocation setting in the spark configuration classification.

The following procedures show how to modify settings using the CLI or console.
To create a cluster with spark.executor.memory set to 2G using the CLI

- Create a cluster with Spark installed and spark.executor.memory set to 2G, using the following:

```
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Spark \\
--instance-type m3.xlarge --instance-count 2 --service-role EMR_DefaultRole --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

myConfig.json:

```json
[  
  {  
    "Classification": "spark-defaults",  
    "Properties": {  
      "spark.executor.memory": "2G"  
    }  
  }  
]
```

Note
If you plan to store your configuration in Amazon S3, you must specify the URL location of the object. For example:

```
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Spark \\
--instance-type m3.xlarge --instance-count 3 --service-role EMR_DefaultRole --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

To create a cluster with spark.executor.memory set to 2G using the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster.
3. Choose Go to advanced options
4. For the Software Configuration field, choose Release or later.
5. Choose either Spark or All Applications from the list, then choose Configure and add.
6. Choose Edit software settings and enter the following configuration:

```
classification=spark-defaults,properties=[spark.executor.memory=2G]
```
7. Select other options as necessary and then choose Create cluster.

To set maximizeResourceAllocation

- Create a cluster with Spark installed and maximizeResourceAllocation set to true using the AWS CLI:

```
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Spark \\
--instance-type m3.xlarge --instance-count 2 --service-role EMR_DefaultRole --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole --configurations file://./myConfig.json
```
Access the Spark Shell

The Spark shell is based on the Scala REPL (Read-Eval-Print-Loop). It allows you to create Spark programs interactively and submit work to the framework. You can access the Spark shell by connecting to the master node with SSH and invoking spark-shell. For more information about connecting to the master node, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide. The following examples use Apache HTTP Server access logs stored in Amazon S3.

**Note**
The bucket used in these examples is available to clients that can access US East (N. Virginia).

By default, the Spark shell creates its own SparkContext object called sc. You can use this context if it is required within the REPL. sqlContext is also available in the shell and it is a HiveContext.

**Example Using the Spark shell to count the occurrences of a string in a file stored in Amazon S3**

This example uses sc to read a textFile in Amazon S3.

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@404721db
```

Spark creates the textFile and associated data structure. Next, the example counts the number of lines in the log file with the string "cartoonnetwork.com":

```
scala> val linesWithCartoonNetwork = textFile.filter(line =>
    line.contains("cartoonnetwork.com")).count()
at <console>:23
</snip>
```

```
Example Using the Python-based Spark shell to count the occurrences of a string in a file stored in Amazon S3

Spark also includes a Python-based shell, pyspark, that you can use to prototype Spark programs written in Python. Just as with spark-shell, invoke pyspark on the master node; it also has the same SparkContext object.

```python
>>> sc
<pspark.context.SparkContext object at 0x7fe7e659fa50>
```

Spark creates the textFile and associated data structure. Next, the example counts the number of lines in the log file with the string "cartoonnetwork.com".

```python
>>> linesWithCartoonNetwork = textfile.filter(lambda line: "cartoonnetwork.com" in line).count()
```

Using Amazon SageMaker Spark for Machine Learning

When using Amazon EMR release version 5.11.0 and later, the aws-sagemaker-spark-sdk component is installed along with Spark. This component installs Amazon SageMaker Spark and associated dependencies for Spark integration with Amazon SageMaker. You can use Amazon SageMaker Spark to construct Spark machine learning (ML) pipelines using Amazon SageMaker stages. For more information, see the Amazon SageMaker Spark Readme on GitHub and Using Apache Spark with Amazon SageMaker in the Amazon SageMaker Developer Guide.

Write a Spark Application

Spark applications can be written in Scala, Java, or Python. There are several examples of Spark applications located on Spark Examples topic in the Apache Spark documentation. The Estimating Pi example is shown below in the three natively supported applications. You can also view complete examples in $SPARK_HOME/examples and at GitHub. For more information about how to build JARs for Spark, see the Quick Start topic in the Apache Spark documentation.

Scala

```scala
package org.apache.spark.examples
```
import scala.math.random
import org.apache.spark._

/** Computes an approximation to pi */
object SparkPi {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Pi")
    val spark = new SparkContext(conf)
    val slices = if (args.length > 0) args(0).toInt else 2
    val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid overflow
    val count = spark.parallelize(1 until n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x*x + y*y < 1) 1 else 0
    }.reduce(_ + _)
    println("Pi is roughly "+ 4.0 * count / n)
    spark.stop()
  }
}

Java

package org.apache.spark.examples;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;
import java.util.ArrayList;
import java.util.List;

/**
 * Computes an approximation to pi
 * Usage: JavaSparkPi [slices]
 */
public final class JavaSparkPi {
  public static void main(String[] args) throws Exception {
    SparkConf sparkConf = new SparkConf().setAppName("JavaSparkPi");
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    int slices = (args.length == 1) ? Integer.parseInt(args[0]) : 2;
    int n = 100000 * slices;
    List<Integer> l = new ArrayList<Integer>(n);
    for (int i = 0; i < n; i++) { l.add(i); }
    JavaRDD<Integer> dataSet = jsc.parallelize(l, slices);
    int count = dataSet.map(new Function<Integer, Integer>() {
      @Override
      public Integer call(Integer integer) {
        double x = Math.random() * 2 - 1;
        double y = Math.random() * 2 - 1;
        return (x*x + y*y < 1) ? 1 : 0;
      }
    }).reduce(new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer integer, Integer integer2) {
        return integer + integer2;
      }
    });
    System.out.println("Pi is roughly " + 4.0 * count / n);
    jsc.stop();
  }
}
Python 2.7

```python
import sys
from random import random
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
    """
    Usage: pi [partitions]
    """
    sc = SparkContext(appName="PythonPi")
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
    print "Pi is roughly %f" % (4.0 * count / n)
    sc.stop()
```

Adding a Spark Step

You can use Amazon EMR Steps in the Amazon EMR Management Guide to submit work to the Spark framework installed on an EMR cluster. In the console and CLI, you do this using a Spark application step, which runs the `spark-submit` script as a step on your behalf. With the API, you use a step to invoke `spark-submit` using `command-runner.jar`.

For more information about submitting applications to Spark, see the Submitting Applications topic in the Apache Spark documentation.

**Note**

If you choose to deploy work to Spark using the client deploy mode, your application files must be in a local path on the EMR cluster. You cannot currently use S3 URIs for this location in client mode. However, you can use S3 URIs with cluster deploy mode.

To submit a Spark step using the console

1. Open the Amazon EMR console at [https://console.aws.amazon.com/elasticmapreduce/](https://console.aws.amazon.com/elasticmapreduce/).
2. In the Cluster List, choose the name of your cluster.
3. Scroll to the Steps section and expand it, then choose Add step.
4. In the Add Step dialog box:
• For **Step type**, choose **Spark application**.

• For **Name**, accept the default name (Spark application) or type a new name.

• For **Deploy mode**, choose **Cluster** or **Client** mode. Cluster mode launches your driver program on the cluster (for JVM-based programs, this is `main()`), while client mode launches the driver program locally. For more information, see **Cluster Mode Overview** in the Apache Spark documentation.

**Note**
Cluster mode allows you to submit work using S3 URIs. Client mode requires that you put the application in the local file system on the cluster master node.

• Specify the desired **Spark-submit options**. For more information about `spark-submit` options, see **Launching Applications with spark-submit**.

• For **Application location**, specify the local or S3 URI path of the application.

• For **Arguments**, leave the field blank.

• For **Action on failure**, accept the default option (**Continue**).

5. Choose **Add**. The step appears in the console with a status of **Pending**.
6. The status of the step changes from **Pending** to **Running** to **Completed** as the step runs. To update the status, choose the **Refresh** icon above the **Actions** column.
7. The results of the step are located in the Amazon EMR console Cluster Details page next to your step under **Log Files** if you have logging configured. You can optionally find step information in the log bucket you configured when you launched the cluster.

**To submit work to Spark using the AWS CLI**

Submit a step when you create the cluster or use the `aws emr add-steps` subcommand in an existing cluster.

1. **Use create-cluster.**

   ```bash
   aws emr create-cluster --name "Add Spark Step Cluster" --release-label emr-5.11.0 --applications Name=Spark \ --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 \ --steps Type=Spark,Name="Spark Program",ActionOnFailure=CONTINUE,Args=[--class,org.apache.spark.examples.SparkPi,/usr/lib/spark/lib/spark-examples.jar,10] --use-default-roles
   
   An alternative using **command-runner.jar**: 

   ```bash
   aws emr create-cluster --name "Add Spark Step Cluster" --release-label emr-5.11.0 \ --applications Name=Spark --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 \ --steps Type=CUSTOM_JAR,Name="Spark Program",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args=[spark-example,SparkPi,10] --use-default-roles
   
   **Note**
   Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

2. **Alternatively, add steps to a cluster already running. Use add-steps.**

   ```bash
   aws emr add-steps --cluster-id j-2AXXXXXXXXXGPLF --steps Type=Spark,Name="Spark Program",ActionOnFailure=CONTINUE,Args=[--class,org.apache.spark.examples.SparkPi,/usr/lib/spark/lib/spark-examples.jar,10]
   ```
An alternative using command-runner.jar:

```bash
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF --steps Type=CUSTOM_JAR,Name="Spark Program",Jar="command-runner.jar",ActionOnFailure=CONTINUE,Args=[spark-example,SparkPi,10]
```

To submit work to Spark using the SDK for Java

- To submit work to a cluster, use a step to run the `spark-submit` script on your EMR cluster. Add the step using the `addJobFlowSteps` method in `AmazonElasticMapReduceClient`:

```java
AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);
AmazonElasticMapReduce emr = new AmazonElasticMapReduceClient(credentials);
StepFactory stepFactory = new StepFactory();
AmazonElasticMapReduceClient emr = new AmazonElasticMapReduceClient(credentials);
AddJobFlowStepsRequest req = new AddJobFlowStepsRequest();
req.withJobFlowId("j-1K48XXXXXXHCB");
List<StepConfig> stepConfigs = new ArrayList<StepConfig>();
HadoopJarStepConfig sparkStepConf = new HadoopJarStepConfig()
    .withJar("command-runner.jar")
    .withArgs("spark-submit","--executor-memory","1g","--class","org.apache.spark.examples.SparkPi","/usr/lib/spark/lib/spark-examples.jar","10");
StepConfig sparkStep = new StepConfig()
    .withName("Spark Step")
    .withActionOnFailure("CONTINUE")
    .withHadoopJarStep(sparkStepConf);
stepConfigs.add(sparkStep);
req.withSteps(stepConfigs);
AddJobFlowStepsResult result = emr.addJobFlowSteps(req);
```

View the results of the step by examining the logs for the step. You can do this in the AWS Management Console if you have enabled logging by choosing **Steps**, selecting your step, and then, for **Log files**, choosing either **stdout** or **stderr**. To see the logs available, choose **View Logs**.

**Overriding Spark Default Configuration Settings**

You may want to override Spark default configuration values on a per-application basis. You can do this when you submit applications using a step, which essentially passes options to `spark-submit`. For example, you may wish to change the memory allocated to an executor process by changing `spark.executor.memory`. You would supply the `--executor-memory` switch with an argument like the following:

```bash
spark-submit --executor-memory 1g --class org.apache.spark.examples.SparkPi /usr/lib/spark/lib/spark-examples.jar 10
```

Similarly, you can tune `--executor-cores` and `--driver-memory`. In a step, you would provide the following arguments to the step:

```bash
--executor-memory 1g --class org.apache.spark.examples.SparkPi /usr/lib/spark/lib/spark-examples.jar 10
```
You can also tune settings that may not have a built-in switch using the `--conf` option. For more information about other settings that are tunable, see the Dynamically Loading Spark Properties topic in the Apache Spark documentation.

View Spark Application History

Amazon EMR collects information from YARN applications on your cluster and keeps historical information for up to seven days after applications have completed. Detailed information is available for Spark applications. This feature enables you to view data that is otherwise available in the Spark History Server's web interface, but without having to establish a potentially complicated SSH tunnel for connecting to the cluster's master node. You can also identify and open relevant log files more easily than you would by connecting to the master node and searching the file structure.

To view Spark application history

1. From the Clusters list, select the Name of a cluster and then select Application history.

   Each application is listed by Application ID.

   ![Application List](image)

2. Expand a row to see basic diagnostic information for the application or select the Application ID to view additional details.

   For more information, see View Application History in the Amazon EMR Management Guide.

Accessing the Spark Web UIs

You can view the Spark web UIs by following the procedures to create an SSH tunnel or create a proxy in the section called Connect to the Cluster in the Amazon EMR Management Guide and then navigating to the YARN ResourceManager for your cluster. Choose the link under Tracking UI for your application. If your application is running, you see ApplicationMaster. This takes you to the Spark application driver's web UI at port 4040 wherever the driver is located. The driver may be located on the cluster's master node if you run in YARN client mode. If you are running an application in YARN cluster mode, the driver is located in the ApplicationMaster for the application on the cluster. If your application has finished, you see History, which takes you to the Spark HistoryServer UI port number at 18080 of the EMR cluster's master node. This is for applications that have already completed. You can also navigate to the Spark HistoryServer UI directly at http://master-public-dns-name:18080/.
Apache Flink

Apache Flink is a streaming dataflow engine that makes it easy to run real-time stream processing on high-throughput data sources. It supports event time semantics for out-of-order events, exactly-once semantics, backpressure control, and APIs optimized for writing both streaming and batch applications.

Additionally, Flink has connectors for third-party data sources, such as the following:

- Amazon Kinesis Streams
- Apache Kafka
- Elasticsearch
- Twitter Streaming API
- Cassandra

Currently, Amazon EMR supports Flink as a YARN application so that you can manage resources along with other applications within a cluster. Flink-on-YARN has an easy way to submit transient Flink jobs or you can create a long-running cluster that accepts multiple jobs and allocates resources according to the overall YARN reservation.

**Note**
Support for the FlinkKinesisConsumer class was added in Amazon EMR version 5.2.1.

Flink Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flink 1.3.2</td>
<td>emr-5.11.0</td>
<td>emrfs, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, flink-client</td>
</tr>
</tbody>
</table>

Creating a Cluster with Flink

Clusters can be launched using the AWS Management Console, AWS CLI, or an AWS SDK.
To launch a cluster with Flink installed using the console

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster, Go to advanced options.
3. For Software Configuration, choose EMR Release emr-5.1.0 or later.
4. Choose Flink as an application, along with any others to install.
5. Select other options as necessary and choose Create cluster.

To launch a cluster with Flink using the AWS CLI

- Create the cluster with the following command:

```bash
aws emr create-cluster --name "Cluster with Flink" --release-label emr-5.11.0 \
--applications Name=Flink --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

Configuring Flink

You may want to configure Flink using a configuration file. For example, the main configuration file for Flink is called `flink-conf.yaml`. This is configurable using the Amazon EMR configuration API so when you start your cluster, you supply a configuration for the file to modify.

To configure the number of task slots used for Flink using the AWS CLI

1. Create a file, `configuration.json`, with the following content:

```json
[
    {
        "Classification": "flink-conf",
        "Properties": {
            "taskmanager.numberOfTaskSlots": "2"
        }
    }
]
```

2. Next, create a cluster with the following configuration:

```bash
aws emr create-cluster --release-label emr-5.11.0 \
--applications Name=Flink \
--configurations file://./configurations.json \
--region us-east-1 \
--log-uri s3://myLogUri \
--instance-type m3.xlarge \
--instance-count 2 \
--service-role EMR_DefaultRole \
--ec2-attributes KeyName=YourKeyName,InstanceProfile=EMR_EC2_DefaultRole
```

Note
It is also possible to change some configurations using the Flink API. For more information, see Basic API Concepts in the Flink documentation.
Parallelism Options

As the owner of your application, you know best what resources should be assigned to tasks within Flink. For the purposes of the examples in this documentation, use the same number of tasks as the slave instances that you use for the application. We generally recommend this for the initial level of parallelism but you can also increase the granularity of parallelism using task slots, which should generally not exceed the number of virtual cores per instance. For more information about Flink's architecture, see Concepts in the Flink documentation.

Configurable Files

Currently, the files that are configurable within the Amazon EMR configuration API are:

- flink-conf.yaml
- log4j.properties
- log4j-yarn-session.properties
- log4j-cli.properties

Working with Flink Jobs in Amazon EMR

There are several ways to interact with Flink on Amazon EMR: through Amazon EMR steps, the Flink interface found on the ResourceManager Tracking UI, and at the command line. All of these also allow you to submit a JAR file of a Flink application to run.

Additionally, you can run Flink applications as a long-running YARN job or as a transient cluster. In a long-running job, you can submit multiple Flink applications to one Flink cluster running on Amazon EMR. If you run Flink as a transient job, your Amazon EMR cluster exists only for the time it takes to run the Flink application, so you are only charged for the resources and time used. In either case, you can submit a Flink job using the Amazon EMR AddSteps API operation, or as a step argument to the RunJobFlow operation or AWS CLI create-cluster command.

Start a Flink Long-Running YARN Job as a Step

You may want to start a long-running Flink job that multiple clients can submit to through YARN API operations. You start a Flink YARN session and submit jobs to the Flink JobManager, which is located on the YARN node that hosts the Flink session Application Master daemon. To start a YARN session, use the following steps from the console, AWS CLI, or Java SDK.

To submit a long-running job using the console

Submit the long-running Flink session using the flink-yarn-session command in an existing cluster.

Note

The flink-yarn-session command was added in Amazon EMR version 5.5.0 as a wrapper for the yarn-session.sh script to simplify execution. If you use an earlier version of Amazon EMR, substitute bash -c "/usr/lib/flink/bin/yarn-session.sh -n 2 -d" for Argument in the steps that follow.

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. In the cluster list, select the cluster you previously launched.
3. In the cluster details page, choose Steps, Add Step.
4. Enter parameters using the guidelines that follow and then choose Add.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step type</td>
<td>Custom JAR</td>
</tr>
<tr>
<td>Name</td>
<td>A name to help you identify the step. For example, Flink_Long_Running_Session.</td>
</tr>
<tr>
<td>Jar location</td>
<td>command-runner.jar</td>
</tr>
<tr>
<td>Arguments</td>
<td>The flink-yarn-session command with arguments appropriate for your application. For example, flink-yarn-session -n 2 -d starts a long-running Flink session within your YARN cluster in a detached state (-d) with two task managers (-n 2). See YARN Setup in the latest Flink documentation for argument details. <strong>Note</strong> With Amazon EMR versions earlier than 5.5.0, you must specify the Flink script yarn-session.sh directly instead of flink-yarn-session, specifying the full path to the script. For example, bash -c &quot;/usr/lib/flink/bin/yarn-session.sh -n 2 -d&quot;.</td>
</tr>
</tbody>
</table>
Submit Work to an Existing, Long-Running Flink YARN Job

You can submit work using a command-line option but you can also use Flink’s native interface proxied through the YARN ResourceManager. To submit through an EMR step using the Flink CLI, specify the long-running Flink cluster’s YARN application ID. To do this, run `yarn application -list` on the EMR command line or through the YarnClient API operation:
Submit a Transient Flink Job

The following example launches the Flink WordCount example by adding a step to an existing cluster.

Example Console

In the console details page for an existing cluster, add the step by choosing Add Step for the Steps field.
Example SDK for Java

The following examples illustrate two approaches to running a Flink job. The first example submits a Flink job to a running cluster. The second example creates a cluster that runs a Flink job and then terminates on completion.

```java
List<StepConfig> stepConfigs = new ArrayList<StepConfig>();
HadoopJarStepConfig flinkWordCountConf = new HadoopJarStepConfig()
    .withJar("command-runner.jar")
    "--input", "s3://myBucket/pg11.txt", "--output", "s3://myBucket/alice/");
StepConfig flinkRunWordCount = new StepConfig()
    .withName("Flink add a wordcount step")
    .withActionOnFailure("CONTINUE")
    .withHadoopJarStep(flinkWordCountConf);
stepConfigs.add(flinkRunWordCount);
    .withJobFlowId("myClusterId")
    .withSteps(stepConfigs));
```

```java
List<StepConfig> stepConfigs = new ArrayList<StepConfig>();
HadoopJarStepConfig flinkWordCountConf = new HadoopJarStepConfig()
    .withJar("command-runner.jar")
    .withArgs("bash","-c", "flink run -m yarn-cluster -yn 2 /usr/lib/flink/examples/streaming/WordCount.jar "
    + "--input", "s3://myBucket/pg11.txt", "--output", "s3://myBucket/alice/");
StepConfig flinkRunWordCountStep = new StepConfig()
    .withName("Flink add a wordcount step and terminate")
    .withActionOnFailure("CONTINUE")
    .withHadoopJarStep(flinkWordCountConf);
stepConfigs.add(flinkRunWordCountStep);
RunJobFlowRequest request = new RunJobFlowRequest()
    .withName("flink-transient")
    .withReleaseLabel("emr-5.2.1")
```

165
.withApplications(myApps)
    .withServiceRole("EMR_DefaultRole")
    .withJobFlowRole("EMR_EC2_DefaultRole")
    .withLogUri("s3://myLogBucket")
    .withInstances(
        new JobFlowInstancesConfig().withEc2KeyName("myKeyName").withInstanceCount(2)
            .withKeepJobFlowAliveWhenNoSteps(false).withMasterInstanceType("m3.xlarge")
                .withSlaveInstanceType("m3.xlarge"))
    .withSteps(stepConfigs);

RunJobFlowResult result = emr.runJobFlow(request);

Example AWS CLI

Use the add-steps subcommand to submit new jobs to an existing Flink cluster:

```bash
aws emr add-steps --cluster-id myClusterId
    --steps Type=CUSTOM_JAR,Name=Flink_Transient_No_Terminate,Jar=command-runner.jar,
    Args="flink","run","-m","yarn-cluster","-yid","application_1473169569237_0002","-yn","2","/
    /usr/lib/flink/examples/streaming/WordCount.jar",
    "--input","s3://myBucket/pg11.txt","--output","s3://myBucket/alice2/"
    --region myRegion
```

Use the create-cluster subcommand to create a transient EMR cluster that terminates when the
Flink job completes:

```bash
aws emr create-cluster --release-label emr-5.2.1
    --name "Flink_Transient"
    --applications Name=Flink
    --configurations file://./configurations.json
    --region us-east-1
    --log-uri s3://myLogUri
    --auto-terminate
    --instance-type m3.xlarge
    --instance-count 2
    --service-role EMR_DefaultRole
    --ec2-attributes KeyName=YourKeyName,InstanceProfile=EMR_EC2_DefaultRole
    --steps Type=CUSTOM_JAR,Jar=command-runner.jar,Name=Flink_Long_Running_Session,
    Args="bash","-c",""flink run -m yarn-cluster -yn 2 /usr/lib/flink/examples/streaming/
    WordCount.jar"
    --input s3://myBucket/pg11.txt --output s3://myBucket/alice/"
```

Using the Scala Shell

Currently, the Flink Scala shell for EMR clusters is only configured to start new YARN sessions. You can
use the Scala shell by following the procedure below.

Using the Flink Scala shell on the master node

1. Log in to the master node using SSH as described in Connect to the Master Node using SSH.
2. Type the following to start a shell:

   ```bash
   % flink-scala-shell yarn -n 1
   ```

   In earlier versions of Amazon EMR, use:
% /usr/lib.flink/bin/start-scala-shell.sh yarn -n 1

This starts the Flink Scala shell so you can interactively use Flink. Just as with other interfaces and options, you can scale the `-n` option value used in the example based on the number of tasks you want to run from the shell.

Finding the Flink Web Interface

The Application Master that belongs to the Flink application hosts the Flink web interface, which is an alternative way to submit a JAR as a job or to view the current status of other jobs. The Flink web interface is active as long as you have a Flink session running. If you have a long-running YARN job already active, you can follow the instructions in the Connect to the Master Node Using SSH topic in the Amazon EMR Management Guide to connect to the YARN ResourceManager. For example, if you’ve set up an SSH tunnel and have activated a proxy in your browser, you choose the ResourceManager connection under Connections in your EMR cluster details page.

After you find the ResourceManager, select the YARN application that’s hosting a Flink session. Choose the link under the Tracking UI column.

In the Flink web interface, you can view configuration, submit your own custom JAR as a job, or monitor jobs in progress.
Apache Sqoop

Apache Sqoop is a tool for transferring data between Amazon S3, Hadoop, HDFS, and RDBMS databases. For more information, see the Apache Sqoop website.

Sqoop Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Sqoop JDBC and Database Support

By default, Sqoop has a MariaDB and PostgreSQL driver installed. The PostgreSQL driver installed for Sqoop only works for PostgreSQL 8.4. To install an alternate set of JDBC connectors for Sqoop, install them in /usr/lib/sqoop/lib. The following are links for various JDBC connectors:

- MariaDB: About MariaDB Connector/J.
- PostgreSQL: PostgreSQL JDBC Driver.
- SQLServer: Download Microsoft JDBC Driver for SQL Server.
- MySQL: Download Connector/J
- Oracle: Get Oracle JDBC drivers and UCP from the Oracle Maven Repository

Sqoop's supported databases are listed at the following url, http://sqoop.apache.org/docs/version/SqoopUserGuide.html#_supported_databases, where version is the version of Sqoop you are using, for example 1.4.6. If the JDBC connect string does not match those in this list, you will need to specify a driver.

For example, you can export to an Amazon Redshift database table with the following command (for JDBC 4.1):
You can use both the MariaDB and MySQL connection strings but if you specify the MariaDB connection string, you need to specify the driver:

```
```

If you are using Secure Socket Layer encryption to access your database, you need to use a JDBC URI like in the following Sqoop export example:

```
```

For more information about SSL encryption in RDS, see Using SSL to Encrypt a Connection to a DB Instance in the Amazon Relational Database Service User Guide.

For more information, see the Apache Sqoop documentation.
Apache Tez

Apache Tez is a framework for creating a complex directed acyclic graph (DAG) of tasks for processing data. In some cases, it is used as an alternative to Hadoop MapReduce. For example, Pig and Hive workflows can run using Hadoop MapReduce or they can use Tez as an execution engine. For more information, see https://tez.apache.org/.

Tez Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tez 0.8.4</td>
<td>emr-5.11.0</td>
<td>emrfs, emr-goodies, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, tez-on-yarn</td>
</tr>
</tbody>
</table>

Creating a Cluster with Tez

Install Tez by choosing that application when you create the cluster.

To launch a cluster with Tez installed using the console

The following procedure creates a cluster with Tez installed. For more information about launching clusters with the console, see Step 3: Launch an Amazon EMR Cluster in the Amazon EMR Management Guide.

1. Open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster to use Quick Create.
3. For Software Configuration, choose Amazon Release Version emr-4.7.0 or later.
4. For Select Applications, choose either All Applications or Tez.
5. Select other options as necessary and then choose Create cluster.

To launch a cluster with Tez using the AWS CLI

- Create the cluster with the following command:

```bash
aws emr create-cluster --name "Cluster with Tez" --release-label emr-5.11.0
```
Configuring Tez

You configure Tez by setting values in `tez-site.xml` using the `tez-site` configuration classification when you create your cluster. If you want to use Hive with Tez, you must also modify the `hive-site` configuration classification.

To change the root logging level in Tez

- Create a cluster with Tez installed and set `tez.am.log.level` to DEBUG, using the following command:

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Tez \ --instance-type m3.xlarge --instance-count 2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

**Note**
Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

`myConfig.json`:

```json
[
  {
    "Classification": "tez-site",
    "Properties": {
      "tez.am.log.level": "DEBUG"
    }
  }
]
```

**Note**
If you plan to store your configuration in Amazon S3, you must specify the URL location of the object. For example:

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Tez Name=Hive \ --instance-type m3.xlarge --instance-count 3 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

**To change the Hive or Pig execution engine to Tez**

1. Create a cluster with Hive and Tez installed and set `hive.execution.engine` to `tez`, using the following command:

```bash
aws emr create-cluster --release-label emr-5.11.0 --applications Name=Tez Name=Hive \ --instance-type m3.xlarge --instance-count 2 --configurations https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```
**Note**

Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

myConfig.json:

```json
[
  {
    "Classification": "hive-site",
    "Properties": {
      "hive.execution.engine": "tez"
    }
  }
]
```

2. To set the execution engine for Pig, modify `pig.properties` by setting `myConfig.json` to the following:

```json
[
  {
    "Classification": "hive-site",
    "Properties": {
      "hive.execution.engine": "tez"
    }
  },
  {
    "Classification": "pig-properties",
    "Properties": {
      "exectype": "tez"
    }
  }
]
```

3. Create the cluster as above but add Pig as an application.

## Using Tez

The following examples show you how to use Tez for the data and scripts used in the tutorial called *Getting Started: Analyzing Big Data with Amazon EMR* shown in Step 3.

### Compare the Hive runtimes of MapReduce vs. Tez

1. Create a cluster as shown in the procedure called *To launch a cluster with Tez installed using the console* (p. 171). Choose Hive as an application in addition to Tez.
2. Connect to the cluster using SSH. For more information, see *Connect to the Master Node Using SSH*.
3. Run the `Hive_CloudFront.q` script using MapReduce with the following command, where `region` is the region in which your cluster is located:

```
hive -f s3://region.elasticmapreduce.samples/cloudfront/code/Hive_CloudFront.q -d INPUT=s3://region.elasticmapreduce.samples -d OUTPUT=s3://myBucket/mr-test/
```

The output should look something like the following:

```text
<snip>
Starting Job = job_1464200677872_0002, Tracking URL = http://ec2-host:20888/proxy/application_1464200677872_0002/
```
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1464200677872_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-05-27 04:53:11,258 Stage-1 map = 0%, reduce = 0%
2016-05-27 04:53:25,820 Stage-1 map = 13%, reduce = 0%, Cumulative CPU 10.45 sec
2016-05-27 04:53:32,034 Stage-1 map = 33%, reduce = 0%, Cumulative CPU 16.06 sec
2016-05-27 04:53:35,139 Stage-1 map = 40%, reduce = 0%, Cumulative CPU 18.9 sec
2016-05-27 04:53:37,211 Stage-1 map = 53%, reduce = 0%, Cumulative CPU 21.6 sec
2016-05-27 04:53:41,371 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 25.08 sec
2016-05-27 04:53:49,675 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 29.93 sec
MapReduce Total cumulative CPU time: 29 seconds 930 msec
Ended Job = job_1464200677872_0002
Moving data to: s3://myBucket/mr-test/os_requests
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 29.93 sec HDFS Read: 599 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 29 seconds 930 msec
OK
Time taken: 49.699 seconds

4. Using a text editor, replace the hive.execution.engine value with tez in /etc/hive/conf/hive-site.xml.
5. Kill the HiveServer2 process with the following command:

```
sudo kill -9 $(pgrep -f HiveServer2)
```

Upstart automatically restarts the Hive server with your configuration changes loaded.

6. Now run the job with the Tez execution engine using the following command:

```
hive -f s3://region.elasticmapreduce.samples/cloudfront/code/Hive_CloudFront.q \
-d INPUT=s3://region.elasticmapreduce.samples -d OUTPUT=s3://myBucket/tez-test/
```

The output should look something like the following:

```
Time taken: 0.517 seconds
Query ID = hadoop_20160527050505_dcdec075f-8338-4041-adc3-d2ffe69dfcdd
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1464200677872_0003)

------------------------------------------------------------------------
| VERTICES | STATUS | TOTAL | COMPLETED | RUNNING | PENDING | FAILED | KILLED |
------------------------------------------------------------------------
| Map 1 ....... | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 |
| Reducer 2 ...... | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 |
------------------------------------------------------------------------
VERTICES: 02/02 [==============================================] 100% ELAPSED TIME: 27.61 s
------------------------------------------------------------------------
Moving data to: s3://myBucket/tez-test/os_requests
OK
Time taken: 30.711 seconds
```

The time to run the same application took approximately 20 seconds (40%) less time using Tez.

Tez Web UI

Tez has its own web user interface. To view the web UI, see:
Timeline Server

The YARN Timeline Service is configured to run when Tez is installed. To view jobs submitted through Tez or MapReduce execution engines using the timeline service, view the web UI:

http://masterDNS:8080/tez-ui

For more information, see View Web Interfaces Hosted on Amazon EMR Clusters in the Amazon EMR Management Guide.
Apache Zeppelin

Use Apache Zeppelin as a notebook for interactive data exploration. For more information about Zeppelin, see https://zeppelin.apache.org/.

To access the Zeppelin web interface, set up an SSH tunnel to the master node and a proxy connection. For more information, see View Web Interfaces Hosted on Amazon EMR Clusters

Zeppelin Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>

Considerations When Using Zeppelin on Amazon EMR

- Connect to Zeppelin using the same SSH tunneling method to connect to other web servers on the master node. Zeppelin server is found at port 8890.
- Zeppelin on Amazon EMR release 5.0.0 and later supports Shiro authentication.
- Zeppelin on Amazon EMR release 5.8.0 and later supports using AWS Glue Data Catalog as the metastore for Spark SQL. For more information, see Using AWS Glue Data Catalog as the Metastore for Spark SQL.
- Zeppelin does not use some of the settings defined in your cluster’s spark-defaults.conf configuration file (though it instructs YARN to allocate executors dynamically if you have enabled that setting). You must set executor settings (such as memory and cores) on the Interpreter tab and then restart the interpreter for them to be used.
- Zeppelin on Amazon EMR does not support the SparkR interpreter.
Apache ZooKeeper

Apache ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. For more information about ZooKeeper, see http://zookeeper.apache.org/.

ZooKeeper Release Information for This Release of Amazon EMR

We strongly recommend Amazon EMR release 5.11.1 (release label emr-5.11.1). This component information applies to both emr-5.11.1 and emr-5.11.0.

<table>
<thead>
<tr>
<th>Application</th>
<th>Amazon EMR Release Label</th>
<th>Components installed with this application</th>
</tr>
</thead>
</table>
Connectors and Utilities

Amazon EMR provides several connectors and utilities to access other AWS services as data sources. You can usually access data in these services within a program. For example, you can specify an Kinesis stream in a Hive query, Pig script, or MapReduce application and then operate on that data.

Topics
- Export, Import, Query, and Join Tables in DynamoDB Using Amazon EMR (p. 178)
- Kinesis (p. 191)
- S3DistCp (s3-dist-cp) (p. 193)

Export, Import, Query, and Join Tables in DynamoDB Using Amazon EMR

Note
The Amazon EMR-DynamoDB Connector is now open-sourced on GitHub. For more information, see https://github.com/awslabs/emr-dynamodb-connector.

DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. Developers can create a database table and grow its request traffic or storage without limit. DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent, fast performance. Using Amazon EMR and Hive you can quickly and efficiently process large amounts of data, such as data stored in DynamoDB. For more information about DynamoDB go to the DynamoDB Developer Guide.

Apache Hive is a software layer that you can use to query map reduce clusters using a simplified, SQL-like query language called HiveQL. It runs on top of the Hadoop architecture. For more information about Hive and HiveQL, go to the HiveQL Language Manual. For more information about Hive and Amazon EMR, see Apache Hive (p. 109)

You can use Amazon EMR with a customized version of Hive that includes connectivity to DynamoDB to perform operations on data stored in DynamoDB:
- Loading DynamoDB data into the Hadoop Distributed File System (HDFS) and using it as input into an Amazon EMR cluster.
- Querying live DynamoDB data using SQL-like statements (HiveQL).
- Joining data stored in DynamoDB and exporting it or querying against the joined data.
- Exporting data stored in DynamoDB to Amazon S3.
- Importing data stored in Amazon S3 to DynamoDB.

To perform each of the following tasks, you'll launch an Amazon EMR cluster, specify the location of the data in DynamoDB, and issue Hive commands to manipulate the data in DynamoDB.

There are several ways to launch an Amazon EMR cluster: you can use the Amazon EMR console, the command line interface (CLI), or you can program your cluster using an AWS SDK or the Amazon EMR API. You can also choose whether to run a Hive cluster interactively or from a script. In this section, we will show you how to launch an interactive Hive cluster from the Amazon EMR console and the CLI.
Using Hive interactively is a great way to test query performance and tune your application. After you have established a set of Hive commands that will run on a regular basis, consider creating a Hive script that Amazon EMR can run for you.

**Warning**

Amazon EMR read or write operations on a DynamoDB table count against your established provisioned throughput, potentially increasing the frequency of provisioned throughput exceptions. For large requests, Amazon EMR implements retries with exponential backoff to manage the request load on the DynamoDB table. Running Amazon EMR jobs concurrently with other traffic may cause you to exceed the allocated provisioned throughput level. You can monitor this by checking the ThrottleRequests metric in Amazon CloudWatch. If the request load is too high, you can relaunch the cluster and set the Read Percent Setting (p. 189) or Write Percent Setting (p. 190) to a lower value to throttle the Amazon EMR operations. For information about DynamoDB throughput settings, see Provisioned Throughput.

**Topics**

- Set Up a Hive Table to Run Hive Commands (p. 179)
- Hive Command Examples for Exporting, Importing, and Querying Data in DynamoDB (p. 183)
- Optimizing Performance for Amazon EMR Operations in DynamoDB (p. 189)

### Set Up a Hive Table to Run Hive Commands

Apache Hive is a data warehouse application you can use to query data contained in Amazon EMR clusters using a SQL-like language. For more information about Hive, go to [http://hive.apache.org/](http://hive.apache.org/).

The following procedure assumes you have already created a cluster and specified an Amazon EC2 key pair. To learn how to get started creating clusters, see Step 3: Launch an Amazon EMR Cluster in the Amazon EMR Management Guide.

**To run Hive commands interactively**

1. Connect to the master node. For more information, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide.
2. At the command prompt for the current master node, type `hive`.
   
   You should see a hive prompt: `hive>
3. Enter a Hive command that maps a table in the Hive application to the data in DynamoDB. This table acts as a reference to the data stored in Amazon DynamoDB; the data is not stored locally in Hive and any queries using this table run against the live data in DynamoDB, consuming the table's read or write capacity every time a command is run. If you expect to run multiple Hive commands against the same dataset, consider exporting it first.

The following shows the syntax for mapping a Hive table to a DynamoDB table.

```sql
CREATE EXTERNAL TABLE hive_tablename
(hive_column1_name column1_datatype, hive_column2_name column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodb_tablename",
"dynamodb.column.mapping" = "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name...");
```

When you create a table in Hive from DynamoDB, you must create it as an external table using the keyword `EXTERNAL`. The difference between external and internal tables is that the data in internal
tables is deleted when an internal table is dropped. This is not the desired behavior when connected to Amazon DynamoDB, and thus only external tables are supported.

For example, the following Hive command creates a table named `hivetable1` in Hive that references the DynamoDB table named `dynamodbtable1`. The DynamoDB table `dynamodbtable1` has a hash-and-range primary key schema. The hash key element is `name` (string type), the range key element is `year` (numeric type), and each item has an attribute value for `holidays` (string set type).

```
CREATE EXTERNAL TABLE hivetable1 (col1 string, col2 bigint, col3 array<string>) 
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' 
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1", 
"dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");
```

Line 1 uses the HiveQL `CREATE EXTERNAL TABLE` statement. For `hivetable1`, you need to establish a column for each attribute name-value pair in the DynamoDB table, and provide the data type. These values are not case-sensitive, and you can give the columns any name (except reserved words).

Line 2 uses the `STORED BY` statement. The value of `STORED BY` is the name of the class that handles the connection between Hive and DynamoDB. It should be set to 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'.

Line 3 uses the `TBLPROPERTIES` statement to associate `hivetable1` with the correct table and schema in DynamoDB. Provide `TBLPROPERTIES` with values for the `dynamodb.table.name` parameter and `dynamodb.column.mapping` parameter. These values are case-sensitive.

**Note**

All DynamoDB attribute names for the table must have corresponding columns in the Hive table; otherwise, the Hive table won't contain the name-value pair from DynamoDB. If you do not map the DynamoDB primary key attributes, Hive generates an error. If you do not map a non-primary key attribute, no error is generated, but you won't see the data in the Hive table. If the data types do not match, the value is null.

Then you can start running Hive operations on `hivetable1`. Queries run against `hivetable1` are internally run against the DynamoDB table `dynamodbtable1` of your DynamoDB account, consuming read or write units with each execution.

When you run Hive queries against a DynamoDB table, you need to ensure that you have provisioned a sufficient amount of read capacity units.

For example, suppose that you have provisioned 100 units of read capacity for your DynamoDB table. This will let you perform 100 reads, or 409,600 bytes, per second. If that table contains 20GB of data (21,474,836,480 bytes), and your Hive query performs a full table scan, you can estimate how long the query will take to run:

\[
21,474,836,480 / 409,600 = 52,429 \text{ seconds} = 14.56 \text{ hours}
\]

The only way to decrease the time required would be to adjust the read capacity units on the source DynamoDB table. Adding more Amazon EMR nodes will not help.

In the Hive output, the completion percentage is updated when one or more mapper processes are finished. For a large DynamoDB table with a low provisioned read capacity setting, the completion percentage output might not be updated for a long time; in the case above, the job will appear to be 0% complete for several hours. For more detailed status on your job's progress, go to the Amazon EMR console; you will be able to view the individual mapper task status, and statistics for data reads. You can also log on to Hadoop interface on the master node and see the Hadoop statistics. This will show you the individual map task status and some data read statistics. For more information, see the following topics:
• Web Interfaces Hosted on the Master Node
• View the Hadoop Web Interfaces

For more information about sample HiveQL statements to perform tasks such as exporting or importing data from DynamoDB and joining tables, see Hive Command Examples for Exporting, Importing, and Querying Data in DynamoDB (p. 183).

To cancel a Hive request

When you execute a Hive query, the initial response from the server includes the command to cancel the request. To cancel the request at any time in the process, use the Kill Command from the server response.

1. Enter Ctrl+C to exit the command line client.
2. At the shell prompt, enter the Kill Command from the initial server response to your request.

Alternatively, you can run the following command from the command line of the master node to kill the Hadoop job, where job-id is the identifier of the Hadoop job and can be retrieved from the Hadoop user interface. For more information about the Hadoop user interface, see How to Use the Hadoop User Interface in the Amazon EMR Developer Guide.

```
hadoop job -kill job-id
```

Data Types for Hive and DynamoDB

The following table shows the available Hive data types and how they map to the corresponding DynamoDB data types.

<table>
<thead>
<tr>
<th>Hive type</th>
<th>DynamoDB type</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>string (S)</td>
</tr>
<tr>
<td>bigint or double</td>
<td>number (N)</td>
</tr>
<tr>
<td>binary</td>
<td>binary (B)</td>
</tr>
<tr>
<td>array</td>
<td>number set (NS), string set (SS), or binary set (BS)</td>
</tr>
</tbody>
</table>

The bigint type in Hive is the same as the Java long type, and the Hive double type is the same as the Java double type in terms of precision. This means that if you have numeric data stored in DynamoDB that has precision higher than is available in the Hive datatypes, using Hive to export, import, or reference the DynamoDB data could lead to a loss in precision or a failure of the Hive query.

Exports of the binary type from DynamoDB to Amazon Simple Storage Service (Amazon S3) or HDFS are stored as a Base64-encoded string. If you are importing data from Amazon S3 or HDFS into the DynamoDB binary type, it should be encoded as a Base64 string.

Hive Options

You can set the following Hive options to manage the transfer of data out of Amazon DynamoDB. These options only persist for the current Hive session. If you close the Hive command prompt and reopen it later on the cluster, these settings will have returned to the default values.
<table>
<thead>
<tr>
<th>Hive Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamodb.throughput.read.percent</td>
<td>Set the rate of read operations to keep your DynamoDB provisioned throughput rate in the allocated range for your table. The value is between 0.1 and 1.5, inclusively. The value of 0.5 is the default read rate, which means that Hive will attempt to consume half of the read provisioned throughout resources in the table. Increasing this value above 0.5 increases the read request rate. Decreasing it below 0.5 decreases the read request rate. This read rate is approximate. The actual read rate will depend on factors such as whether there is a uniform distribution of keys in DynamoDB. If you find your provisioned throughput is frequently exceeded by the Hive operation, or if live read traffic is being throttled too much, then reduce this value below 0.5. If you have enough capacity and want a faster Hive operation, set this value above 0.5. You can also oversubscribe by setting it up to 1.5 if you believe there are unused input/output operations available.</td>
</tr>
<tr>
<td>dynamodb.throughput.write.percent</td>
<td>Set the rate of write operations to keep your DynamoDB provisioned throughput rate in the allocated range for your table. The value is between 0.1 and 1.5, inclusively. The value of 0.5 is the default write rate, which means that Hive will attempt to consume half of the write provisioned throughout resources in the table. Increasing this value above 0.5 increases the write request rate. Decreasing it below 0.5 decreases the write request rate. This write rate is approximate. The actual write rate will depend on factors such as whether there is a uniform distribution of keys in DynamoDB. If you find your provisioned throughput is frequently exceeded by the Hive operation, or if live write traffic is being throttled too much, then reduce this value below 0.5. If you have enough capacity and want a faster Hive operation, set this value above 0.5. You can also oversubscribe by setting it up to 1.5 if you believe there are unused input/output operations available or this is the initial data upload to the table and there is no live traffic yet.</td>
</tr>
<tr>
<td>dynamodb.endpoint</td>
<td>Specify the endpoint in case you have tables in different regions. For more information about the available DynamoDB endpoints, see Regions and Endpoints.</td>
</tr>
<tr>
<td>dynamodb.max.map.tasks</td>
<td>Specify the maximum number of map tasks when reading data from DynamoDB. This value must be equal to or greater than 1.</td>
</tr>
<tr>
<td>dynamodb.retry.duration</td>
<td>Specify the number of minutes to use as the timeout duration for retrying Hive commands. This value must be an integer equal to or greater than 0. The default timeout duration is two minutes.</td>
</tr>
</tbody>
</table>
These options are set using the `SET` command as shown in the following example.

```
SET dynamodb.throughput.read.percent=1.0;
```

```
INSERT OVERWRITE TABLE s3_export SELECT * FROM hiveTableName;
```

**Hive Command Examples for Exporting, Importing, and Querying Data in DynamoDB**

The following examples use Hive commands to perform operations such as exporting data to Amazon S3 or HDFS, importing data to DynamoDB, joining tables, querying tables, and more.

Operations on a Hive table reference data stored in DynamoDB. Hive commands are subject to the DynamoDB table's provisioned throughput settings, and the data retrieved includes the data written to the DynamoDB table at the time the Hive operation request is processed by DynamoDB. If the data retrieval process takes a long time, some data returned by the Hive command may have been updated in DynamoDB since the Hive command began.

Hive commands `DROP TABLE` and `CREATE TABLE` only act on the local tables in Hive and do not create or drop tables in DynamoDB. If your Hive query references a table in DynamoDB, that table must already exist before you run the query. For more information about creating and deleting tables in DynamoDB, see Working with Tables in DynamoDB in the *Amazon DynamoDB Developer Guide*.

**Note**
When you map a Hive table to a location in Amazon S3, do not map it to the root path of the bucket, s3://mybucket, as this may cause errors when Hive writes the data to Amazon S3. Instead map the table to a subpath of the bucket, s3://mybucket/mypath.

**Exporting Data from DynamoDB**

You can use Hive to export data from DynamoDB.

**To export a DynamoDB table to an Amazon S3 bucket**

- Create a Hive table that references data stored in DynamoDB. Then you can call the `INSERT OVERWRITE` command to write the data to an external directory. In the following example, `s3://bucketname/path/subpath/` is a valid path in Amazon S3. Adjust the columns and datatypes in the `CREATE` command to match the values in your DynamoDB. You can use this to create an archive of your DynamoDB data in Amazon S3.

```
CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 array<string>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1", "dynamodb.column.mapping" = "col1:name, col2:year, col3:holidays");
```

```
INSERT OVERWRITE DIRECTORY 's3://bucketname/path/subpath/' SELECT * FROM hiveTableName;
```
To export a DynamoDB table to an Amazon S3 bucket using formatting

- Create an external table that references a location in Amazon S3. This is shown below as s3_export. During the CREATE call, specify row formatting for the table. Then, when you use INSERT OVERWRITE to export data from DynamoDB to s3_export, the data is written out in the specified format. In the following example, the data is written out as comma-separated values (CSV).

```
CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 array<string>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1", "dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");

CREATE EXTERNAL TABLE s3_export (a_col string, b_col bigint, c_col array<string>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT * FROM hiveTableName;
```

To export a DynamoDB table to an Amazon S3 bucket without specifying a column mapping

- Create a Hive table that references data stored in DynamoDB. This is similar to the preceding example, except that you are not specifying a column mapping. The table must have exactly one column of type map<string, string>. If you then create an EXTERNAL table in Amazon S3 you can call the INSERT OVERWRITE command to write the data from DynamoDB to Amazon S3. You can use this to create an archive of your DynamoDB data in Amazon S3. Because there is no column mapping, you cannot query tables that are exported this way. Exporting data without specifying a column mapping is available in Hive 0.8.1.5 or later, which is supported on Amazon EMR AMI 2.2.x and later.

```
CREATE EXTERNAL TABLE hiveTableName (item map<string,string>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1");

CREATE EXTERNAL TABLE s3TableName (item map<string, string>) ROW FORMAT DELIMITED FIELDS TERMINATED BY '	' LINES TERMINATED BY '\n' LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3TableName SELECT * FROM hiveTableName;
```

To export a DynamoDB table to an Amazon S3 bucket using data compression

- Hive provides several compression codecs you can set during your Hive session. Doing so causes the exported data to be compressed in the specified format. The following example compresses the exported files using the Lempel-Ziv-Oberhumer (LZO) algorithm.

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 array<string>)
```
A store by 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1",
"dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;

The available compression codecs are:
- org.apache.hadoop.io.compress.GzipCodec
- org.apache.hadoop.io.compress.DefaultCodec
- com.hadoop.compression.lzo.LzoCodec
- com.hadoop.compression.lzo.LzopCodec
- org.apache.hadoop.io.compress.BZip2Codec
- org.apache.hadoop.io.compress.SnappyCodec

To export a DynamoDB table to HDFS
- Use the following Hive command, where hdfs:///directoryName is a valid HDFS path and hiveTableName is a table in Hive that references DynamoDB. This export operation is faster than exporting a DynamoDB table to Amazon S3 because Hive 0.7.1.1 uses HDFS as an intermediate step when exporting data to Amazon S3. The following example also shows how to set dynamodb.throughput.read.percent to 1.0 in order to increase the read request rate.

CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 array<string>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1",
"dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");

SET dynamodb.throughput.read.percent=1.0;

INSERT OVERWRITE DIRECTORY 'hdfs:///directoryName' SELECT * FROM hiveTableName;

You can also export data to HDFS using formatting and compression as shown above for the export to Amazon S3. To do so, simply replace the Amazon S3 directory in the examples above with an HDFS directory.

To read non-printable UTF-8 character data in Hive
- You can read and write non-printable UTF-8 character data with Hive by using the STORED AS SEQUENCEFILE clause when you create the table. A SequenceFile is Hadoop binary file format; you need to use Hadoop to read this file. The following example shows how to export data from DynamoDB into Amazon S3. You can use this functionality to handle non-printable UTF-8 encoded characters.

CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 array<string>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1",
"dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");
CREATE EXTERNAL TABLE `s3_export`(`a_col` string, `b_col` bigint, `c_col` array<string>) STORED AS SEQUENCEFILE LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE `s3_export` SELECT * FROM `hiveTableName`;

Importing Data to DynamoDB

When you write data to DynamoDB using Hive you should ensure that the number of write capacity units is greater than the number of mappers in the cluster. For example, clusters that run on m1.xlarge EC2 instances produce 8 mappers per instance. In the case of a cluster that has 10 instances, that would mean a total of 80 mappers. If your write capacity units are not greater than the number of mappers in the cluster, the Hive write operation may consume all of the write throughput, or attempt to consume more throughput than is provisioned. For more information about the number of mappers produced by each EC2 instance type, go to Configure Hadoop (p. 47). There, you will find a "Task Configuration" section for each of the supported configurations.

The number of mappers in Hadoop are controlled by the input splits. If there are too few splits, your write command might not be able to consume all the write throughput available.

If an item with the same key exists in the target DynamoDB table, it will be overwritten. If no item with the key exists in the target DynamoDB table, the item is inserted.

To import a table from Amazon S3 to DynamoDB

• You can use Amazon EMR (Amazon EMR) and Hive to write data from Amazon S3 to DynamoDB.

CREATE EXTERNAL TABLE `s3_import`(`a_col` string, `b_col` bigint, `c_col` array<string>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION 's3://bucketname/path/subpath/';

CREATE EXTERNAL TABLE `hiveTableName`(`col1` string, `col2` bigint, `col3` array<string>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1", "dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");

INSERT OVERWRITE TABLE `hiveTableName` SELECT * FROM `s3_import`;

To import a table from an Amazon S3 bucket to DynamoDB without specifying a column mapping

• Create an EXTERNAL table that references data stored in Amazon S3 that was previously exported from DynamoDB. Before importing, ensure that the table exists in DynamoDB and that it has the same key schema as the previously exported DynamoDB table. In addition, the table must have exactly one column of type map<string, string>. If you then create a Hive table that is linked to DynamoDB, you can call the INSERT OVERWRITE command to write the data from Amazon S3 to DynamoDB. Because there is no column mapping, you cannot query tables that are imported this way. Importing data without specifying a column mapping is available in Hive 0.8.1.5 or later, which is supported on Amazon EMR AMI 2.2.3 and later.

CREATE EXTERNAL TABLE `s3TableName` (item map<string, string>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

CREATE EXTERNAL TABLE hiveTableName (item map<string,string>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1");

INSERT OVERWRITE TABLE hiveTableName SELECT *
FROM s3TableName;

To import a table from HDFS to DynamoDB

- You can use Amazon EMR and Hive to write data from HDFS to DynamoDB.

CREATE EXTERNAL TABLE hdfs_import
(a_col string, b_col bigint, c_col array<string>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 'hdfs:///directoryName';

CREATE EXTERNAL TABLE hiveTableName
(col1 string, col2 bigint, col3 array<string>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "dynamodbtable1",
"dynamodb.column.mapping" = "col1:name,col2:year,col3:holidays");

INSERT OVERWRITE TABLE hiveTableName SELECT * FROM hdfs_import;

Querying Data in DynamoDB

The following examples show the various ways you can use Amazon EMR to query data stored in DynamoDB.

To find the largest value for a mapped column (max)

- Use Hive commands like the following. In the first command, the CREATE statement creates a Hive table that references data stored in DynamoDB. The SELECT statement then uses that table to query data stored in DynamoDB. The following example finds the largest order placed by a given customer.

CREATE EXTERNAL TABLE hive_purchases
(customerId bigint, total_cost double, items_purchased array<String>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "Purchases",
"dynamodb.column.mapping" = "customerId:CustomerId,total_cost:Cost,items_purchased:Items");

SELECT max(total_cost) from hive_purchases where customerId = 717;

To aggregate data using the GROUP BY clause

- You can use the GROUP BY clause to collect data across multiple records. This is often used with an aggregate function such as sum, count, min, or max. The following example returns a list of the largest orders from customers who have placed more than three orders.

SELECT customerId, sum(total_cost) as total from hive_purchases group by customerId having count(*) > 3;
CREATE EXTERNAL TABLE hive_purchases(customerId bigint, total_cost double, items_purchased array<String>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "Purchases", "dynamodb.column.mapping" = "customerId:CustomerId,total_cost:Cost,items_purchased:Items");
SELECT customerId, max(total_cost) from hive_purchases GROUP BY customerId HAVING count(*) > 3;

To join two DynamoDB tables

- The following example maps two Hive tables to data stored in DynamoDB. It then calls a join across those two tables. The join is computed on the cluster and returned. The join does not take place in DynamoDB. This example returns a list of customers and their purchases for customers that have placed more than two orders.

CREATE EXTERNAL TABLE hive_purchases(customerId bigint, total_cost double, items_purchased array<String>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "Purchases", "dynamodb.column.mapping" = "customerId:CustomerId,total_cost:Cost,items_purchased:Items");
CREATE EXTERNAL TABLE hive_customers(customerId bigint, customerName string, customerAddress array<String>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "Customers", "dynamodb.column.mapping" = "customerId:CustomerId,customerName:Name,customerAddress:Address");
SELECT c.customerId, c.customerName, count(*) as count from hive_customers c JOIN hive_purchases p ON c.customerId=p.customerId GROUP BY c.customerId, c.customerName HAVING count > 2;

To join two tables from different sources

- In the following example, Customer_S3 is a Hive table that loads a CSV file stored in Amazon S3 and hive_purchases is a table that references data in DynamoDB. The following example joins together customer data stored as a CSV file in Amazon S3 with order data stored in DynamoDB to return a set of data that represents orders placed by customers who have "Miller" in their name.

CREATE EXTERNAL TABLE hive_purchases(customerId bigint, total_cost double, items_purchased array<String>) STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' TBLPROPERTIES ("dynamodb.table.name" = "Purchases", "dynamodb.column.mapping" = "customerId:CustomerId,total_cost:Cost,items_purchased:Items");
CREATE EXTERNAL TABLE Customer_S3(customerId bigint, customerName string, customerAddress array<String>) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION 's3://bucketname/path/subpath/';
SELECT c.customerId, c.customerName, c.customerAddress from Customer_S3 c
JOIN hive_purchases p 
ON c.customerid=p.customerid 
where c.customerName like '%Miller%';

**Note**
In the preceding examples, the CREATE TABLE statements were included in each example for clarity and completeness. When running multiple queries or export operations against a given Hive table, you only need to create the table one time, at the beginning of the Hive session.

### Optimizing Performance for Amazon EMR Operations in DynamoDB

Amazon EMR operations on a DynamoDB table count as read operations, and are subject to the table's provisioned throughput settings. Amazon EMR implements its own logic to try to balance the load on your DynamoDB table to minimize the possibility of exceeding your provisioned throughput. At the end of each Hive query, Amazon EMR returns information about the cluster used to process the query, including how many times your provisioned throughput was exceeded. You can use this information, as well as CloudWatch metrics about your DynamoDB throughput, to better manage the load on your DynamoDB table in subsequent requests.

The following factors influence Hive query performance when working with DynamoDB tables.

**Provisioned Read Capacity Units**

When you run Hive queries against a DynamoDB table, you need to ensure that you have provisioned a sufficient amount of read capacity units.

For example, suppose that you have provisioned 100 units of Read Capacity for your DynamoDB table. This will let you perform 100 reads, or 409,600 bytes, per second. If that table contains 20GB of data (21,474,836,480 bytes), and your Hive query performs a full table scan, you can estimate how long the query will take to run:

\[
21,474,836,480 / 409,600 = 52,429 \text{ seconds} = 14.56 \text{ hours}
\]

The only way to decrease the time required would be to adjust the read capacity units on the source DynamoDB table. Adding more nodes to the Amazon EMR cluster will not help.

In the Hive output, the completion percentage is updated when one or more mapper processes are finished. For a large DynamoDB table with a low provisioned Read Capacity setting, the completion percentage output might not be updated for a long time; in the case above, the job will appear to be 0% complete for several hours. For more detailed status on your job's progress, go to the Amazon EMR console; you will be able to view the individual mapper task status, and statistics for data reads.

You can also log on to Hadoop interface on the master node and see the Hadoop statistics. This will show you the individual map task status and some data read statistics. For more information, see the following topics:

- [Web Interfaces Hosted on the Master Node](#)

**Read Percent Setting**

By default, Amazon EMR manages the request load against your DynamoDB table according to your current provisioned throughput. However, when Amazon EMR returns information about your job that includes a high number of provisioned throughput exceeded responses, you can adjust the default read
rate using the `dynamodb.throughput.read.percent` parameter when you set up the Hive table. For more information about setting the read percent parameter, see `Hive Options (p. 181)`.

**Write Percent Setting**

By default, Amazon EMR manages the request load against your DynamoDB table according to your current provisioned throughput. However, when Amazon EMR returns information about your job that includes a high number of provisioned throughput exceeded responses, you can adjust the default write rate using the `dynamodb.throughput.write.percent` parameter when you set up the Hive table. For more information about setting the write percent parameter, see `Hive Options (p. 181)`.

**Retry Duration Setting**

By default, Amazon EMR re-runs a Hive query if it has not returned a result within two minutes, the default retry interval. You can adjust this interval by setting the `dynamodb.retry.duration` parameter when you run a Hive query. For more information about setting the write percent parameter, see `Hive Options (p. 181)`.

**Number of Map Tasks**

The mapper daemons that Hadoop launches to process your requests to export and query data stored in DynamoDB are capped at a maximum read rate of 1 MiB per second to limit the read capacity used. If you have additional provisioned throughput available on DynamoDB, you can improve the performance of Hive export and query operations by increasing the number of mapper daemons. To do this, you can either increase the number of EC2 instances in your cluster or increase the number of mapper daemons running on each EC2 instance.

You can increase the number of EC2 instances in a cluster by stopping the current cluster and re-launching it with a larger number of EC2 instances. You specify the number of EC2 instances in the `Configure EC2 Instances` dialog box if you're launching the cluster from the Amazon EMR console, or with the `--num-instances` option if you're launching the cluster from the CLI.

The number of map tasks run on an instance depends on the EC2 instance type. For more information about the supported EC2 instance types and the number of mappers each one provides, see `Task Configuration (p. 66)`. There, you will find a "Task Configuration" section for each of the supported configurations.

```json
{
    "configurations": [
    {
        "classification": "mapred-site",
        "properties": {
            "mapred.tasktracker.map.tasks.maximum": "10"
        }
    }
    ]
}
```

**Parallel Data Requests**

Multiple data requests, either from more than one user or more than one application to a single table may drain read provisioned throughput and slow performance.

**Process Duration**

Data consistency in DynamoDB depends on the order of read and write operations on each node. While a Hive query is in progress, another application might load new data into the DynamoDB table or modify
or delete existing data. In this case, the results of the Hive query might not reflect changes made to the data while the query was running.

**Avoid Exceeding Throughput**

When running Hive queries against DynamoDB, take care not to exceed your provisioned throughput, because this will deplete capacity needed for your application's calls to `DynamoDB::Get`. To ensure that this is not occurring, you should regularly monitor the read volume and throttling on application calls to `DynamoDB::Get` by checking logs and monitoring metrics in Amazon CloudWatch.

**Request Time**

Scheduling Hive queries that access a DynamoDB table when there is lower demand on the DynamoDB table improves performance. For example, if most of your application's users live in San Francisco, you might choose to export daily data at 4 a.m. PST, when the majority of users are asleep, and not updating records in your DynamoDB database.

**Time-Based Tables**

If the data is organized as a series of time-based DynamoDB tables, such as one table per day, you can export the data when the table becomes no longer active. You can use this technique to back up data to Amazon S3 on an ongoing fashion.

**Archived Data**

If you plan to run many Hive queries against the data stored in DynamoDB and your application can tolerate archived data, you may want to export the data to HDFS or Amazon S3 and run the Hive queries against a copy of the data instead of DynamoDB. This conserves your read operations and provisioned throughput.

---

**Kinesis**

Amazon EMR clusters can read and process Amazon Kinesis streams directly, using familiar tools in the Hadoop ecosystem such as Hive, Pig, MapReduce, the Hadoop Streaming API, and Cascading. You can also join real-time data from Amazon Kinesis with existing data on Amazon S3, Amazon DynamoDB, and HDFS in a running cluster. You can directly load the data from Amazon EMR to Amazon S3 or DynamoDB for post-processing activities. For information about Amazon Kinesis service highlights and pricing, see Amazon Kinesis.

**What Can I Do With Amazon EMR and Amazon Kinesis Integration?**

Integration between Amazon EMR and Amazon Kinesis makes certain scenarios much easier; for example:

- **Streaming log analysis**—You can analyze streaming web logs to generate a list of top 10 error types every few minutes by region, browser, and access domain.

- **Customer engagement**—You can write queries that join clickstream data from Amazon Kinesis with advertising campaign information stored in a DynamoDB table to identify the most effective categories of ads that are displayed on particular websites.

- **Ad-hoc interactive queries**—You can periodically load data from Amazon Kinesis streams into HDFS and make it available as a local Impala table for fast, interactive, analytic queries.
Checkpointed Analysis of Amazon Kinesis Streams

Users can run periodic, batched analysis of Amazon Kinesis streams in what are called *iterations*. Because Amazon Kinesis stream data records are retrieved by using a sequence number, iteration boundaries are defined by starting and ending sequence numbers that Amazon EMR stores in a DynamoDB table. For example, when *iteration0* ends, it stores the ending sequence number in DynamoDB so that when the *iteration1* job begins, it can retrieve subsequent data from the stream. This mapping of iterations in stream data is called *checkpointing*. For more information, see Kinesis Connector.

If an iteration was checkpointed and the job failed processing an iteration, Amazon EMR attempts to reprocess the records in that iteration, provided that the data records have not reached the 24-hour limit for Amazon Kinesis streams.

Checkpointing is a feature that allows you to:

- Start data processing after a sequence number processed by a previous query that ran on same stream and logical name
- Re-process the same batch of data from Kinesis that was processed by an earlier query

To enable checkpointing, set the `kinesis.checkpoint.enabled` parameter to `true` in your scripts. Also, configure the following parameters:

<table>
<thead>
<tr>
<th>Configuration Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kinesis.checkpoint.metastore.table.name</code></td>
<td>DynamoDB table name where checkpoint information will be stored</td>
</tr>
<tr>
<td><code>kinesis.checkpoint.metastore.hash.key.name</code></td>
<td>Hash key name for the DynamoDB table</td>
</tr>
<tr>
<td><code>kinesis.checkpoint.metastore.hash.range.name</code></td>
<td>Range key name for the DynamoDB table</td>
</tr>
<tr>
<td><code>kinesis.checkpoint.logical.name</code></td>
<td>A logical name for current processing</td>
</tr>
<tr>
<td><code>kinesis.checkpoint.iteration.no</code></td>
<td>Iteration number for processing associated with the logical name</td>
</tr>
<tr>
<td><code>kinesis.rerun.iteration.without.wait</code></td>
<td>Boolean value that indicates if a failed iteration can be rerun without waiting for timeout; the default is false</td>
</tr>
</tbody>
</table>

Provisioned IOPS Recommendations for Amazon DynamoDB Tables

The Amazon EMR connector for Amazon Kinesis uses the DynamoDB database as its backing for checkpointing metadata. You must create a table in DynamoDB before consuming data in an Amazon Kinesis stream with an Amazon EMR cluster in checkpointed intervals. The table must be in the same region as your Amazon EMR cluster. The following are general recommendations for the number of IOPS you should provision for your DynamoDB tables; let $j$ be the maximum number of Hadoop jobs (with different logical name+iteration number combination) that can run concurrently and $s$ be the maximum number of shards that any job will process:

- **For Read Capacity Units**: $j \times s / 5$
- **For Write Capacity Units**: $j \times s$
Performance Considerations

Amazon Kinesis shard throughput is directly proportional to the instance size of nodes in Amazon EMR clusters and record size in the stream. We recommend that you use m1.xlarge or larger instances on master and core nodes for production workloads.

Schedule Amazon Kinesis Analysis with Amazon EMR

When you are analyzing data on an active Amazon Kinesis stream, limited by timeouts and a maximum duration for any iteration, it is important that you run the analysis frequently to gather periodic details from the stream. There are multiple ways to execute such scripts and queries at periodic intervals; we recommend using AWS Data Pipeline for recurrent tasks like these. For more information, see AWS Data Pipeline PigActivity and AWS Data Pipeline HiveActivity in the AWS Data Pipeline Developer Guide.

S3DistCp (s3-dist-cp)

Apache DistCp is an open-source tool you can use to copy large amounts of data. S3DistCp is an extension of DistCp that is optimized to work with AWS, particularly Amazon S3. The command for S3DistCp in Amazon EMR version 4.0 and later is s3-dist-cp, which you add as a step in a cluster or at the command line. Using S3DistCp, you can efficiently copy large amounts of data from Amazon S3 into HDFS where it can be processed by subsequent steps in your Amazon EMR cluster. You can also use S3DistCp to copy data between Amazon S3 buckets or from HDFS to Amazon S3. S3DistCp is more scalable and efficient for parallel copying large numbers of objects across buckets and across AWS accounts.

For specific commands that demonstrate the flexibility of S3DistCp in real-world scenarios, see Seven Tips for Using S3DistCp on the AWS Big Data blog.

Like DistCp, S3DistCp uses MapReduce to copy in a distributed manner, sharing the copy, error handling, recovery, and reporting tasks across several servers. For more information about the Apache DistCp open source project, see the DistCp Guide in Apache Hadoop documentation.

During a copy operation, S3DistCp stages a temporary copy of the output in HDFS on the cluster. There must be enough free space in HDFS to stage the data, otherwise the copy operation fails. In addition, if S3DistCp fails, it does not clean the temporary HDFS directory, so you must purge the temporary files manually. For example, if you copy 500 GB of data from HDFS to S3, S3DistCp copies the entire 500 GB into a temporary directory in HDFS, then uploads the data to Amazon S3 from the temporary directory. When the copy is complete, S3DistCp removes the files from the temporary directory. If you only have 250 GB of space remaining in HDFS prior to the copy, the copy operation fails.

If S3DistCp is unable to copy some or all of the specified files, the cluster step fails and returns a non-zero error code. If this occurs, S3DistCp does not clean up partially copied files.

Important

S3DistCp does not support Amazon S3 bucket names that contain the underscore character.

S3DistCp Options

When you call S3DistCp, you can specify options that change how it copies and compresses data. These are described in the following table. The options are added to the step using the arguments list. Examples of the S3DistCp arguments are shown in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>--src=LOCATION</td>
<td>Location of the data to copy. This can be either an HDFS or Amazon S3 location.</td>
<td>Yes</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Option: <code>--src=LOCATION</code></td>
<td>Destination for the data. This can be either an HDFS or Amazon S3 location.</td>
<td>Yes</td>
</tr>
<tr>
<td>Option: <code>--srcPattern=PATTERN</code></td>
<td>A regular expression that filters the copy operation to a subset of the data at <code>--src</code>. If neither <code>--srcPattern</code> nor <code>--groupBy</code> is specified, all data at <code>--src</code> is copied to <code>--dest</code>.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Important: S3DistCp does not support Amazon S3 bucket names that contain the underscore character.</td>
<td></td>
</tr>
<tr>
<td>Option: <code>--groupBy=PATTERN</code></td>
<td>A regular expression that causes S3DistCp to concatenate files that match the expression. For example, you could use this option to combine all of the log files written in one hour into a single file. The concatenated filename is the value matched by the regular expression for the grouping.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Important: S3DistCp does not support Amazon S3 bucket names that contain the underscore character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parentheses indicate how files should be grouped, with all of the items that match the parenthetical statement being combined into a single output file. If the regular expression does not include a parenthetical statement, the cluster fails on the S3DistCp step and return an error.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Important: S3DistCp does not support Amazon S3 bucket names that contain the underscore character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the regular expression argument contains special characters, such as an asterisk (*), either the regular expression or the entire <code>--args</code> string must be enclosed in single quotes (').</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: <code>--srcPattern=.*daemons.*-hadoop-.*</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When <code>--groupBy</code> is specified, only files that match the specified pattern are copied. You do not need to specify <code>--groupBy</code> and <code>--srcPattern</code> at the same time.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: <code>--groupBy=.*subnetid.*(0-9)[0-9]+.*</code></td>
<td></td>
</tr>
</tbody>
</table>
### S3DistCp Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--targetSize=SIZE</code></td>
<td>The size, in mebibytes (MiB), of the files to create based on the <code>--groupBy</code> option. This value must be an integer. When <code>--targetSize</code> is set, S3DistCp attempts to match this size; the actual size of the copied files may be larger or smaller than this value. Jobs are aggregated based on the size of the data file, thus it is possible that the target file size will match the source data file size. If the files concatenated by <code>--groupBy</code> are larger than the value of <code>--targetSize</code>, they are broken up into part files, and named sequentially with a numeric value appended to the end. For example, a file concatenated into <code>myfile.gz</code> would be broken into parts as: <code>myfile0.gz</code>, <code>myfile1.gz</code>, etc. Example: <code>--targetSize=2</code></td>
<td>No</td>
</tr>
<tr>
<td><code>--appendToLastFile</code></td>
<td>Specifies the behavior of S3DistCp when copying to files from Amazon S3 to HDFS which are already present. It appends new file data to existing files. If you use <code>--appendToLastFile</code> with <code>--groupBy</code>, new data is appended to files which match the same groups. This option also respects the <code>--targetSize</code> behavior when used with <code>--groupBy</code>.</td>
<td>No</td>
</tr>
<tr>
<td><code>--outputCodec=CODEC</code></td>
<td>Specifies the compression codec to use for the copied files. This can take the values: gzip, gz, lzo, snappy, or none. You can use this option, for example, to convert input files compressed with Gzip into output files with LZO compression, or to uncompress the files as part of the copy operation. If you choose an output codec, the filename will be appended with the appropriate extension (e.g. for gz and gzip, the extension is .gz) If you do not specify a value for <code>--outputCodec</code>, the files are copied over with no change in their compression. Example: <code>--outputCodec=lzo</code></td>
<td>No</td>
</tr>
<tr>
<td><code>--s3ServerSideEncryption</code></td>
<td>Ensures that the target data is transferred using SSL and automatically encrypted in Amazon S3 using an AWS service-side key. When retrieving data using S3DistCp, the objects are automatically unencrypted. If you attempt to copy an unencrypted object to an encryption-required Amazon S3 bucket, the operation fails. For more information, see Using Data Encryption. Example: <code>--s3ServerSideEncryption</code></td>
<td>No</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>--deleteOnSuccess</td>
<td>If the copy operation is successful, this option causes S3DistCp to delete the copied files from the source location. This is useful if you are copying output files, such as log files, from one location to another as a scheduled task, and you don't want to copy the same files twice. Example: --deleteOnSuccess</td>
<td>No</td>
</tr>
<tr>
<td>--disableMultipartUpload</td>
<td>Disables the use of multipart upload. Example: --disableMultipartUpload</td>
<td>No</td>
</tr>
<tr>
<td>--multipartUploadChunkSize=SIZE</td>
<td>The size, in MiB, of the multipart upload part size. By default, it uses multipart upload when writing to Amazon S3. The default chunk size is 16 MiB. Example: --multipartUploadChunkSize=32</td>
<td>No</td>
</tr>
<tr>
<td>--numberFiles</td>
<td>Prepends output files with sequential numbers. The count starts at 0 unless a different value is specified by --startingIndex. Example: --numberFiles</td>
<td>No</td>
</tr>
<tr>
<td>--startingIndex=INDEX</td>
<td>Used with --numberFiles to specify the first number in the sequence. Example: --startingIndex=1</td>
<td>No</td>
</tr>
<tr>
<td>--outputManifest=FILENAME</td>
<td>Creates a text file, compressed with Gzip, that contains a list of all the files copied by S3DistCp. Example: --outputManifest=manifest-1.gz</td>
<td>No</td>
</tr>
<tr>
<td>--previousManifest=PATH</td>
<td>Reads a manifest file that was created during a previous call to S3DistCp using the --outputManifest flag. When the --previousManifest flag is set, S3DistCp excludes the files listed in the manifest from the copy operation. If --outputManifest is specified along with --previousManifest, files listed in the previous manifest also appear in the new manifest file, although the files are not copied. Example: --previousManifest=/usr/bin/manifest-1.gz</td>
<td>No</td>
</tr>
<tr>
<td>--requirePreviousManifest</td>
<td>Requires a previous manifest created during a previous call to S3DistCp. If this is set to false, no error is generated when a previous manifest is not specified. The default is true.</td>
<td>No</td>
</tr>
</tbody>
</table>
Adding S3DistCp as a Step in a Cluster

You can call S3DistCp by adding it as a step in your cluster. Steps can be added to a cluster at launch or to a running cluster using the console, CLI, or API. The following examples demonstrate adding an S3DistCp step to a running cluster. For more information on adding steps to a cluster, see Submit Work to a Cluster.

To add an S3DistCp step to a running cluster using the AWS CLI

For more information on using Amazon EMR commands in the AWS CLI, see http://docs.aws.amazon.com/cli/latest/reference/emr.
To add a step to a cluster that calls S3DistCp, pass the parameters that specify how S3DistCp should perform the copy operation as arguments.

The following example copies daemon logs from Amazon S3 to hdfs:///output. In the following command:

- `--cluster-id` specifies the cluster
- `Jar` is the location of the S3DistCp JAR file
- `Args` is a comma-separated list of the option name-value pairs to pass in to S3DistCp. For a complete list of the available options, see S3DistCp Options (p. 193).

To add an S3DistCp copy step to a running cluster, put the following in a JSON file saved in Amazon S3 or your local file system as `myStep.json` for this example. Replace `j-3GYXXXXXXX9IOK` with your cluster ID and replace `mybucket` with your Amazon S3 bucket name.

```json
{
  "Name":"S3DistCp step",
  "Args":["s3-dist-cp","--s3Endpoint=s3.amazonaws.com","--src=s3://mybucket/logs/j-3GYXXXXXXX9IOJ/node/","--dest=hdfs:///output","--srcPattern=.*[a-zA-Z,]++"],
  "ActionOnFailure":"CONTINUE",
  "Type":"CUSTOM_JAR",
  "Jar":"command-runner.jar"
}
```

```
aws emr add-steps --cluster-id j-3GYXXXXXXX9IOK --steps file://./myStep.json
```

**Example Copy log files from Amazon S3 to HDFS**

This example also illustrates how to copy log files stored in an Amazon S3 bucket into HDFS by adding a step to a running cluster. In this example the `--srcPattern` option is used to limit the data copied to the daemon logs.

To copy log files from Amazon S3 to HDFS using the `--srcPattern` option, put the following in a JSON file saved in Amazon S3 or your local file system as `myStep.json` for this example. Replace `j-3GYXXXXXXX9IOK` with your cluster ID and replace `mybucket` with your Amazon S3 bucket name.

```json
{
  "Name":"S3DistCp step",
  "Args":["s3-dist-cp","--s3Endpoint=s3.amazonaws.com","--src=s3://mybucket/logs/j-3GYXXXXXXX9IOJ/node/","--dest=hdfs:///output","--srcPattern=.*daemons.*-hadoop-*"],
  "ActionOnFailure":"CONTINUE",
  "Type":"CUSTOM_JAR",
  "Jar":"command-runner.jar"
}
```

```
aws emr add-steps --cluster-id j-3GYXXXXXXX9IOK --steps file://./myStep.json
```
Command Runner

Many scripts or programs are placed on the shell login path environment so you do not need to specify the full path when executing them when using command-runner.jar. You also do not have to know the full path to command-runner.jar. command-runner.jar is located on the AMI so there is no need to know a full URI as was the case with script-runner.jar.

The following is a list of scripts that can be executed with command-runner.jar:

- hadoop-streaming
  Submit a Hadoop streaming program. In the console and some SDKs, this is a streaming step.
- hive-script
  Run a Hive script. In the console and SDKs, this is a Hive step.
- pig-script
  Run a Pig script. In the console and SDKs, this is a Pig step.
- spark-submit
  Run a Spark application. In the console, this is a Spark step.
- s3-dist-cp
  Distributed copy large amounts of data from Amazon S3 into HDFS.
- hadoop-lzo
  Run the Hadoop LZO indexer on a directory.

The following is an example usage of command-runner.jar using the AWS CLI:

```
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF --steps Name="Command Runner",Jar="command-runner.jar",Args=["spark-submit","Args..."]
```
This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

The following are links to all versions of Amazon EMR Release Guide:

- Release 5.11.1 and 5.11.0 (this guide)
- Release 5.10.0
- Release 5.9.0
- Release 5.8.1 and 5.8.0
- Release 5.7.0
- Release 5.6.0
- Release 5.5.1 and 5.5.0
- Release 5.4.0
- Release 5.3.0
- Release 5.2.2
- Release 5.2.1
- Release 5.2.0
- Release 5.1.0
- Release 5.0.3
- Release 5.0.0
- Release 4.9.3 and 4.9.2
- Release 4.9.1
- Release 4.8.4
- Release 4.8.3
- Release 4.8.2
- Release 4.8.0
- Release 4.7.2
- Release 4.7.1
- Release 4.7
- Release 4.6
- Release 4.5
- Release 4.4
- Release 4.3
- Release 4.2
- Release 4.1
- Release 4.0
# Document History

The following table describes the important changes to the documentation after the last release of Amazon EMR.

**API version:** 2009-03-31

**Latest documentation update:** December 21, 2017

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon EMR Release 5.10.0</td>
<td>This guide supports the Amazon EMR 5.11.0 release.</td>
<td>December 21, 2017</td>
</tr>
</tbody>
</table>