Amazon EMR

Amazon EMR Release Guide

Amazon EMR: Amazon EMR Release Guide

Table of Contents

About Amazon EMR Releases

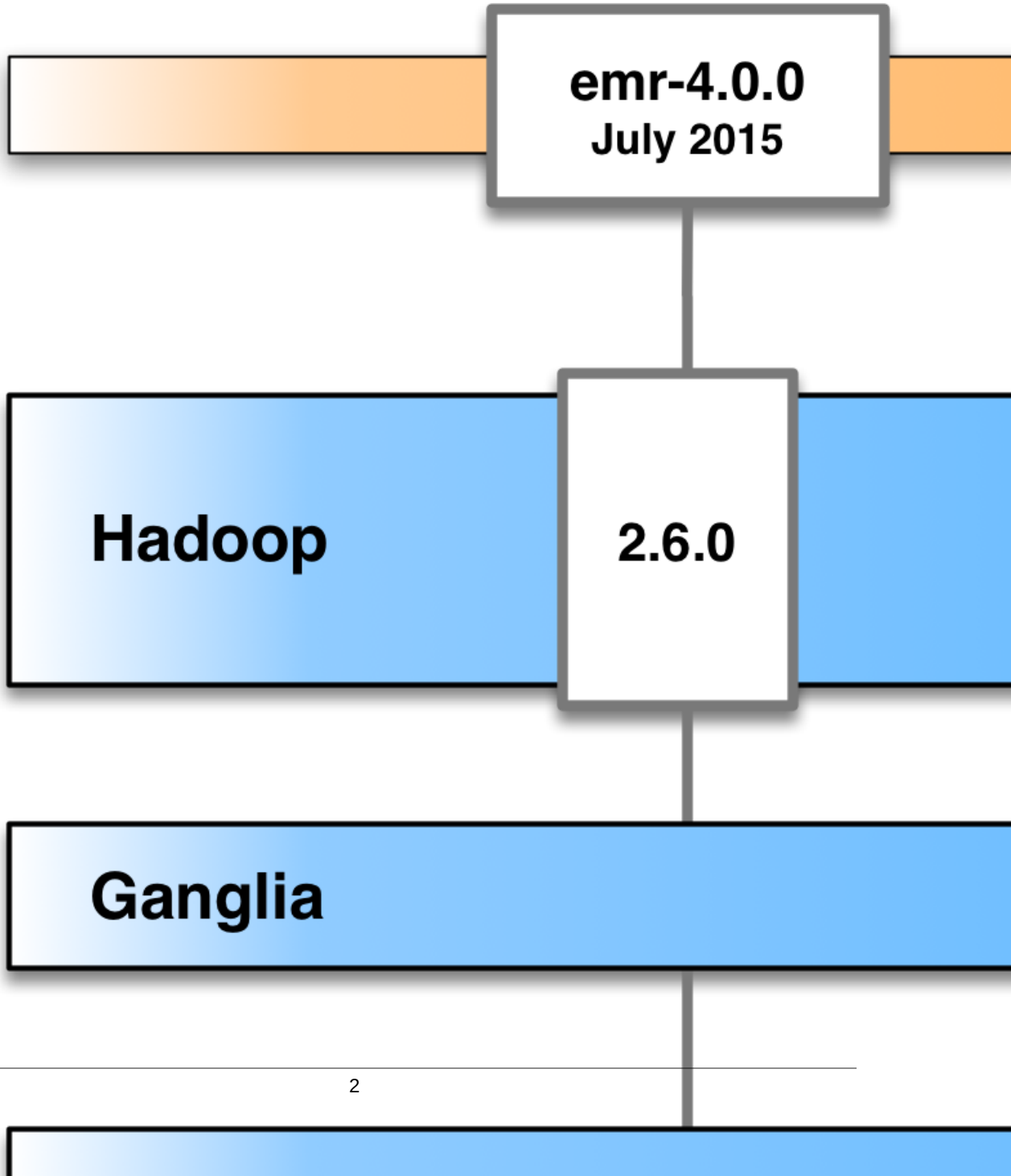
This documentation is for versions 4.x and 5.x of Amazon EMR. For information about Amazon EMR AMI versions 2.x and 3.x, see the Amazon EMR Developer Guide (PDF).

This document provides information about Amazon EMR 4.x and 5.x software releases. A release is a set of software applications and components which 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 which enable cluster and resource management, interoperability with other AWS services, and additional configuration optimizations for installed software.

Topics

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, Mahout, Oozie, Phoenix, Pig, Presto, Spark, Sqoop, Tez, Zeppelin, and ZooKeeper.


emr-4.0.0
July 2015
Hadoop
2.6.0
Ganglia

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:

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. 89).

The following components are included with Amazon EMR:

Component	Version	Description
emr-ddb	4.2.0	Amazon DynamoDB connector for Hadoop ecosystem applications.
emr-goodies	2.2.0	Extra convenience libraries for the Hadoop ecosystem.
emr-kinesis	3.2.0	Amazon Kinesis connector for Hadoop ecosystem applications.
emr-s3-dist-cp	2.4.0	Distributed copy application optimized for Amazon S3.
emrfs	2.13.0	Amazon S3 connector for Hadoop ecosystem applications.
flink-client	1.1.3	Apache Flink command line client scripts and applications.
ganglia-monitor	3.7.2	Embedded Ganglia agent for Hadoop ecosystem applications along with the Ganglia monitoring agent.
ganglia-metadata-collector	3.7.2	Ganglia metadata collector for aggregating metrics from Ganglia monitoring agents.
ganglia-web	3.7.1	Web application for viewing metrics collected by the Ganglia metadata collector.
hadoop-client	2.7.3-amzn-1	Hadoop command-line clients such as 'hdfs', 'hadoop', or 'yarn'.

Component	Version	Description
hadoop-hdfs-datanode	2.7.3-amzn-1	HDFS node-level service for storing blocks.
hadoop-hdfs-library	2.7.3-amzn-1	HDFS command-line client and library
hadoop-hdfs-namenode	2.7.3-amzn-1	HDFS service for tracking file names and block locations.
hadoop-httpfs-server	2.7.3-amzn-1	HTTP endpoint for HDFS operations.
hadoop-kms-server	2.7.3-amzn-1	Cryptographic key management server based on Hadoop's KeyProvider API.
hadoop-mapred	2.7.3-amzn-1	MapReduce execution engine libraries for running a MapReduce application.
hadoop-yarn-nodemanager	2.7.3-amzn-1	YARN service for managing containers on an individual node.
hadoop-yarn-resourcemanager	2.7.3-amzn-1	YARN service for allocating and managing cluster resources and distributed applications.
hadoop-yarn-timeline-server	2.7.3-amzn-1	Service for retrieving current and historical information for YARN applications.
hbase-hmaster	1.2.3	Service for an HBase cluster responsible for coordination of Regions and execution of administrative commands.
hbase-region-server	1.2.3	Service for serving one or more HBase regions.
hbase-client	1.2.3	HBase command-line client.
hbase-rest-server	1.2.3	Service providing a RESTful HTTP endpoint for HBase.
hbase-thrift-server	1.2.3	Service providing a Thrift endpoint to HBase.
hcatalog-client	2.1.0-amzn-0	The 'hcat' command line client for manipulating hcatalog-server.
hcatalog-server	2.1.0-amzn-0	Service providing HCatalog, a table and storage management layer for distributed applications.
hcatalog-webhcat-server	2.1.0-amzn-0	HTTP endpoint providing a REST interface to HCatalog.
hive-client	2.1.0-amzn-0	Hive command line client.

Component	Version	Description
hive-metastore-server	2.1.0-amzn-0	Service for accessing the Hive metastore, a semantic repository storing metadata for SQL on Hadoop operations.
hive-server	2.1.0-amzn-0	Service for accepting Hive queries as web requests.
hue-server	3.10.0-amzn-0	Web application for analyzing data using Hadoop ecosystem applications
mahout-client	0.12.2	Library for machine learning.
mysql-server	5.5.52	MySQL database server.
oozie-client	4.2.0	Oozie command-line client.
oozie-server	4.2.0	Service for accepting Oozie workflow requests.
phoenix-library	4.7.0-HBase-1.2	The phoenix libraries for server and client
phoenix-query-server	4.7.0-HBase-1.2	A light weight server providing JDBC access as well as Protocol Buffers and JSON format access to the Avatica API
presto-coordinator	0.157.1	Service for accepting queries and managing query execution among presto-workers.
presto-worker	0.157.1	Service for executing pieces of a query.
pig-client	0.16.0-amzn-0	Pig command-line client.
spark-client	2.0.2	Spark command-line clients.
spark-history-server	2.0.2	Web UI for viewing logged events for the lifetime of a completed Spark application.
spark-on-yarn	2.0.2	In-memory execution engine for YARN.
spark-yarn-slave	2.0.2	Apache Spark libraries needed by YARN slaves.
sqoop-client	1.4.6	Apache Sqoop command-line client.
tez-on-yarn	0.8.4	The tez YARN application and libraries.
webserver	2.4.23	Apache HTTP server.

Component	Version	Description
zeppelin-server	0.6.2	Web-based notebook that enables interactive data analytics.
zookeeper-server	3.4.9	Centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
zookeeper-client	3.4.9	ZooKeeper command line client.

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 Command Line Interface 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 previous releases.

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.

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.

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.

Previous Releases

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.

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 Amazon S3 Storage Mode for HBase (p. 68).

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.

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

For more information about other Amazon EMR fixes and features, see Previous Releases (p. 8).

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 Apache Hive (p. 87) and Apache Pig (p. 109).
- 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.

Application name changes

Amazon EMR 4.7.2	Amazon EMR 5.0.0
Oozie-Sandbox	Oozie
Presto-Sandbox	Presto
Sqoop-Sandbox	Sqoop
Zeppelin-Sandbox	Zeppelin
ZooKeeper-Sandbox	ZooKeeper

- 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 Apache Zeppelin (p. 144).
- Added support for security configurations, which allow you to create and apply encryption options more easily. For more information, see Data Encryption (p. 146).

Release 4.7.2

The following release notes include information for Amazon EMR 4.7.2. For more information about other Amazon EMR fixes and features, see Previous Releases (p. 8).

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 Creating an AWSCredentialsProvider for EMRFS (p. 189).
- 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

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.

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/redshift/jdbc`. `/usr/share/aws/redshift/jdbc/RedshiftJDBC41.jar` is the JDBC 4.1-compatible Amazon Redshift driver and `/usr/share/aws/redshift/jdbc/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 (p. 23).

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

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 (p. 23).

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

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

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 (p. 17).

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 nameserver 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

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

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 you install by supplying a configuration object when specifying applications you want installed at cluster creation time. Configuration objects consist of a classification, properties, and optional nested configurations. A classification refers to an application-specific configuration file. Properties are the settings you want to change in that file. You typically supply configurations in a list, allowing you to edit multiple configuration files in one JSON list.

Example JSON for a list of configurations is provided below:

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "hadoop.security.groups.cache.secs": "250"
    }
  },
  {
    "Classification": "mapred-site",
    "Properties": {
      "mapred.tasktracker.map.tasks.maximum": "2",
      "mapreduce.map.sort.spill.percent": "90",
      "mapreduce.tasktracker.reduce.tasks.maximum": "5"
    }
  }
]
```

The classification usually specifies the file name that you want modified. 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:

```
[
  {
```

```
    "Classification": "hadoop-env",
    "Properties": {

    },
    "Configurations": [
      {
        "Classification": "export",
        "Properties": {
          "HADOOP_DATANODE_HEAPSIZE": "2048",
          "HADOOP_NAMENODE_OPTS": "-XX:GCTimeRatio=19"
        },
        "Configurations": [

        ]
      }
    ]
  }
]
```

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

```
[
  {
    "Classification": "yarn-env",
    "Properties": {

    },
    "Configurations": [
      {
        "Classification": "export",
        "Properties": {
          "YARN_RESOURCEMANAGER_OPTS": "-Xdebug -
Xrunjdwp:transport=dt_socket"
        },
        "Configurations": [

        ]
      }
    ]
  }
]
```

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

Application	Release label classification	Valid properties	When to use
Spark	`spark`	`maximizeResourceAllocation`	Configure executors to utilize the maximum resources of each node.

The following are all configuration classifications for this release:

Configuration Classifications

Classifications	Description
capacity-scheduler	Change values in Hadoop's capacity-scheduler.xml file.
core-site	Change values in Hadoop's core-site.xml file.
emrfs-site	Change EMRFS settings.
flink-conf	Change flink-conf.yaml settings.
flink-log4j	Change Flink log4j.properties settings.
flink-log4j-yarn-session	Change Flink log4j-yarn-session.properties settings.
flink-log4j-cli	Change Flink log4j-cli.properties settings.
hadoop-env	Change values in the Hadoop environment for all Hadoop components.
hadoop-log4j	Change values in Hadoop's log4j.properties file.
hadoop-ssl-server	Change hadoop ssl server configuration
hadoop-ssl-client	Change hadoop ssl client configuration
hbase	Amazon EMR-curated settings for Apache HBase.
hbase-env	Change values in HBase's environment.
hbase-log4j	Change values in HBase's hbase-log4j.properties file.
hbase-metrics	Change values in HBase's hadoop-metrics2-hbaase.properties file.
hbase-policy	Change values in HBase's hbase-policy.xml file.
hbase-site	Change values in HBase's hbase-site.xml file.
hdfs-encryption-zones	Configure HDFS encryption zones.
hdfs-site	Change values in HDFS's hdfs-site.xml.
hcatalog-env	Change values in HCatalog's environment.
hcatalog-server-jndi	Change values in HCatalog's jndi.properties.
hcatalog-server-proto-hive-site	Change values in HCatalog's proto-hive-site.xml.
hcatalog-webhcat-env	Change values in HCatalog WebHCat's environment.
hcatalog-webhcat-log4j2	Change values in HCatalog WebHCat's log4j2.properties.
hcatalog-webhcat-site	Change values in HCatalog WebHCat's webhcat-site.xml file.

Classifications	Description
hive-beeline-log4j2	Change values in Hive's beeline-log4j2.properties file.
hive-env	Change values in the Hive environment.
hive-exec-log4j2	Change values in Hive's hive-exec-log4j2.properties file.
hive-llap-daemon-log4j2	Change values in Hive's llap-daemon-log4j2.properties file.
hive-log4j2	Change values in Hive's hive-log4j2.properties file.
hive-site	Change values in Hive's hive-site.xml file
hiveserver2-site	Change values in Hive Server2's hiveserver2-site.xml file
hue-ini	Change values in Hue's ini file
httpfs-env	Change values in the HTTPFS environment.
httpfs-site	Change values in Hadoop's httpfs-site.xml file.
hadoop-kms-acls	Change values in Hadoop's kms-acls.xml file.
hadoop-kms-env	Change values in the Hadoop KMS environment.
hadoop-kms-log4j	Change values in Hadoop's kms-log4j.properties file.
hadoop-kms-site	Change values in Hadoop's kms-site.xml file.
mapred-env	Change values in the MapReduce application's environment.
mapred-site	Change values in the MapReduce application's mapred-site.xml file.
oozie-env	Change values in Oozie's environment.
oozie-log4j	Change values in Oozie's oozie-log4j.properties file.
oozie-site	Change values in Oozie's oozie-site.xml file.
phoenix-hbase-metrics	Change values in Phoenix's hadoop-metrics2-hbase.properties file.
phoenix-hbase-site	Change values in Phoenix's hbase-site.xml file.
phoenix-log4j	Change values in Phoenix's log4j.properties file.
phoenix-metrics	Change values in Phoenix's hadoop-metrics2-phoenix.properties file.
pig-properties	Change values in Pig's pig.properties file.
pig-log4j	Change values in Pig's log4j.properties file.

Classifications	Description
presto-log	Change values in Presto's log.properties file.
presto-config	Change values in Presto's config.properties file.
presto-connector-blackhole	Change values in Presto's blackhole.properties file.
presto-connector-cassandra	Change values in Presto's cassandra.properties file.
presto-connector-hive	Change values in Presto's hive.properties file.
presto-connector-jmx	Change values in Presto's jmx.properties file.
presto-connector-kafka	Change values in Presto's kafka.properties file.
presto-connector-localfile	Change values in Presto's localfile.properties file.
presto-connector-mongodb	Change values in Presto's mongodb.properties file.
presto-connector-mysql	Change values in Presto's mysql.properties file.
presto-connector-postgresql	Change values in Presto's postgresql.properties file.
presto-connector-raptor	Change values in Presto's raptor.properties file.
presto-connector-redis	Change values in Presto's redis.properties file.
presto-connector-tpch	Change values in Presto's tpch.properties file.
spark	Amazon EMR-curated settings for Apache Spark.
spark-defaults	Change values in Spark's spark-defaults.conf file.
spark-env	Change values in the Spark environment.
spark-hive-site	Change values in Spark's hive-site.xml file
spark-log4j	Change values in Spark's log4j.properties file.
spark-metrics	Change values in Spark's metrics.properties file.
sqoop-env	Change values in Sqoop's environment.
sqoop-oraoop-site	Change values in Sqoop OraOop's oraoop-site.xml file.
sqoop-site	Change values in Sqoop's sqoop-site.xml file.
tez-site	Change values in Tez's tez-site.xml file.
yarn-env	Change values in the YARN environment.
yarn-site	Change values in YARN's yarn-site.xml file.
zeppelin-env	Change values in the Zeppelin environment.
zookeeper-config	Change values in ZooKeeper's zoo.cfg file.

Classifications	Description
zookeeper-log4j	Change values in ZooKeeper's log4j.properties file.

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 a 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:

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge
 --instance-count 2 --applications Name=Hive --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

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

```
aws emr create-cluster --release-label emr-5.2.1 --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-5.2.1")
 .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 Java 8

You set `JAVA_HOME` for an application by supplying the setting to its environment classification, *`application`*`-env`. For Hadoop and Hive, this would look like:

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

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:

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

Setting	Hostname/Port
fs.default.name	default (hdfs://*emrDeterminedIP*:8020)
dfs.datanode.address	default (0.0.0.0:50010)
dfs.datanode.http.address	default (0.0.0.0:50075)
dfs.datanode.https.address	default (0.0.0.0:50475)
dfs.datanode.ipc.address	default (0.0.0.0:50020)
dfs.http.address	default (0.0.0.0:50070)
dfs.https.address	default (0.0.0.0:50470)
dfs.secondary.http.address	default (0.0.0.0:50090)
yarn.nodemanager.address	default (${yarn.nodemanager.hostname}:0)
yarn.nodemanager.localizer.address	default (${yarn.nodemanager.hostname}:8040)
yarn.nodemanager.webapp.address	default (${yarn.nodemanager.hostname}:8042)
yarn.resourcemanager.address	default (${yarn.resourcemanager.hostname}:8032)
yarn.resourcemanager.admin.address	default (${yarn.resourcemanager.hostname}:8033)
yarn.resourcemanager.resource-tracker.address	default (${yarn.resourcemanager.hostname}:8031)
yarn.resourcemanager.scheduler.address	default (${yarn.resourcemanager.hostname}:8030)
yarn.resourcemanager.webapp.address	default (${yarn.resourcemanager.hostname}:8088)
yarn.web-proxy.address	default (no-value)
yarn.resourcemanager.hostname	*emrDeterminedIP*

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.

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-openjdk/bin/java -Xmx256m -Dhive.log.dir=/var/log/
hive -Dhive.log.file=hive-metastore.log -Dhive.log.threshold=INFO -
Dhadoop.log.dir=/usr/lib/hadoop
hive      6557  0.2  0.6 849508 202396 ?       Sl   16:32   0:09 /usr/
lib/jvm/java-openjdk/bin/java -Xmx256m -Dhive.log.dir=/var/log/hive -
Dhive.log.file=hive-server2.log -Dhive.log.threshold=INFO -Dhadoop.log.dir=/
usr/lib/hadoop/l
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 -
Dhadoo
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-y
mapred    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

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Hadoop 2.7.3	emr-5.2.1	emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-mapred, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager

Topics

Create or Run a Hadoop Application

Topics

Build Binaries Using Amazon EMR

You can use Amazon EMR (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

1	Connect to the master node of your cluster.
2	Copy source files to the master node.
3	Build binaries with any necessary optimizations.
4	Copy binaries from the master node to Amazon S3.

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:

```
mkdir SourceFiles
```

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

```
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:

  ```
  master$ cat /etc/issue
  ```

  The output looks similar to the following.

  ```
  Debian GNU/Linux 5.0
  ```

  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 msr 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 cx16 xtpr 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 (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 .

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.

```
aws emr create-cluster --name "Test cluster" –-release-label emr-5.2.1
 --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 (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 use to Hadoop's Streaming utility.

Hadoop Process

1	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.

Parameter	Description	Required
-input	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.	Yes
-output	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 *`input`*.	Yes
-mapper	Name of the mapper executable. Type: String Default: None	Yes
-reducer	Name of the reducer executable. Type: String Default: None	Yes
-cacheFile	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]	No
-cacheArchive	JAR file to extract into the working directory Type: String Default: None Constraints: [URI]#[symlink directory name to create in working directory	No

Parameter	Description	Required
-combiner	Combines results Type: String Default: None Constraints: Java class name	No

The following code sample is a mapper executable written in Python. This script is part of the WordCount sample application.

```
#!/usr/bin/python
import sys

def main(argv):
  line = sys.stdin.readline()
  try:
    while line:
      line = line.rstrip()
      words = line.split()
      for word in words:
        print "LongValueSum:" + word + "\t" + "1"
      line = sys.stdin.readline()
  except "end of file":
    return None
if __name__ == "__main__":
  main(sys.argv)
```

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**.
   - For **Name**, accept the default name (Streaming program) or type a new name.

- For **Mapper**, type or browse to the location of your mapper class in Hadoop, or an S3 bucket where the mapper executable, such as a Python program, resides. The path value must be in the form *BucketName*/*path*/*MapperExecutable*.
- For **Reducer**, type or browse to the location of your reducer class in Hadoop, or an S3 bucket where the reducer executable, such as a Python program, resides. The path value must be in the form *BucketName*/*path*/*MapperExecutable*. Amazon EMR supports the special *aggregate* keyword. For more information, go to the Aggregate library supplied by Hadoop.
- For **Input S3 location**, type or browse to the location of your input data.
- 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.

AWS CLI

These examples demonstrate how to use the AWS CLI to create a cluster and submit a Streaming step.

To create a cluster and submit a Streaming step using the AWS CLI

- To create a cluster and submit a Streaming step using the AWS CLI, type the following command and replace *myKey* with the name of your EC2 key pair.

```
aws emr create-cluster --name "Test cluster" --release-label emr-5.2.1 --
applications Name=Hue Name=Hive Name=Pig --use-default-roles \
--ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-
count 3 \
--steps Type=STREAMING,Name="Streaming
 Program",ActionOnFailure=CONTINUE,Args=[--files,pathtoscripts,-
mapper,mapperscript,-reducer,reducerscript,aggregate,-
input,pathtoinputdata,-output,pathtooutputbucket]
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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.

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. 27).

For more information about building a Hadoop MapReduce application, go to http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

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.

```
aws emr create-cluster --name "Test cluster" --release-label emr-5.2.1 \
--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","pathtooutputb
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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. 33).
- Launch the cluster with a modified `mapreduce.application.classpath` setting in `mapred-site.xml` using the `mapred-site` configuration classificiation. 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","pathtooutputbuc
 \
--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

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

m1.medium

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	384
YARN_PROXYSERVER_HEAPSIZE	192
YARN_NODEMANAGER_HEAPSIZE	256
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	256
HADOOP_NAMENODE_HEAPSIZE	384
HADOOP_DATANODE_HEAPSIZE	192

m1.large

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	768
YARN_PROXYSERVER_HEAPSIZE	384
YARN_NODEMANAGER_HEAPSIZE	512
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	512
HADOOP_NAMENODE_HEAPSIZE	768
HADOOP_DATANODE_HEAPSIZE	384

m1.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	1024
YARN_PROXYSERVER_HEAPSIZE	512
YARN_NODEMANAGER_HEAPSIZE	768
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1024
HADOOP_NAMENODE_HEAPSIZE	2304
HADOOP_DATANODE_HEAPSIZE	384

m2.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	1536

Parameter	Value
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1024
HADOOP_NAMENODE_HEAPSIZE	3072
HADOOP_DATANODE_HEAPSIZE	384

m2.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	1536
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	6144
HADOOP_DATANODE_HEAPSIZE	384

m2.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

m3.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2396
YARN_PROXYSERVER_HEAPSIZE	2396
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2396
HADOOP_NAMENODE_HEAPSIZE	1740
HADOOP_DATANODE_HEAPSIZE	757

m3.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2703
YARN_PROXYSERVER_HEAPSIZE	2703
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2703
HADOOP_NAMENODE_HEAPSIZE	3276
HADOOP_DATANODE_HEAPSIZE	1064

m4.large

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2252
YARN_PROXYSERVER_HEAPSIZE	2252
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2252
HADOOP_NAMENODE_HEAPSIZE	1024
HADOOP_DATANODE_HEAPSIZE	614

m4.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2416
YARN_PROXYSERVER_HEAPSIZE	2416
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2416
HADOOP_NAMENODE_HEAPSIZE	2048
HADOOP_DATANODE_HEAPSIZE	778

m4.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2744
YARN_PROXYSERVER_HEAPSIZE	2744
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2744

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	3481
HADOOP_DATANODE_HEAPSIZE	1105

m4.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3399
YARN_PROXYSERVER_HEAPSIZE	3399
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3399
HADOOP_NAMENODE_HEAPSIZE	6758
HADOOP_DATANODE_HEAPSIZE	1761

m4.10xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	5365
YARN_PROXYSERVER_HEAPSIZE	5365
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	5365
HADOOP_NAMENODE_HEAPSIZE	16588
HADOOP_DATANODE_HEAPSIZE	3727

m4.16xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	7331
YARN_PROXYSERVER_HEAPSIZE	7331
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	7331
HADOOP_NAMENODE_HEAPSIZE	26419
HADOOP_DATANODE_HEAPSIZE	4096

c1.medium

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	192
YARN_PROXYSERVER_HEAPSIZE	96
YARN_NODEMANAGER_HEAPSIZE	128
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	128
HADOOP_NAMENODE_HEAPSIZE	192
HADOOP_DATANODE_HEAPSIZE	96

c1.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	768
YARN_PROXYSERVER_HEAPSIZE	384
YARN_NODEMANAGER_HEAPSIZE	512
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	512
HADOOP_NAMENODE_HEAPSIZE	768
HADOOP_DATANODE_HEAPSIZE	384

c3.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2124
YARN_PROXYSERVER_HEAPSIZE	2124
YARN_NODEMANAGER_HEAPSIZE	2124
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2124
HADOOP_NAMENODE_HEAPSIZE	972
HADOOP_DATANODE_HEAPSIZE	588

c3.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2396
YARN_PROXYSERVER_HEAPSIZE	2396
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2396

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	1740
HADOOP_DATANODE_HEAPSIZE	757

c3.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2703
YARN_PROXYSERVER_HEAPSIZE	2703
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2703
HADOOP_NAMENODE_HEAPSIZE	3276
HADOOP_DATANODE_HEAPSIZE	1064

c3.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3317
YARN_PROXYSERVER_HEAPSIZE	3317
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3317
HADOOP_NAMENODE_HEAPSIZE	6348
HADOOP_DATANODE_HEAPSIZE	1679

c4.large

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	1152
YARN_PROXYSERVER_HEAPSIZE	1152
YARN_NODEMANAGER_HEAPSIZE	1152
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1152
HADOOP_NAMENODE_HEAPSIZE	576
HADOOP_DATANODE_HEAPSIZE	384

c4.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2124
YARN_PROXYSERVER_HEAPSIZE	2124
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2124
HADOOP_NAMENODE_HEAPSIZE	972
HADOOP_DATANODE_HEAPSIZE	588

c4.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2396
YARN_PROXYSERVER_HEAPSIZE	2396
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2396
HADOOP_NAMENODE_HEAPSIZE	1740
HADOOP_DATANODE_HEAPSIZE	757

c4.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2703
YARN_PROXYSERVER_HEAPSIZE	2703
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2703
HADOOP_NAMENODE_HEAPSIZE	3276
HADOOP_DATANODE_HEAPSIZE	1064

c4.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3317
YARN_PROXYSERVER_HEAPSIZE	3317
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3317

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	6348
HADOOP_DATANODE_HEAPSIZE	1679

cc2.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

cg1.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	3840
HADOOP_DATANODE_HEAPSIZE	384

cr1.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

d2.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

d2.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

d2.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

d2.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	7089
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	7086

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

hi1.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3328
YARN_PROXYSERVER_HEAPSIZE	3328
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3328
HADOOP_NAMENODE_HEAPSIZE	6400
HADOOP_DATANODE_HEAPSIZE	1689

hs1.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

i2.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

i2.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

i2.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

i2.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

g2.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	1536
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	1024

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	2304
HADOOP_DATANODE_HEAPSIZE	384

r3.xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

r3.2xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

r3.4xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

r3.8xlarge

Parameter	Value
YARN_RESOURCEMANAGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

HDFS Configuration

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

Parameter	Definition	Default value
dfs.block.size	The size of HDFS blocks. When operating on data stored in HDFS, the split size is generally the size of an HDFS block. Larger numbers provide less task granularity, but also put less strain on the cluster `NameNode`.	134217728 (128 MB)
dfs.replication	The number of copies of each block to store for durability. For small clusters, set this to 2 because the cluster is small and 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.	1 for clusters < four nodes 2 for clusters < ten nodes 3 for all other clusters

Task Configuration

Topics

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.

The defaults for these settings per instance type are shown in the following tables. The settings change when HBase is installed and those are also provided along with the initial defaults.

> **Note**
> HBase is only supported on Amazon EMR releases 4.6.0 or later.

m1.medium

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx512m
mapreduce.reduce.java.opts	-Xmx768m
mapreduce.map.memory.mb	768
mapreduce.reduce.memory.mb	1024
yarn.app.mapreduce.am.resource.mb	1024
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	2048
yarn.nodemanager.resource.memory-mb	2048

m1.large

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx512m	-Xmx512m
mapreduce.reduce.java.opts	-Xmx1024m	-Xmx1024m
mapreduce.map.memory.mb	768	768
mapreduce.reduce.memory.mb	1536	1536
yarn.app.mapreduce.am.resource.mb	1536	1536
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	5120	2560
yarn.nodemanager.resource.memory-mb	5120	2560

m1.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx512m	-Xmx512m
mapreduce.reduce.java.opts	-Xmx1536m	-Xmx1536m
mapreduce.map.memory.mb	768	768
mapreduce.reduce.memory.mb	2048	2048
yarn.app.mapreduce.am.resource.mb	2048	2048
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	12288	6144
yarn.nodemanager.resource.memory-mb	12288	6144

m2.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx864m	-Xmx864m
mapreduce.reduce.java.opts	-Xmx1536m	-Xmx1536m
mapreduce.map.memory.mb	1024	1024
mapreduce.reduce.memory.mb	2048	2048
yarn.app.mapreduce.am.resource.mb	2048	2048
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	14336	7168
yarn.nodemanager.resource.memory-mb	14336	7168

m2.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1280m	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	2560	2560
yarn.app.mapreduce.am.resource.mb	2560	2560
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	30720	15360
yarn.nodemanager.resource.memory-mb	30720	15360

m2.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1280m	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	2560	2560
yarn.app.mapreduce.am.resource.mb	2560	2560
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	61440	30720
yarn.nodemanager.resource.memory-mb	61440	30720

m3.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	11520	5760
yarn.nodemanager.resource.memory-mb	11520	5760

m3.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23040	11520
yarn.nodemanager.resource.memory-mb	23040	11520

m4.large

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2458m	-Xmx2458m
mapreduce.reduce.java.opts	-Xmx4916m	-Xmx4916m
mapreduce.map.memory.mb	3072	3072
mapreduce.reduce.memory.mb	6144	6144
yarn.app.mapreduce.am.resource.mb	6144	6144
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	6144	3072
yarn.nodemanager.resource.memory-mb	6144	3072

m4.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1229m	-Xmx1229m
mapreduce.reduce.java.opts	-Xmx2548m	-Xmx2458m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	3072	3072
yarn.app.mapreduce.am.resource.mb	3072	3072
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	12288	6144
yarn.nodemanager.resource.memory-mb	12288	6144

m4.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1229m	-Xmx1229m
mapreduce.reduce.java.opts	-Xmx2458m	-Xmx2458m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	3072	3072
yarn.app.mapreduce.am.resource.mb	3072	3072
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	24576	12288
yarn.nodemanager.resource.memory-mb	24576	12288

m4.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1434m	-Xmx1434m
mapreduce.reduce.java.opts	-Xmx2868m	-Xmx2868m
mapreduce.map.memory.mb	1792	1792
mapreduce.reduce.memory.mb	3584	3584
yarn.app.mapreduce.am.resource.mb	3584	3584
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	57344	28672
yarn.nodemanager.resource.memory-mb	57344	28672

m4.10xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1557m	-Xmx1557m
mapreduce.reduce.java.opts	-Xmx3114m	-Xmx3114m
mapreduce.map.memory.mb	1946	1946
mapreduce.reduce.memory.mb	3892	3892
yarn.app.mapreduce.am.resource.mb	3892	3892
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	155648	124544
yarn.nodemanager.resource.memory-mb	155648	124544

m4.16xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1587m
mapreduce.reduce.java.opts	-Xmx3114m
mapreduce.map.memory.mb	1984
mapreduce.reduce.memory.mb	3968
yarn.app.mapreduce.am.resource.mb	3968
yarn.scheduler.minimum-allocation-mb	32
yarn.scheduler.maximum-allocation-mb	253952
yarn.nodemanager.resource.memory-mb	253952

c1.medium

Configuration Option	Default Value
io.sort.mb	100
mapreduce.map.java.opts	-Xmx288m
mapreduce.reduce.java.opts	-Xmx288m
mapreduce.map.memory.mb	512
mapreduce.reduce.memory.mb	512
yarn.app.mapreduce.am.resource.mb	
yarn.scheduler.minimum-allocation-mb	32
yarn.scheduler.maximum-allocation-mb	512
yarn.nodemanager.resource.memory-mb	1024

c1.xlarge

Configuration Option	Default Value	With HBase Installed
io.sort.mb	150	150
mapreduce.map.java.opts	-Xmx864m	-Xmx864m
mapreduce.reduce.java.opts	-Xmx1536m	-Xmx1536m
mapreduce.map.memory.mb	1024	1024
mapreduce.reduce.memory.mb	2048	2048
yarn.app.mapreduce.am.resource.mb	2048	2048
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	5120	2560
yarn.nodemanager.resource.memory-mb	5120	2560

c3.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1126m	-Xmx1126m
mapreduce.reduce.java.opts	-Xmx2252m	-Xmx2252m
mapreduce.map.memory.mb	1408	1408
mapreduce.reduce.memory.mb	2816	2816
yarn.app.mapreduce.am.resource.mb	2816	2816
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	5632	2816
yarn.nodemanager.resource.memory-mb	5632	2816

c3.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	11520	5760
yarn.nodemanager.resource.memory-mb	11520	5760

c3.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23040	11520
yarn.nodemanager.resource.memory-mb	23040	11520

c3.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1331m	-Xmx1331m
mapreduce.reduce.java.opts	-Xmx2662m	-Xmx2662m
mapreduce.map.memory.mb	1664	1664
mapreduce.reduce.memory.mb	3328	3328
yarn.app.mapreduce.am.resource.mb	3328	3328
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	53248	26624
yarn.nodemanager.resource.memory-mb	53248	26624

c4.large

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx717m	-Xmx717m
mapreduce.reduce.java.opts	-Xmx1434m	-Xmx1434m
mapreduce.map.memory.mb	896	896
mapreduce.reduce.memory.mb	1792	1792
yarn.app.mapreduce.am.resource.mb	1792	1792
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	1792	896
yarn.nodemanager.resource.memory-mb	1792	896

c4.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1126m	-Xmx1126m
mapreduce.reduce.java.opts	-Xmx2252m	-Xmx2252m
mapreduce.map.memory.mb	1408	1408
mapreduce.reduce.memory.mb	2816	2816
yarn.app.mapreduce.am.resource.mb	2816	2816
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	5632	2816
yarn.nodemanager.resource.memory-mb	5632	2816

c4.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	11520	5760
yarn.nodemanager.resource.memory-mb	11520	5760

c4.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1152m	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1440	1440
mapreduce.reduce.memory.mb	2880	2880
yarn.app.mapreduce.am.resource.mb	2880	2880
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23040	11520
yarn.nodemanager.resource.memory-mb	23040	11520

c4.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1183m	-Xmx1183m
mapreduce.reduce.java.opts	-Xmx2366m	-Xmx2366m
mapreduce.map.memory.mb	1479	1479
mapreduce.reduce.memory.mb	2958	2958
yarn.app.mapreduce.am.resource.mb	2958	2958
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	53248	26624
yarn.nodemanager.resource.memory-mb	53248	26624

cg1.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1280m	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	2560	2560
yarn.app.mapreduce.am.resource.mb	2560	2560
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	20480	10240
yarn.nodemanager.resource.memory-mb	20480	10240

cc2.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1280m	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	2560	2560
yarn.app.mapreduce.am.resource.mb	2560	2560
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	56320	28160
yarn.nodemanager.resource.memory-mb	56320	28160

cr1.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx6042m	-Xmx6042m
mapreduce.reduce.java.opts	-Xmx12084m	-Xmx12084m
mapreduce.map.memory.mb	7552	7552
mapreduce.reduce.memory.mb	15104	15104
yarn.app.mapreduce.am.resource.mb	15104	15104
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	241664	211456
yarn.nodemanager.resource.memory-mb	241664	211456

d2.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2342m	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m	-Xmx4684m
mapreduce.map.memory.mb	2928	2928
mapreduce.reduce.memory.mb	5856	5856
yarn.app.mapreduce.am.resource.mb	5856	5856
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23424	11712
yarn.nodemanager.resource.memory-mb	23424	11712

d2.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2714m	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m	-Xmx5428m
mapreduce.map.memory.mb	3392	3392
mapreduce.reduce.memory.mb	6784	6784
yarn.app.mapreduce.am.resource.mb	6784	6784
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	54272	27136
yarn.nodemanager.resource.memory-mb	54272	27136

d2.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2918m	-Xmx2918m
mapreduce.reduce.java.opts	-Xmx5836m	-Xmx5836m
mapreduce.map.memory.mb	3648	3648
mapreduce.reduce.memory.mb	7296	7296
yarn.app.mapreduce.am.resource.mb	7296	7296
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	116736	87552
yarn.nodemanager.resource.memory-mb	116736	87552

d2.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2417m	-Xmx2417m
mapreduce.reduce.java.opts	-Xmx4384m	-Xmx4834m
mapreduce.map.memory.mb	3021	3021
mapreduce.reduce.memory.mb	6042	6042
yarn.app.mapreduce.am.resource.mb	6042	6042
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	241664	211470
yarn.nodemanager.resource.memory-mb	241664	211470

g2.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx512m	-Xmx512m
mapreduce.reduce.java.opts	-Xmx1536m	-Xmx1536m
mapreduce.map.memory.mb	768	768
mapreduce.reduce.memory.mb	2048	2048
yarn.app.mapreduce.am.resource.mb	2048	2048
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	12288	6144
yarn.nodemanager.resource.memory-mb	12288	6144

hi1.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2688m	-Xmx2688m
mapreduce.reduce.java.opts	-Xmx5376m	-Xmx5376m
mapreduce.map.memory.mb	3360	3360
mapreduce.reduce.memory.mb	6720	6720
yarn.app.mapreduce.am.resource.mb	6720	6720
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	53760	26880
yarn.nodemanager.resource.memory-mb	53760	26880

hs1.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx1280m	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m	-Xmx2304m
mapreduce.map.memory.mb	1536	1536
mapreduce.reduce.memory.mb	2560	2560
yarn.app.mapreduce.am.resource.mb	2560	2560
yarn.scheduler.minimum-allocation-mb	256	32
yarn.scheduler.maximum-allocation-mb	8192	28160
yarn.nodemanager.resource.memory-mb	56320	28160

i2.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2342m	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m	-Xmx4684m
mapreduce.map.memory.mb	2928	2928
mapreduce.reduce.memory.mb	5856	5856
yarn.app.mapreduce.am.resource.mb	5856	5856
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23424	11712
yarn.nodemanager.resource.memory-mb	23424	11712

i2.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2714m	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m	-Xmx5428m
mapreduce.map.memory.mb	3392	3392
mapreduce.reduce.memory.mb	6784	6784
yarn.app.mapreduce.am.resource.mb	6784	6784
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	54272	27136
yarn.nodemanager.resource.memory-mb	54272	27136

i2.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2918m	-Xmx2918m
mapreduce.reduce.java.opts	-Xmx5836m	-Xmx5836m
mapreduce.map.memory.mb	3648	3648
mapreduce.reduce.memory.mb	7296	7296
yarn.app.mapreduce.am.resource.mb	7296	7296
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	116736	87552
yarn.nodemanager.resource.memory-mb	116736	87552

i2.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx3021m	-Xmx3021m
mapreduce.reduce.java.opts	-Xmx6042m	-Xmx6042m
mapreduce.map.memory.mb	3776	3776
mapreduce.reduce.memory.mb	7552	7552
yarn.app.mapreduce.am.resource.mb	7552	7552
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	241664	211456
yarn.nodemanager.resource.memory-mb	241664	211456

r3.xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2342m	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m	-Xmx4684m
mapreduce.map.memory.mb	2982	2982
mapreduce.reduce.memory.mb	5856	5856
yarn.app.mapreduce.am.resource.mb	5856	5856
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	23424	11712
yarn.nodemanager.resource.memory-mb	23424	11712

r3.2xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2714m	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m	-Xmx5428m
mapreduce.map.memory.mb	3392	3392
mapreduce.reduce.memory.mb	6784	6784
yarn.app.mapreduce.am.resource.mb	6784	6784
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	54272	27136
yarn.nodemanager.resource.memory-mb	54272	27136

r3.4xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx2918m	-Xmx2918m
mapreduce.reduce.java.opts	-Xmx5836m	-Xmx5836m
mapreduce.map.memory.mb	3648	3648
mapreduce.reduce.memory.mb	7296	7296
yarn.app.mapreduce.am.resource.mb	7296	7296
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	116736	87552
yarn.nodemanager.resource.memory-mb	116736	87552

r3.8xlarge

Configuration Option	Default Value	With HBase Installed
mapreduce.map.java.opts	-Xmx3021m	-Xmx3021m
mapreduce.reduce.java.opts	-Xmx6042m	-Xmx6042m
mapreduce.map.memory.mb	3776	3776
mapreduce.reduce.memory.mb	7552	7552
yarn.app.mapreduce.am.resource.mb	7552	7552
yarn.scheduler.minimum-allocation-mb	32	32
yarn.scheduler.maximum-allocation-mb	241664	211456
yarn.nodemanager.resource.memory-mb	241664	211456

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.

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.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Ganglia 3.7.2	emr-5.2.1	emrfs, emr-goodies, ganglia-monitor, ganglia-metadata-collector, ganglia-web, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, webserver

Topics

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.

- Type the following command to add Ganglia when you create a cluster and replace *myKey* with the name of your EC2 key pair.

```
aws emr create-cluster --name "Spark cluster with Ganglia" --release-
label emr-5.2.1 \
--applications Name=Spark Name=Ganglia --ec2-attributes KeyName=myKey --
instance-type m3.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 EMR service role and EC2 instance profile, type aws `emr create-default-roles` to create them before typing the `create-cluster` subcommand.

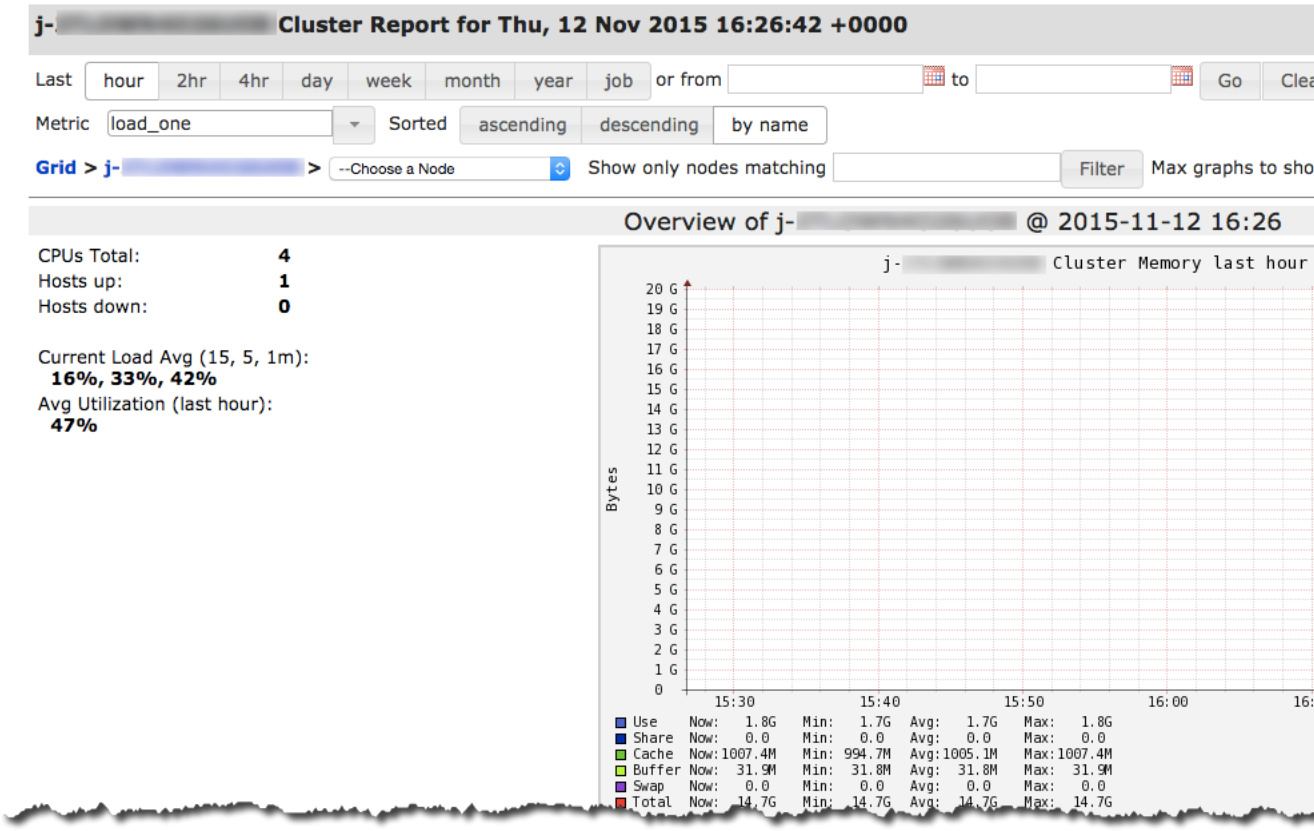
For more information about using Amazon EMR commands in the AWS CLI, see http://docs.aws.amazon.com/cli/latest/reference/emr.

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.*
- application_xxxxxxxxxx_xxxx.driver.*
- application_xxxxxxxxxx_xxxx.executor.*

You can view a complete list of these metrics by choosing the Gmetrics link, on the Host Overview page.

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. It was developed as part of Apache Software Foundation's Hadoop project and runs on top of Hadoop Distributed File System (HDFS) to provide non-relational database capabilities for the Hadoop ecosystem. HBase provides you a fault-tolerant, efficient way of storing large quantities of sparse data using column-based compression and storage. In addition, it provides fast lookup of data because large portions of data are cached in-memory. Cluster instance storage is still used. HBase is optimized for sequential write operations, and is highly efficient for batch inserts, updates, and deletes. HBase also supports cell versioning so you can look up and use several previous versions of a cell or a row.

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).

Additionally, HBase on Amazon EMR provides the ability to create snapshots of your HBase data directly to Amazon Simple Storage Service (Amazon S3). You can restore from previously created snapshots. Another option is Amazon S3 storage mode, which allows you to use Amazon S3 directly for the HBase root directory and metadata. Using Amazon S3 storage mode, you can start a new cluster, seamlessly pointing the new cluster to the root directory location in Amazon S3.

For more information, see the HBase website and documentation.

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.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
HBase 1.2.3	emr-5.2.1	emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-

Application	Amazon EMR Release Label	Components installed with this application
		library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-mapred, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hbase-hmaster, hbase-client, hbase-region-server, hbase-rest-server, hbase-thrift-server, zookeeper-client, zookeeper-server

Topics

Creating a Cluster with HBase Using the Console

The following procedure creates a cluster with HBase installed. For more information about launching 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 **Amazon Release Version** `emr-5.2.1` or later. Choose **HBase** and other applications as desired.
4. Under **HBase Storage Settings**, for **Storage mode**, select **HDFS** or **Amazon S3**. For more information, see Amazon S3 Storage Mode for HBase (p. 68).
5. Select other options as necessary and then choose **Create cluster**.

Creating a Cluster with HBase Using AWS CLI

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

```
aws emr create-cluster --name "Test cluster" --release-label emr-5.2.1 \
--applications Name=HBase --use-default-roles --ec2-attributes KeyName=myKey
 \
--instance-type m3.xlarge --instance-count 3
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

If you are using Amazon S3 storage mode for HBase, specify the `--configurations` option with a reference to a 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",}
},
{
  "Classification": "hbase",
  "Properties": {
  "hbase.emr.storageMode":"s3",
  }
}
```

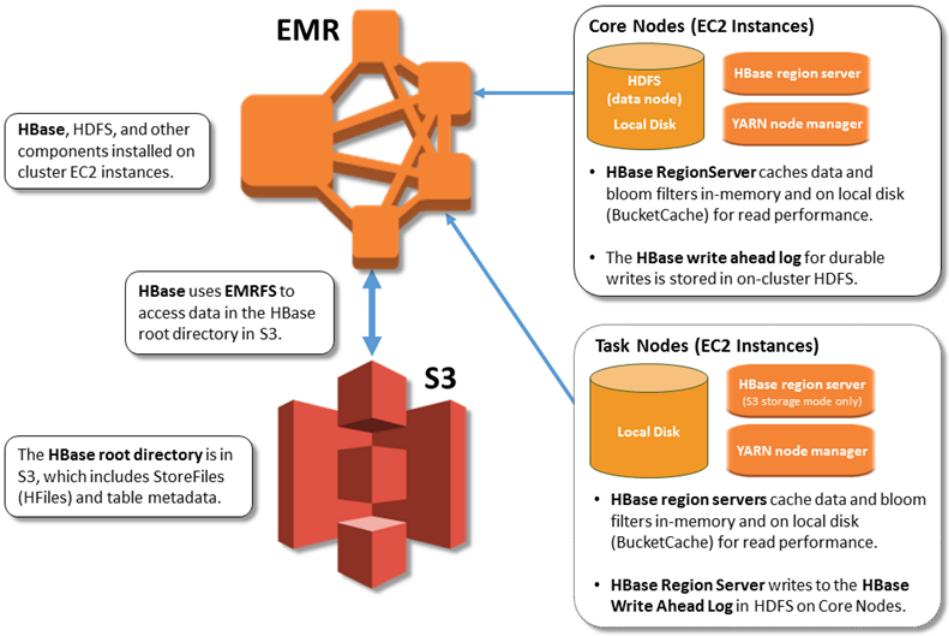
For more information about Amazon S3 storage mode for HBase, see Amazon S3 Storage Mode for HBase (p. 68). For more information about classifications, see Configuring Applications (p. 17).

Amazon S3 Storage Mode for HBase

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

- The HBase root directory is stored in Amazon S3, including store files (HFiles) 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.

The following illustration shows the HBase components relevant to Amazon S3 storage mode.



Enabling Amazon S3 Storage Mode for HBase

You can enable Amazon S3 storage mode 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 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, and 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 Using the Console (p. 67). An example configuration object is shown in the following JSON snippet.

Important
We strongly recommend that you use EMRFS consistent view when you enable Amazon S3 storage mode 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*.

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

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 store files 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 store file 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 abut 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 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 store files 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. Also, 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 store files 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 archived HFiles. 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.

HBase Properties for Amazon S3 Storage Mode Performance Tuning

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

Configuration Property	Default Value	Description
`hbase.bucketcache.size`	8,192	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

Configuration Property	Default Value	Description
		BucketCache sizes generally correspond to improved performance
`hbase.region.memstore.flush.size`	134217728	The maximum limit, in bytes, that a single memstore flushes data to Amazon S3.
`hbase.region.memstore.block.multiplier`	4	A multiplier that determines the MemStore upper limit at which updates are blocked. If the MemStore exceeds `hbase.hregion.memstore.flush.size` multiplied by this value, updates are blocked. MemStore flushes and compaction may happen to unblock updates.
`hbase.hstore.blockingStoreFiles`	10	The maximum number of store files that can exist in a store before updates are blocked.
`hbase.hregion.max.filesize`	10737418240	The maximum combined size of HFiles, in bytes, that can exist in a region before the region is split.

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. 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. 87) 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.

  Note

```
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

```
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",\
Args=[ "hbase", "snapshot", "export","-snapshot","snapshotName","-copy-
to","s3://bucketName/folder","-mappers","2"]
```

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");
```

Import snapshot from Amazon S3

Note
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","-chuser","hbase");
```

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.

```
echo 'disable tableName; \
restore_snapshot snapshotName; \
enable tableName' | hbase shell
```

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

```
[
  {
    "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")
  .withArgs("bash","-c","echo $'disable \"tableName\"; restore_snapshot
 \"snapshotName\"; enable \"snapshotName\"' | hbase shell");
```

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:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=HBase \
--instance-type m3.xlarge --instance-count 2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`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-5.2.1 --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. 47) 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

Interface	Port	Protocol
HMaster	16000	TCP
HMaster UI	16010	HTTP
RegionServer	16020	TCP
RegionServer Info	16030	HTTP
REST server	8070	HTTP
REST UI	8085	HTTP
Thrift server	9090	TCP
Thrift server UI	9095	HTTP

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": "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.

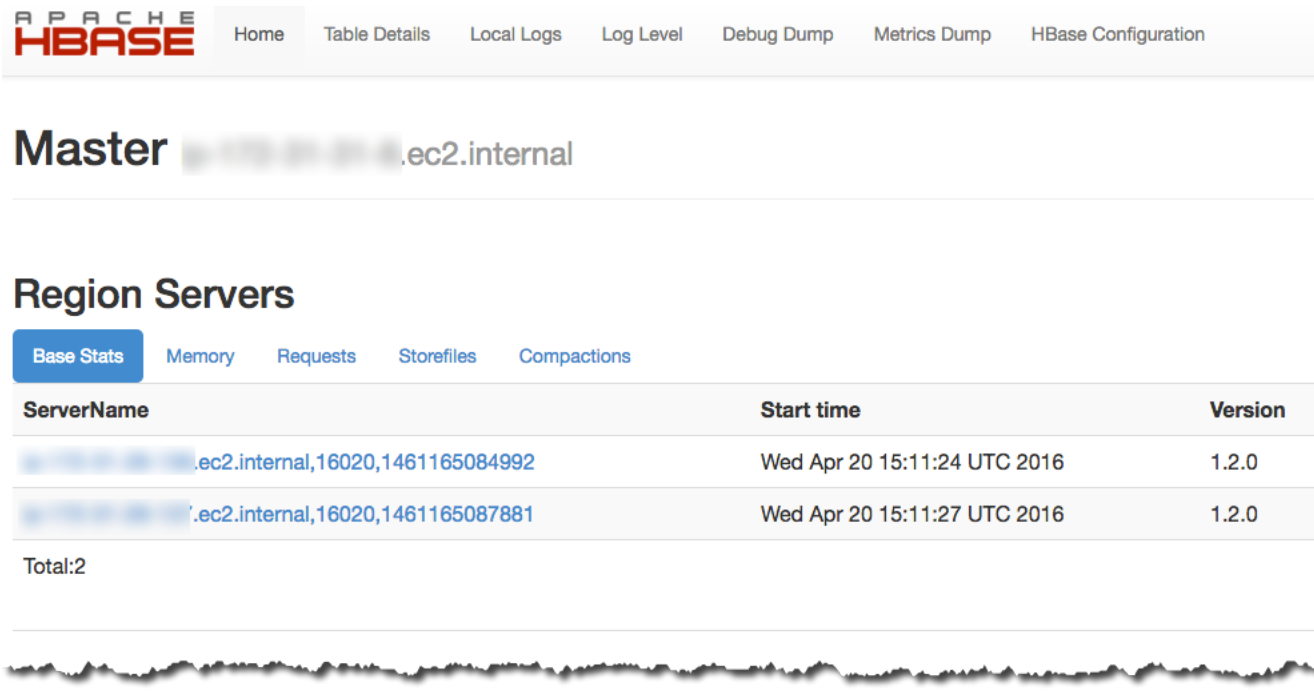
```
[
  {
    "Classification":"hbase-site",
    "Properties": {
       "hbase.hregion.memstore.flush.size": "134217728"
    }
  }
]
```

View the HBase User Interface

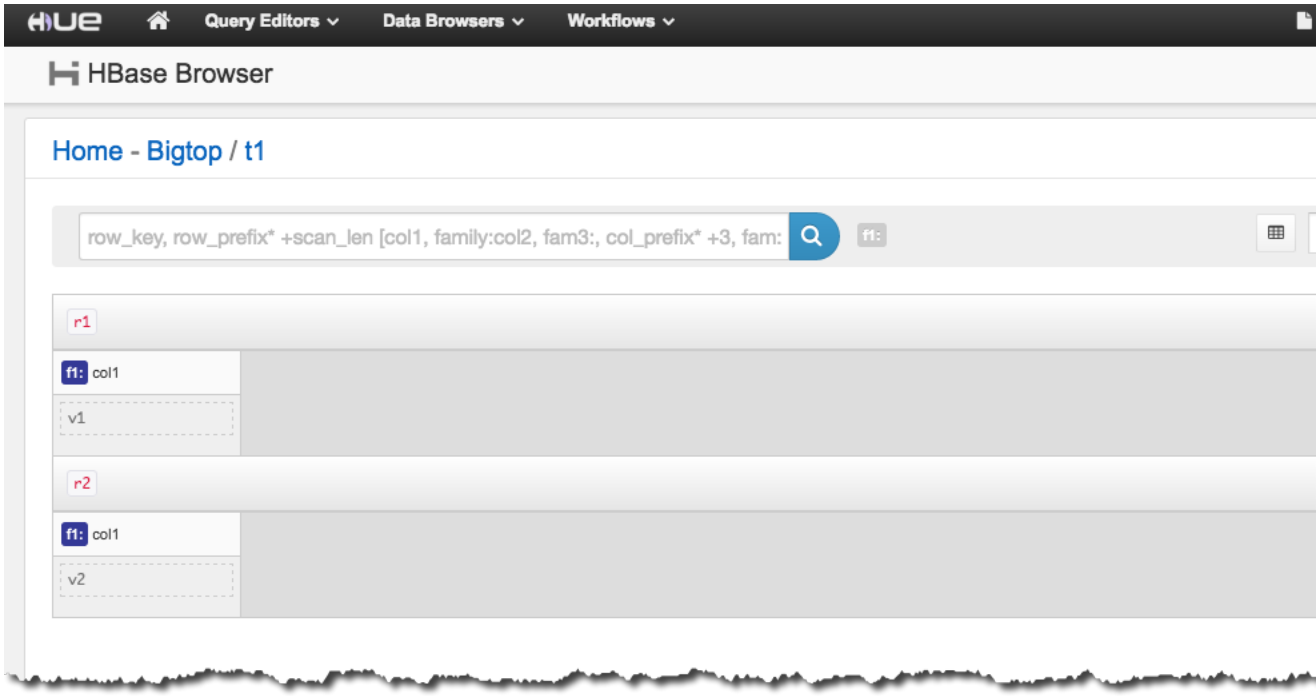
HBase provides a web-based user interface that you can use to monitor your HBase cluster. When you run HBase on Amazon EMR, the web interface runs on the master node and can be viewed using port forwarding, also known as creating an SSH tunnel.

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. 72):



For more information about Hue, see Hue (p. 95).

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. 63).

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-5.2.1 \
--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. 80).

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.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
HCatalog 2.1.0	emr-5.2.1	emrfs, emr-ddb, emr-goodies, emr-kinesis, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hcatalog-client, hcatalog-server, hcatalog-webhcat-server, hive-client, mysql-server

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 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 the **Software Configuration** field, choose **Amazon Release Version emr-4.4.0** or later.
4. In the **Select Applications** field, choose either **All Applications** or **HCatalog**.
5. Select other options as necessary and then choose **Create cluster**.

To launch a cluster with HCatalog using the AWS CLI

- Create the cluster with the following command:

```
aws emr create-cluster --name "Cluster with Hcat" --release-
label emr-5.2.1 \
--applications Name=HCatalog --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

Using HCatalog

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

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

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

```
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/';
ALTER TABLE impressions ADD PARTITION (dt='2009-04-13-08-05');
```

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;"
<snip>
(1239610346000,m9nwdo67Nx6q2kI25qt5On7peICfUM,omkxkaRpNhGPDucAiBErSh1cs0MThC,cartoonnetwork
 (compatible; MSIE 7.0; Windows NT 6.0; FunWebProducts;
 GTB6; SLCC1; .NET CLR 2.0.50727; Media Center PC
 5.0; .NET,wcVWWTascoPbGt6bdqDbuWTPPHgOPs,69.191.224.234,2009-04-13-08-05)
(1239611000000,NjriQjdODgWBKnkGJUP6GNTbDeK4An,AWtXPkfaWGOaNeL9OOsFU8Hcj6eLHt,cartoonnetwork
 (compatible; MSIE 7.0; Windows NT 5.1; GTB6; .NET CLR
 1.1.4322),OaMU1F2gE4CtADVHAbKjjRRks5kIgg,57.34.133.110,2009-04-13-08-05)
(1239610462000,Irpv3oiu0I5QNQiwSSTIshrLdo9cM1,i1LDq44LRSJF0hbmhB8Gk7k9gMWtBq,cartoonnetwork
 (compatible; MSIE 6.0; Windows NT 5.2; SV1; .NET CLR 1.1.4322;
 InfoPath.1),QSb3wkLR4JAIut4Uq6FNFQIR1rCVwU,42.174.193.253,2009-04-13-08-05)
(1239611007000,q2Awfnpe0JAvhInaIp0VGx9KTs0oPO,s3HvTflPB8JIE0IuM6hOEebWWpOtJV,cartoonnetwork
 (compatible; MSIE 6.0; Windows NT 5.2; SV1; .NET CLR 1.1.4322;
 InfoPath.1),QSb3wkLR4JAIut4Uq6FNFQIR1rCVwU,42.174.193.253,2009-04-13-08-05)
(1239610398000,c362vpAB0soPKGHRS43cj6TRwNeOGn,jeas5nXbQInGAgFB8jlkhnprN6cMw7,cartoonnetwork
 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB6; .NET CLR
 1.1.4322),k96n5PnUmwHKfiUI0TFP0TNMfADgh9,51.131.29.87,2009-04-13-08-05)
7120 [main] INFO  org.apache.pig.Main  - Pig script completed in 7 seconds
 and 199 milliseconds (7199 ms)
16/03/08 23:17:10 INFO pig.Main: Pig script completed in 7 seconds and 199
 milliseconds (7199 ms)
```

```
cartoonnetwork.com|Mozilla/4.0 (comp...|wcVWWTascoPbGt6bd...|69.191.224.234|
2009-04-13-08-05|
|   1239611000000|NjriQjdODgWBKnkGJ...|AWtXPkfaWGOaNeL9O...|
cartoonnetwork.com|Mozilla/4.0 (comp...|QaMU1F2qE4CtADVHA...| 57.34.133.110|
2009-04-13-08-05|
|   1239610462000|Irpv3oiu0I5QNQiwS...|i1LDq44LRSJF0hbmh...|
cartoonnetwork.com|Mozilla/4.0 (comp...|QSb3wkLR4JAIut4Uq...|42.174.193.253|
2009-04-13-08-05|
|   1239611007000|q2AwfnpeOJAvhInaI...|s3HvTflPB8JIE0IuM...|
cartoonnetwork.com|Mozilla/4.0 (comp...|QSb3wkLR4JAIut4Uq...|42.174.193.253|
2009-04-13-08-05|
|   1239610398000|c362vpAB0soPKGHRS...|jeas5nXbQInGAgFB8...|
cartoonnetwork.com|Mozilla/4.0 (comp...|k96n5PnUmwHKfiUI0...|  51.131.29.87|
2009-04-13-08-05|
|   1239610600000|cjBTpruoaiEtqLuMX...|XwlohBSs8Ipxs1bRa...|
cartoonnetwork.com|Mozilla/4.0 (comp...|k96n5PnUmwHKfiUI0...|  51.131.29.87|
2009-04-13-08-05|
|   1239610804000|Ms3eJHNAEItpxvimd...|4SIj4pGmgVLl625BD...|
cartoonnetwork.com|Mozilla/4.0 (comp...|k96n5PnUmwHKfiUI0...|  51.131.29.87|
2009-04-13-08-05|
|   1239610872000|h5bccHX6wJReDi1jL...|EFAWIiBdVfnxwAMWP...|
cartoonnetwork.com|Mozilla/4.0 (comp...|k96n5PnUmwHKfiUI0...|  51.131.29.87|
2009-04-13-08-05|
|   1239610365000|874NBpGmxNFfxEPKM...|xSvE4XtGbdtXPF2Lb...|
cartoonnetwork.com|Mozilla/5.0 (Maci...|eWDEVVUphlnRa273j...| 22.91.173.232|
2009-04-13-08-05|
|   1239610348000|X8gISpUTSqh1A5reS...|TrFblGT99AgE75vuj...|
 corriere.it|Mozilla/4.0 (comp...|tX1sMpnhJUhmAF7AS...|   55.35.44.79|
2009-04-13-08-05|
|   1239610743000|kbKreLWB6QVueFrDm...|kVnxx9Ie2i3OLTxFj...|
 corriere.it|Mozilla/4.0 (comp...|tX1sMpnhJUhmAF7AS...|   55.35.44.79|
2009-04-13-08-05|
|   1239610812000|9lxOSRpEi3bmEeTCu...|1B2sff99AEIwSuLVV...|
 corriere.it|Mozilla/4.0 (comp...|tX1sMpnhJUhmAF7AS...|   55.35.44.79|
2009-04-13-08-05|
|   1239610876000|lijjmCf2kuxfBTnjL...|AjvufgUtakUFcsIM9...|
 corriere.it|Mozilla/4.0 (comp...|tX1sMpnhJUhmAF7AS...|   55.35.44.79|
2009-04-13-08-05|
|   1239610941000|t8t8trgjNRPIlmxuD...|agu2u2TCdqWP08rAA...|
 corriere.it|Mozilla/4.0 (comp...|tX1sMpnhJUhmAF7AS...|   55.35.44.79|
2009-04-13-08-05|
|   1239610490000|OGRLPVNGxiGgrCmWL...|mJg2raBUpPrC8OlUm...|
 corriere.it|Mozilla/4.0 (comp...|r2k96t1CNjSU9fJKN...|   71.124.66.3|
2009-04-13-08-05|
|   1239610556000|OnJID12x0RXKPUgrD...|P7Pm2mPdW6wO8KA3R...|
 corriere.it|Mozilla/4.0 (comp...|r2k96t1CNjSU9fJKN...|   71.124.66.3|
2009-04-13-08-05|
|   1239610373000|WflsvKIgOqfIE5KwR...|TJHd1VBspNcua0XPn...|
 corriere.it|Mozilla/5.0 (Maci...|fj2L1ILTFGMfhdrt3...| 75.117.56.155|
2009-04-13-08-05|
|   1239610768000|4MJR0XxiVCU1ueXKV...|1OhGWmbvKf8ajoU8a...|
 corriere.it|Mozilla/5.0 (Maci...|fj2L1ILTFGMfhdrt3...| 75.117.56.155|
2009-04-13-08-05|
|   1239610832000|gWIrpDiN57i3sHatv...|RNL4C7xPi3tdar2Uc...|
 corriere.it|Mozilla/5.0 (Maci...|fj2L1ILTFGMfhdrt3...| 75.117.56.155|
2009-04-13-08-05|
|   1239610789000|pTne9k62kJ14QViXI...|RVxJVIQousjxUVI3r...|
 pixnet.net|Mozilla/5.0 (Maci...|1bGOKiBD2xmui9OkF...| 33.176.101.80|
2009-04-13-08-05|
+---------------+--------------------+--------------------
+------------------+--------------------+--------------------+--------------
+----------------+
only showing top 20 rows


scala>
```

Example Accessing the Table using Spark SQL

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, as well as the ability to customize query processing by creating table schemas 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/.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Hive 2.1.0	emr-5.2.1	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-resourcemanager, hadoop-yarn-timeline-server, hive-client, hcatalog-server, hive-server, mysql-server, tez-on-yarn

Samples

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`.

Topics

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

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,recordTimeStamp: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
 recordTimeStamp < "2012-01-01";
```

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

Query example	Result description
SELECT * FROM *table_name*;	Selects all items (rows) from a given table and includes data from all columns available for those items.
SELECT * FROM *table_name* WHERE *field_name* =*value*;	Selects some items (rows) from a given table and includes data from all columns available for those items.
SELECT *column1_name*, *column2_name*, *column3_name* FROM *table_name*;	Selects all items (rows) from a given table and includes data from some columns available for those items.
SELECT *column1_name*, *column2_name*, *column3_name* FROM *table_name* WHERE *field_name* =*value*;	Selects some items (rows) from a given table and includes data from some columns available for those items.

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.

```
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,recordTimeStamp: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.

```
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,recordTimeStamp:RangeKey");
```

Create a Hive Metastore Outside the Cluster

Hive records metastore information in a MySQL database that is located, by default, on the master node. The metastore contains a description of the input data, including the partition names and data types, contained in the input files.

When a cluster terminates, all associated cluster nodes shut down. All data stored on a cluster node, including the Hive metastore, is deleted. Information stored elsewhere, such as in your Amazon S3 bucket, persists.

If you have multiple clusters that share common data and update the metastore, you should locate the shared metastore on persistent storage.

To share the metastore between clusters, override the default location of the MySQL database to an external persistent storage location on an Amazon RDS 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 cluster

1. Create a MySQL database.

   Relational Database Service (RDS) provides a cloud-based MySQL database. For information about how to create an Amazon RDS database, see https://aws.amazon.com/rds/.
2. Modify your security groups to allow JDBC connections between your MySQL 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.
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`:

```
[
    {
      "Classification": "hive-site",
      "Properties": {
```

```
        "javax.jdo.option.ConnectionURL": "jdbc:mysql:\/
\/hostname:3306\/hive?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-5.2.1 --instance-type
 m3.xlarge --instance-count 2 \
--applications Name=Hive --configurations ./hiveConfiguration.json --
use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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-5.2.1 --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 MySQL. <*username*> and <*password*> are the credentials for your MySQL database. `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.

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
)
LOCATION s3://mybucket/hdfs/
```

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 an ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity) driver. Amazon EMR supports both JDBC and ODBC connectivity.

To connect to Hive via JDBC requires you to download the JDBC driver and install a SQL client. The following example demonstrates using SQL Workbench/J to connect to Hive using JDBC.

To download JDBC drivers

Download and extract the drivers for Amazon EMR 4.x releases at the following location:

- Hive 1.0 JDBC drivers (driver version 1.0.4): https://amazon-odbc-jdbc-drivers.s3.amazonaws.com/public/AmazonHiveJDBC_1.0.4.1004.zip

To install and configure SQL Workbench

1. Download the SQL Workbench/J client for your operating system from http://www.sql-workbench.net/downloads.html.
2. Go to the Installing and starting SQL Workbench/J page and follow the instructions for installing SQL Workbench/J on your system.
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.

Hive version	Command
1.0	`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.

Hive version	Tunnel settings
1.0	**Source port**: 10000 **Destination**: *master-public-dns-name*:10000

4. Add the JDBC driver to SQL Workbench/J.

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, go to http://wiki.apache.org/hadoop/Hive/HiveClient and http://wiki.apache.org/hadoop/Hive/HiveJDBCInterface.

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, go to http://gethue.com.

Supported and unsupported features of Hue on Amazon EMR

Hue on Amazon EMR supports the following:

- 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.
- 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, which 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.

Hue on Amazon EMR does not support the following:

- Livy Server
- Spark notebook functionality

Using Hue or the AWS Management Console

Cluster administrators use the AWS Management Console to launch and administer clusters. This is also the case when you want to launch a cluster with Hue installed. On the other hand, end users may

interact entirely with their Amazon EMR cluster through an application such as Hue. Hue acts as a front end for the applications on the cluster and it allows users to interact with their cluster in a more user-friendly interface. The applications in Hue, such as the Hive and Pig editors, replace the need to log in to the cluster to run scripts interactively with their respective shell applications.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Hue 3.10.0	emr-5.2.1	emrfs, emr-ddb, emr-goodies, emr-kinesis, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hue-server, mysql-server, oozie-client, oozie-server

Topics

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.

```
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.

```
aws emr create-cluster --name "Hue cluster" --release-label emr-5.2.1 \
--applications Name=Hue Name=Hive Name=Pig --use-default-roles --ec2-
attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

Note
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 will use the instance type specified in the command.

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

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:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=Hue
 Name=Spark Name=Hive \
--instance-type m3.xlarge --instance-count 2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

> **Note**
> For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.json`:

```
[{
  "Classification": "hue-ini",
  "Properties": {},
  "Configurations": [
    {
      "Classification": "desktop",
      "Properties": {},
      "Configurations": [
        {
          "Classification": "database",
          "Properties": {
            "name": "dbname",
```

```
            "user": "username",
            "password": "password",
            "host": "hueinstance.c3b8apyyjyzi.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.

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. 99)

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: `<login username>@nt_domain`. This template is used to search all directory subtrees beginning at the base distinguished name. If the nt domain is not configured, Hue searches for an exact distinguished name pattern for the user (defined by the `ldap_username_pattern` configuration setting). In this instance, the server searches for a matching `ldap_username_pattern` value in all directory subtrees beginning at the base distinguished name.

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:

```
aws emr create-cluster --release-label emr-5.2.1 --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`:

```
[
  {
    "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": []
                }
              ]
            }
          ]
        }
      ]
    }
  ]
```

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 Mahout

Amazon EMR (Amazon EMR) supports Apache Mahout, a machine learning framework for Hadoop. For more information about Mahout, go to 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 post on the AWS Big Data blog.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Mahout 0.12.2	emr-5.2.1	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-resourcemanager, mahout-client

Apache Oozie

Use the Apache Oozie Workflow Scheduler to manage and coordinate Hadoop jobs.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Oozie 4.2.0	emr-5.2.1	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-resourcemanager, hadoop-yarn-timeline-server, oozie-client, oozie-server, tez-on-yarn

For more information about Apache Oozie, see http://oozie.apache.org/.

Important
The Oozie native web interface is not supported on Amazon EMR. If you would like to use a front-end interface for Oozie, try the Hue Oozie application.

Apache Phoenix

Apache Phoenix is an application used for OLTP workloads and low-level latency SQL. Phoenix uses Apache HBase as its backing store and you can connect to it using a JDBC driver bundled with Phoenix. For more information, see https://phoenix.apache.org/.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Phoenix 4.7.0	emr-5.2.1	emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-mapred, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hbase-hmaster, hbase-client, hbase-region-server, phoenix-library, phoenix-query-server, zookeeper-client, zookeeper-server

Creating a Cluster with Phoenix

Install Phoenix by choosing that application when you create the cluster. The following procedure creates a cluster with Phoenix and HBase installed. For more information about launching clusters with the console, see Step 3: Launch an Amazon EMR Cluster in the *Amazon EMR Management Guide*.

To launch a cluster with Phoenix installed using 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 **Amazon Release Version emr-4.7.0** or later.
4. For **Select Applications**, choose either **All Applications** or **Phoenix** and **HBase**.

   Note
   Selecting Phoenix always includes and installs HBase components, but this is made explicit in examples.
5. Select other options as necessary and then choose **Create cluster**.

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

- Create the cluster with the following command:

```
aws emr create-cluster --name "Cluster with Phoenix" --release-
label emr-5.2.1 \
--applications Name=Phoenix Name=HBase --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

Configuring Phoenix

You configure Phoenix by setting values in `hbase-site.xml` using the `hbase-site` configuration classification when you create your cluster.

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

To change a setting in Phoenix

- Create a cluster with Phoenix and HBase installed and set `phoenix.schema.dropMetaData` to false, using the following command:

```
aws emr create-cluster --release-label emr-5.2.1 --applications
 Name=Phoenix \
Name=HBase --instance-type m3.xlarge --instance-count 2 --configurations
 https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.json`:

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

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-5.2.1 --applications
 Name=Phoenix Name=Hive \
Name=HBase --instance-type m3.xlarge --instance-count 3 --
configurations https://s3.amazonaws.com/mybucket/myfolder/
myConfig.json
```

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:

```
aws emr create-cluster --name "Cluster with Phoenix" --log-uri
 s3://myBucket/myLogFolder --release-label emr-5.2.1 \
--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:

   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/.hbase-snapshot \
-snapshot customer_snapshot1 \
-copy-to hdfs://masterDNSName:8020/user/hbase \
-mappers 2 -chuser hbase -chmod 700'
```

   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
```

   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:

```
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:

   createTable.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"
     }
   ]
   ```

   ```
   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:

   ```
   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:

   restoreSnapshot.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"
  }
]
```

```
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:

```
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF \
--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:

```
+----------------------------------------
+-----------------------------------+
|             CREDIT_CARD_TYPE             |              NUM_CUSTOMERS
     |
+----------------------------------------
+-----------------------------------+
| american_express                         | 5728
     |
| dankort                                  | 5782
     |
| diners_club                              | 5795
     |
| discover                                 | 5715
     |
| forbrugsforeningen                       | 5691
     |
| jcb                                      | 5762
     |
| laser                                    | 5769
     |
| maestro                                  | 5816
     |
| mastercard                               | 5697
     |
| solo                                     | 5586
     |
| switch                                   | 5781
     |
| visa                                     | 5659
     |
+----------------------------------------
+-----------------------------------+
```

Apache Pig

Amazon EMR (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*.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Pig 0.16.0	emr-5.2.1	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-resourcemanager, hadoop-yarn-timeline-server, pig-client, tez-on-yarn

Submit Pig Work

This section demonstrates submitting Pig work to an Amazon EMR cluster. The examples that follow are based on the Amazon EMR sample: Apache Log Analysis using Pig. The sample evaluates Apache log files and then generates 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

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

with this:

```
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/.
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-5.2.1
 --applications Name=Pig \
```

```
--use-default-roles --ec2-attributes KeyName=myKey --instance-
type m3.xlarge --instance-count 3 \
--steps Type=PIG,Name="Pig Program",ActionOnFailure=CONTINUE,Args=[-
f,s3://elasticmapreduce/samples/pig-apache/do-reports2.pig,-
p,INPUT=s3://elasticmapreduce/samples/pig-apache/input,-p,OUTPUT=s3://
mybucket/pig-apache/output]
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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 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 about using Amazon EMR commands in the AWS CLI, see http://docs.aws.amazon.com/cli/latest/reference/emr.

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, go to http://pig.apache.org/docs/r0.14.0/udf.html.

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/.

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. 109).
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.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Presto 0.157.1	emr-5.2.1	emrfs, emr-goodies, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hive-client, hcatalog-server, mysql-server, presto-coordinator, presto-worker

For more information about Presto, go to https://prestodb.io/.

Note

- 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`
  - `jvm.config`

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

Adding Database Connectors

You can add JDBC connectors at cluster launch using the configuration classifications. The connectors currently available match those supported by Presto. For further information about connectors, see https://prestodb.io/docs/current/connector.html.

These classifications are called:

- presto-connector-blackhole
- presto-connector-cassandra
- presto-connector-hive
- presto-connector-jmx
- presto-connector-kafka
- presto-connector-localfile
- presto-connector-mongodb
- presto-connector-mysql
- presto-connector-postgresql
- presto-connector-raptor
- presto-connector-redis
- presto-connector-tpch

Example Configuring a Cluster with the PostgreSQL JDBC

To launch a cluster with the PostgreSQL connector installed and configured create a file, `myConfig.json`, like the following:

```
[
  {
    "Classification": "presto-connector-postgresql",
    "Properties": {
      "connection-url": "jdbc:postgresql://example.net:5432/database",
      "connection-user": "MYUSER",
      "connection-password": "MYPASS"
    },
    "Configurations": []
  }
]
```

Then use the following to create the cluster:

```
aws emr create-cluster --name PrestoConnector --release-label emr-5.2.1 --
instance-type m3.xlarge \
--instance-count 2 --applications Name=Hadoop Name=Hive Name=Pig Name=Presto
 \
--use-default-roles --no-auto-terminate --ec2-attributes KeyName=myKey \
--log-uri s3://my-bucket/logs --enable-debugging \
--configurations file://./myConfig.json
```

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.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Spark 2.0.2	emr-5.2.1	emrfs, emr-goodies, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave

Topics

Create a Cluster With Spark

To launch a cluster with Spark installed using the console

The following procedure creates a cluster with Spark 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-5.2.1*** 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. 117).

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-5.2.1 --
applications Name=Spark \
--ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count
 3 --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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:

```
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-5.2.1")
    .withInstances(new JobFlowInstancesConfig()
    .withEc2KeyName("myKeyName")
    .withInstanceCount(1)
        .withKeepJobFlowAliveWhenNoSteps(true)
        .withMasterInstanceType("m3.xlarge")
        .withSlaveInstanceType("m3.xlarge")
    );
RunJobFlowResult result = emr.runJobFlow(request);
```

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. 118).

Topics

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

Setting	Description	Value
spark.executor.memory	Amount of memory to use per executor process. (for example, 1g, 2g)	Setting is configured based on the slave instance types in the cluster.
spark.executor.cores	The number of cores to use on each executor.	Setting is configured based on the slave instance types in the cluster.
spark.dynamicAllocation.enabled	Whether to use dynamic resource allocation, which scales the number of executors	true (emr-4.4.0 or greater)

Setting	Description	Value
	registered with an application up and down based on the workload.	**Note** Spark Shuffle Service is automatically configured by Amazon EMR.

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

Setting	Description	Value
spark.default.parallelism	Default number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize when not set by user.	2X number of CPU cores available to YARN containers.
spark.driver.memory	Amount of memory to use for the driver process, i.e. where SparkContext is initialized. (for example, 1g, 2g).	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.
spark.executor.memory	Amount of memory to use per executor process. (for example, 1g, 2g)	Setting is configured based on the slave instance types in the cluster.
spark.executor.cores	The number of cores to use on each executor.	Setting is configured based on the slave instance types in the cluster.
spark.executor.instances	The number of executors.	Setting is configured based on the slave instance types in the cluster. Set unless `spark.dynamicAllocation.enabled` explicitly set to true at the same time.

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.

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-5.2.1 --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
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.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-5.2.1 --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-5.2.1 --applications Name=Spark
 \
--instance-type m3.xlarge --instance-count 2 --service-role
 EMR_DefaultRole --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole --
configurations file://./myConfig.json
```

Or using Amazon S3:

```
aws emr create-cluster --release-label emr-5.2.1 --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
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.json`:

```
[
  {
    "Classification": "spark",
    "Properties": {
      "maximizeResourceAllocation": "true"
    }
  }
]
```

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

scala> val textFile = sc.textFile("s3://elasticmapreduce/
samples/hive-ads/tables/impressions/dt=2009-04-13-08-05/
ec2-0-51-75-39.amazon.com-2009-04-13-08-05.log")
```

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()
linesWithCartoonNetwork: org.apache.spark.rdd.RDD[String] =
 MapPartitionsRDD[2] at filter at <console>:23
<snip>
<Spark program runs>
scala> linesWithCartoonNetwork
res2: Long = 9
```

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.

```
>>> sc
<pyspark.context.SparkContext object at 0x7fe7e659fa50>
>>> textfile = sc.textFile("s3://elasticmapreduce/
samples/hive-ads/tables/impressions/dt=2009-04-13-08-05/
ec2-0-51-75-39.amazon.com-2009-04-13-08-05.log")
```

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".

```
>>> linesWithCartoonNetwork = textfile.filter(lambda line:
 "cartoonnetwork.com" in line).count()
15/06/04 17:12:22 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
 from the embedded binaries
15/06/04 17:12:22 INFO lzo.LzoCodec: Successfully loaded & initialized
 native-lzo library [hadoop-lzo rev EXAMPLE]
15/06/04 17:12:23 INFO fs.EmrFileSystem: Consistency disabled, using
 com.amazon.ws.emr.hadoop.fs.s3n.S3NativeFileSystem as filesystem
 implementation
<snip>
<Spark program continues>
>>> linesWithCartoonNetwork
9
```

Write a Spark Application

Spark applications can be written in Scala, Java, or Python. The 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

```
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

```
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 `script-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/.
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`.

```
aws emr create-cluster --name "Add Spark Step Cluster" --release-
label emr-5.2.1 --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`:

```
aws emr create-cluster --name "Add Spark Step Cluster" --release-
label emr-5.2.1 \
--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
For Windows, replace the above Linux line continuation character (\) with the caret (^).

2. Alternatively, add steps to a cluster already running. Use `add-steps`.

```
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF --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`:

```
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:

```
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:

```
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:

```
--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.

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
Currently, the FlinkKinesisConsumer class does not work on Amazon EMR.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Flink 1.1.3	emr-5.2.1	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, flink-client

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:

```
aws emr create-cluster --name "Cluster with Flink" --release-
label emr-5.2.1 \
--applications Name=Flink --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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:

```
[
    {
      "Classification": "flink-conf",
      "Properties": {
        "taskmanager.numberOfTaskSlots":"2"
      }
    }
]
```

2. Next, create a cluster with the following configuration:

```
aws emr create-cluster --release-label emr-5.2.1 \
--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

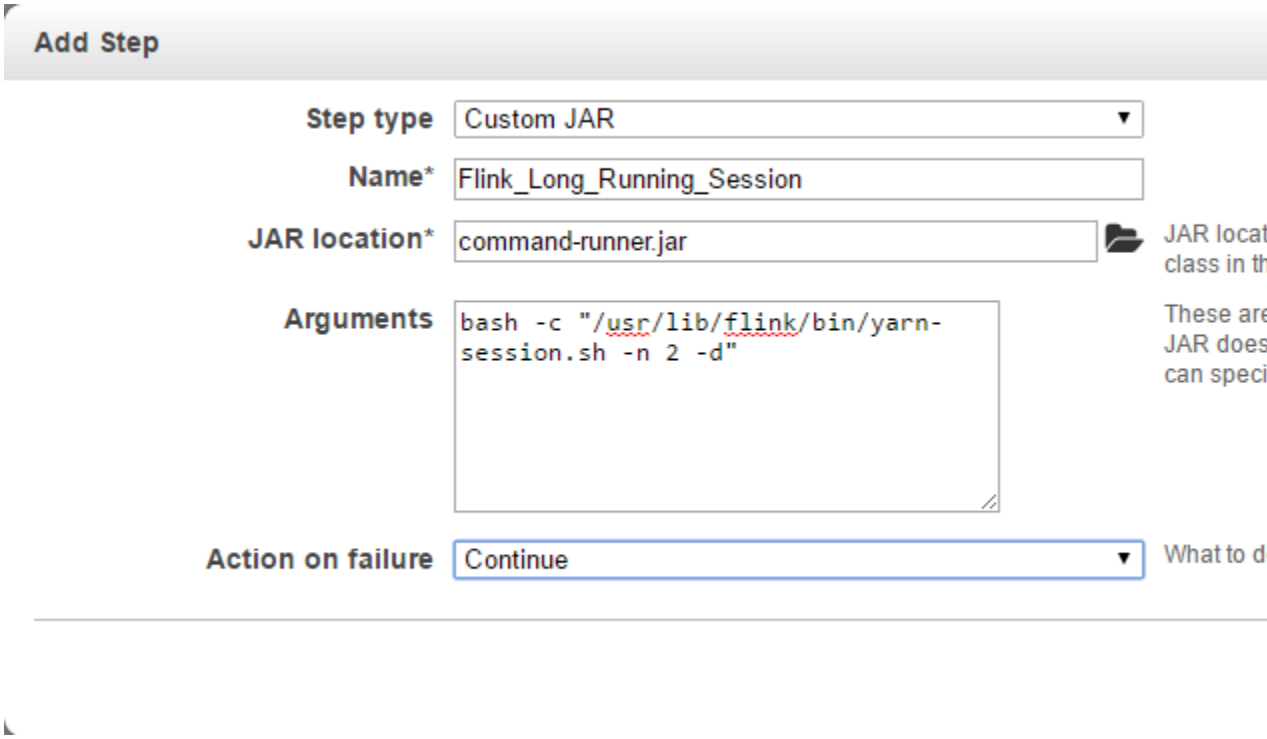
You may want to start a long-running Flink job that multiple clients can submit to through YARN API operations. You start a Flink YARN session and submit jobs to the Flink JobManager, which is located on the YARN node that hosts the Flink session Application Master daemon. To start a YARN session, use 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 by running the `yarn-session.sh` Flink shell script in an existing cluster.

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 the arguments as shown and choose **Add**.



The `-d` option runs the long-running Flink cluster in a "detached" state.

To submit a long-running Flink job using the AWS CLI

- To launch a long-running Flink cluster within EMR, use the `create-cluster` command:

```
aws emr create-cluster --release-label emr-5.2.1 \
--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=MyKeyName,InstanceProfile=EMR_EC2_DefaultRole \
--steps Type=CUSTOM_JAR,Jar=command-
runner.jar,Name=Flink_Long_Running_Session,\
Args="bash","-c","/usr/lib/flink/bin/yarn-session.sh -n 2 -d""
```

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:

```
$ yarn application -list
16/09/07 19:32:13 INFO client.RMProxy: Connecting to ResourceManager at
 ip-10-181-83-19.ec2.internal/10.181.83.19:8032
Total number of applications (application-types: [] and states: [SUBMITTED,
 ACCEPTED, RUNNING]):1
Application-Id    Application-Name    Application-Type    User    Queue
 State    Final-State    Progress    Tracking-URL
application_1473169569237_0002    Flink session with 14 TaskManagers
 (detached)         Apache Flink     hadoop    default            RUNNING
      UNDEFINED            100% http://ip-10-136-154-194.ec2.internal:33089
```

SDK for Java

```
List<StepConfig> stepConfigs = new ArrayList<StepConfig>();

HadoopJarStepConfig flinkWordCountConf = new HadoopJarStepConfig()
    .withJar("command-runner.jar")
    .withArgs("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/");

StepConfig flinkRunWordCount = new StepConfig()
  .withName("Flink add a wordcount step")
  .withActionOnFailure("CONTINUE")
  .withHadoopJarStep(flinkWordCountConf);

stepConfigs.add(flinkRunWordCount);

AddJobFlowStepsResult res = emr.addJobFlowSteps(new AddJobFlowStepsRequest()
   .withJobFlowId("myClusterId")
   .withSteps(stepConfigs));
```

AWS CLI

Use the `add-steps` subcommand to submit new jobs to an existing Flink cluster:

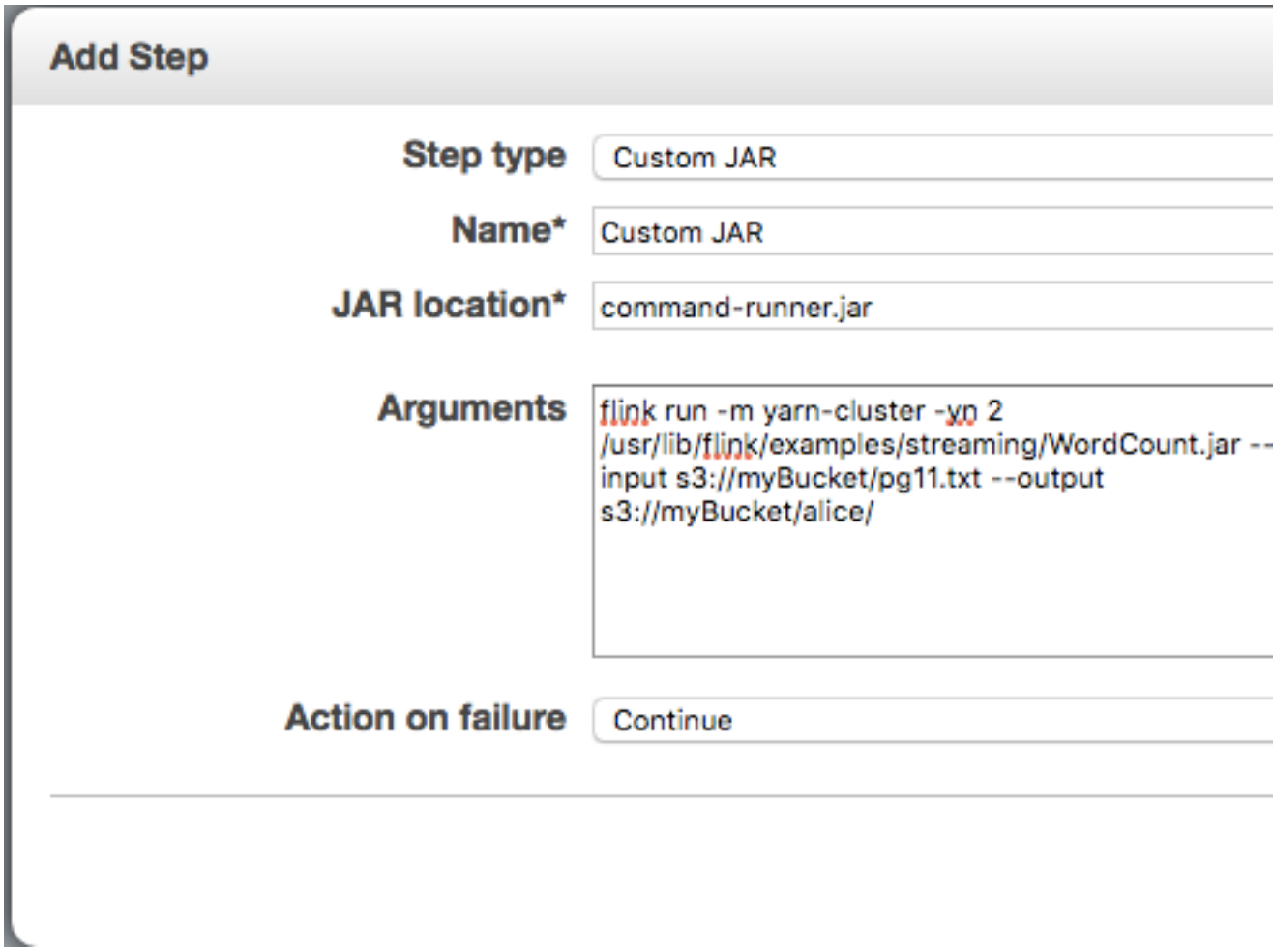
```
aws emr add-steps --cluster-id myClusterId \
--steps Type=CUSTOM_JAR,Name=Flink_Submit_To_Long_Running,Jar=command-
runner.jar,\
Args="flink","run","-m","yarn-cluster","-
yid","application_1473169569237_0002","-yn","2",\
"/usr/lib/flink/examples/streaming/WordCount.jar",\
"--input","s3://myBucket/pg11.txt","--output","s3://myBucket/alice2/" \
--region myRegion
```

Submit a Transient Flink Job

The following example launches the Flink WordCount example by adding a step to an existing cluster.

Console

In the console details page for an existing cluster, add the step by choosing **Add Step** for the **Steps** field.



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.

```
List<StepConfig> stepConfigs = new ArrayList<StepConfig>();

HadoopJarStepConfig flinkWordCountConf = new HadoopJarStepConfig()
    .withJar("command-runner.jar")
    .withArgs("flink", "run", "-m", "yarn-cluster", "-yn", "2", "/usr/lib/
flink/examples/streaming/WordCount.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);

AddJobFlowStepsResult res = emr.addJobFlowSteps(new AddJobFlowStepsRequest()
   .withJobFlowId("myClusterId")
```

```
   .withSteps(stepConfigs));
```

```
    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")
      .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);
```

AWS CLI

Use the `add-steps` subcommand to submit new jobs to an existing Flink cluster:

```
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:

```
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 http://docs.aws.amazon.com//emr/latest/ManagementGuide/emr-connect-master-node-ssh.html.
2. Type the following to start a shell:

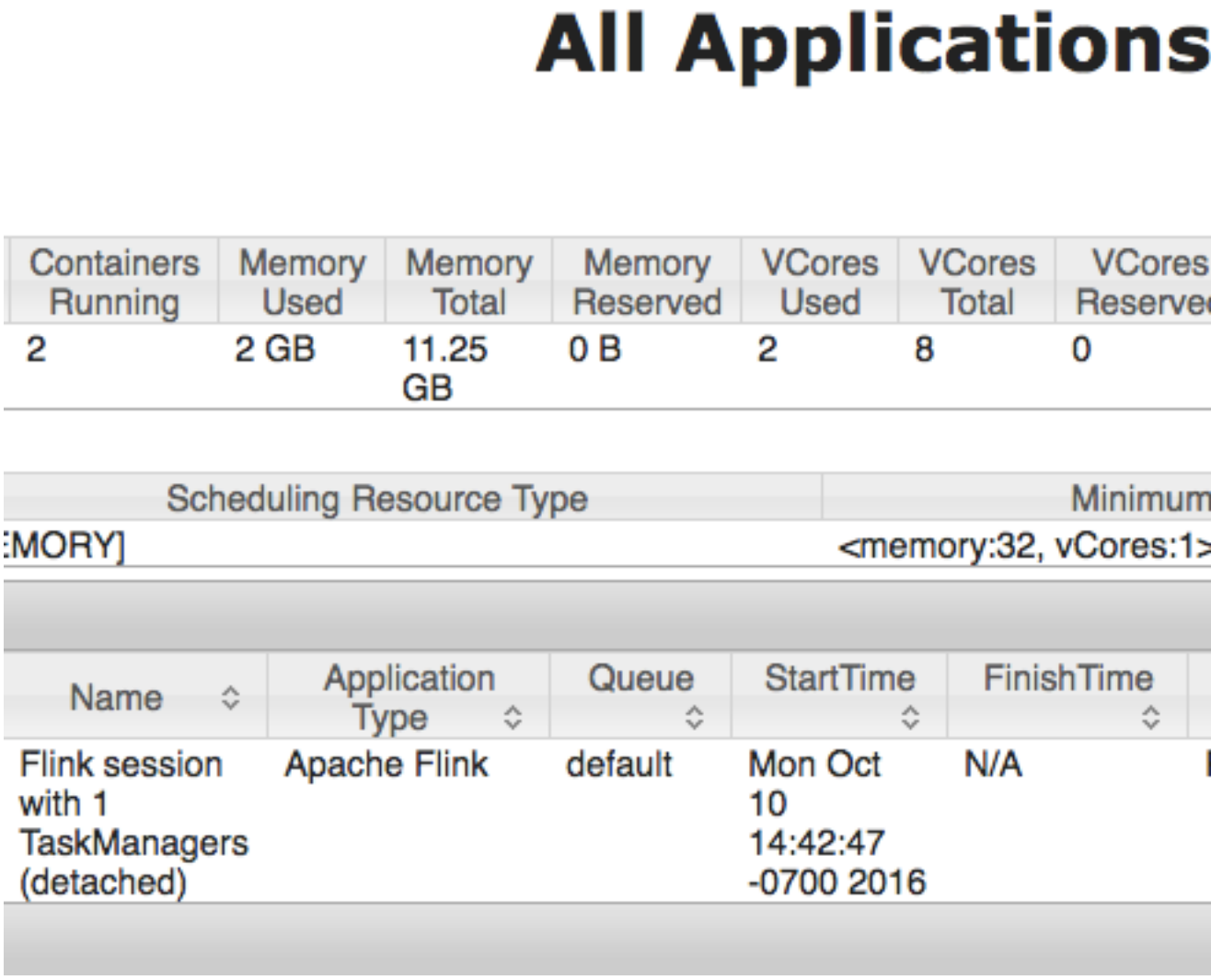
   ```
   % /usr/lib/flink/bin/start-scala-shell.sh yarn -n 1
   ```

   This will start 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.

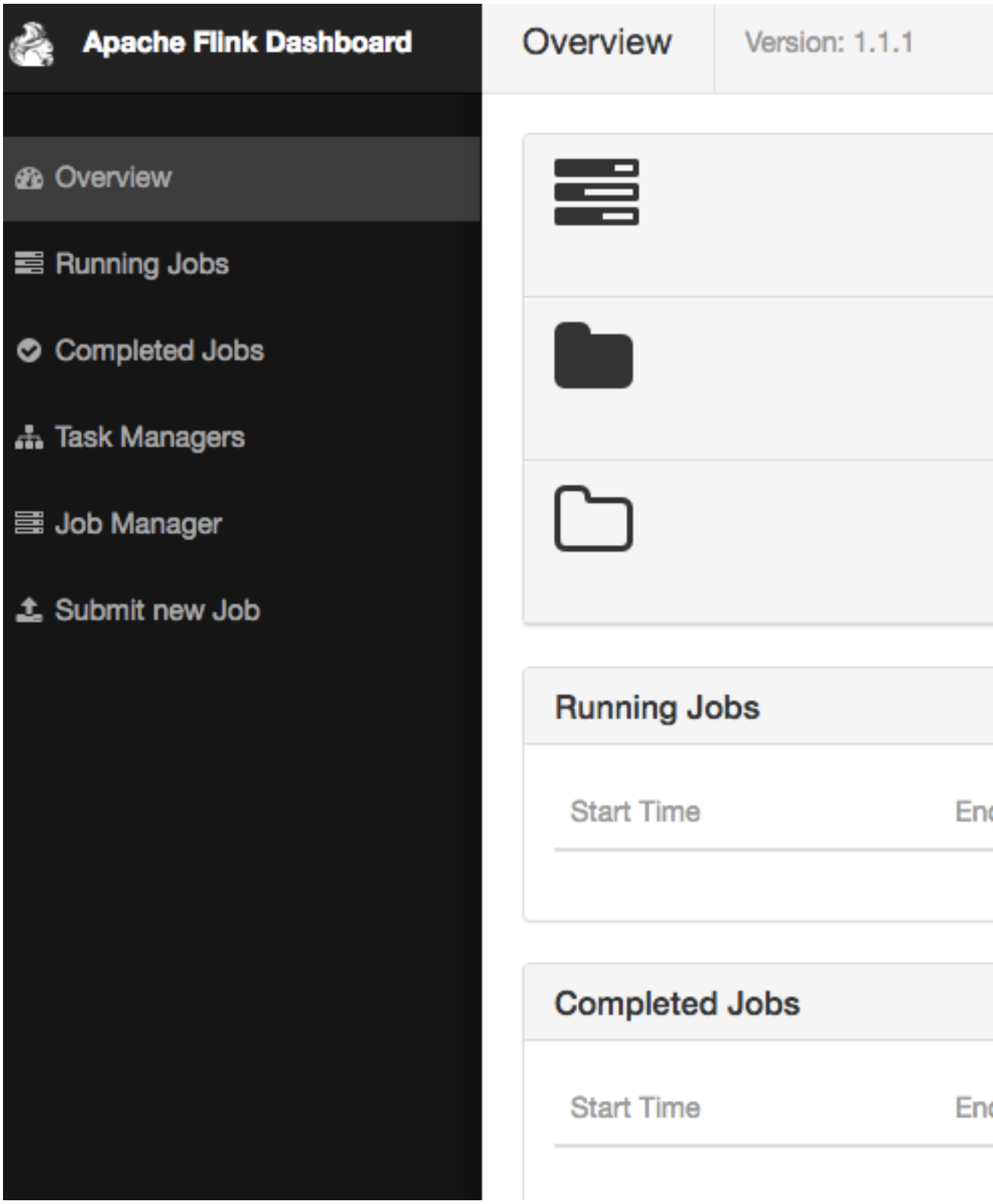
Finding the Flink Web Interface

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 Flink Dashboard
Overview
Version: 1.1.1
Overview
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit new Job
Running Jobs
Start Time
Completed Jobs
Start Time

Apache Sqoop

Sqoop is a tool for transferring data between Amazon S3, Hadoop, HDFS, and RDBMS databases.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Sqoop 1.4.6	emr-5.2.1	emrfs, emr-ddb, emr-goodies, 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, mysql-server, sqoop-client

By default, Sqoop has a MariaDB and PostgresSQL driver installed. The PostgresSQL driver installed for Sqoop will only work for PostgreSQL 8.4. To install an alternate set of JDBC connectors for Sqoop, you need to install them in `/usr/lib/sqoop/lib`. The following are links for various JDBC connectors:

- MariaDB: About MariaDB Connector/J.
- PostgreSQL: Version 9.4-1208 Released.
- SQLServer: Microsoft JDBC Drivers 6.0 (Preview), 4.2, 4.1, and 4.0 for SQL Server.
- MySQL: Download Connector/J
- Oracle: Get Oracle JDBC drivers and UCP from the Oracle Maven Repository

Sqoop's supported databases are shown here: http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_supported_databases. 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):

```
sqoop export --connect jdbc:redshift://$MYREDSHIFTHOST:5439/mydb --
table mysqoopexport --export-dir s3://mybucket/myinputfiles/ --driver
 com.amazon.redshift.jdbc41.Driver --username master --password Mymasterpass1
```

You can use both the MariaDB and MySQL connection strings but if you specify the MariaDB connection string, you need to specify the driver:

```
sqoop export --connect jdbc:mariadb://$HOSTNAME:3306/mydb --
table mysqoopexport --export-dir s3://mybucket/myinputfiles/ --driver
 org.mariadb.jdbc.Driver --username master --password Mymasterpass1
```

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:

```
sqoop export --connect jdbc:mariadb://$HOSTNAME:3306/mydb?
verifyServerCertificate=false&useSSL=true&requireSSL=true --
table mysqoopexport --export-dir s3://mybucket/myinputfiles/ --driver
 org.mariadb.jdbc.Driver --username master --password Mymasterpass1
```

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/.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Tez 0.8.4	emr-5.2.1	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

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:

```
aws emr create-cluster --name "Cluster with Tez" --release-label emr-5.2.1
 \
--applications Name=Tez --ec2-attributes KeyName=myKey \
--instance-type m3.xlarge --instance-count 3 --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

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:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=Tez \
--instance-type m3.xlarge --instance-count 2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.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:

```
aws emr create-cluster --release-label emr-5.2.1 --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:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=Tez
 Name=Hive \
--instance-type m3.xlarge --instance-count 2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

   Note
   For Windows, replace the above Linux line continuation character (\) with the caret (^).

   `myConfig.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:

```
[
    {
      "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. 139). 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:

```
<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_dcdc075f-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:

```
http://masterDNS:8080/tez-ui
```

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:8188
```

For more information, see View Web Interfaces Hosted on Amazon EMR Clusters in the *Amazon EMR Management Guide*.

Apache Zeppelin

Use Apache Zeppelin as an interactive notebook that enables interactive data exploration.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
Zeppelin 0.6.2	emr-5.2.1	emrfs, emr-goodies, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave, zeppelin-server

For more information about Apache Zeppelin, go to https://zeppelin.apache.org/.

Note

- 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 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

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Release Information

Application	Amazon EMR Release Label	Components installed with this application
ZooKeeper 3.4.9	emr-5.2.1	emrfs, emr-goodies, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, zookeeper-client, zookeeper-server

For more information about Apache ZooKeeper, see http://zookeeper.apache.org/.

Data Encryption

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

Beginning with Amazon EMR version 4.8.0, you can use Amazon EMR security configurations to configure data encryption settings for clusters more easily. In earlier versions of Amazon EMR, you had to specify Amazon S3 encryption options individually as part of a cluster configuration. We recommend using security configurations because it simplifies setup, allows you to reuse security configurations, and provides additional encryption options.

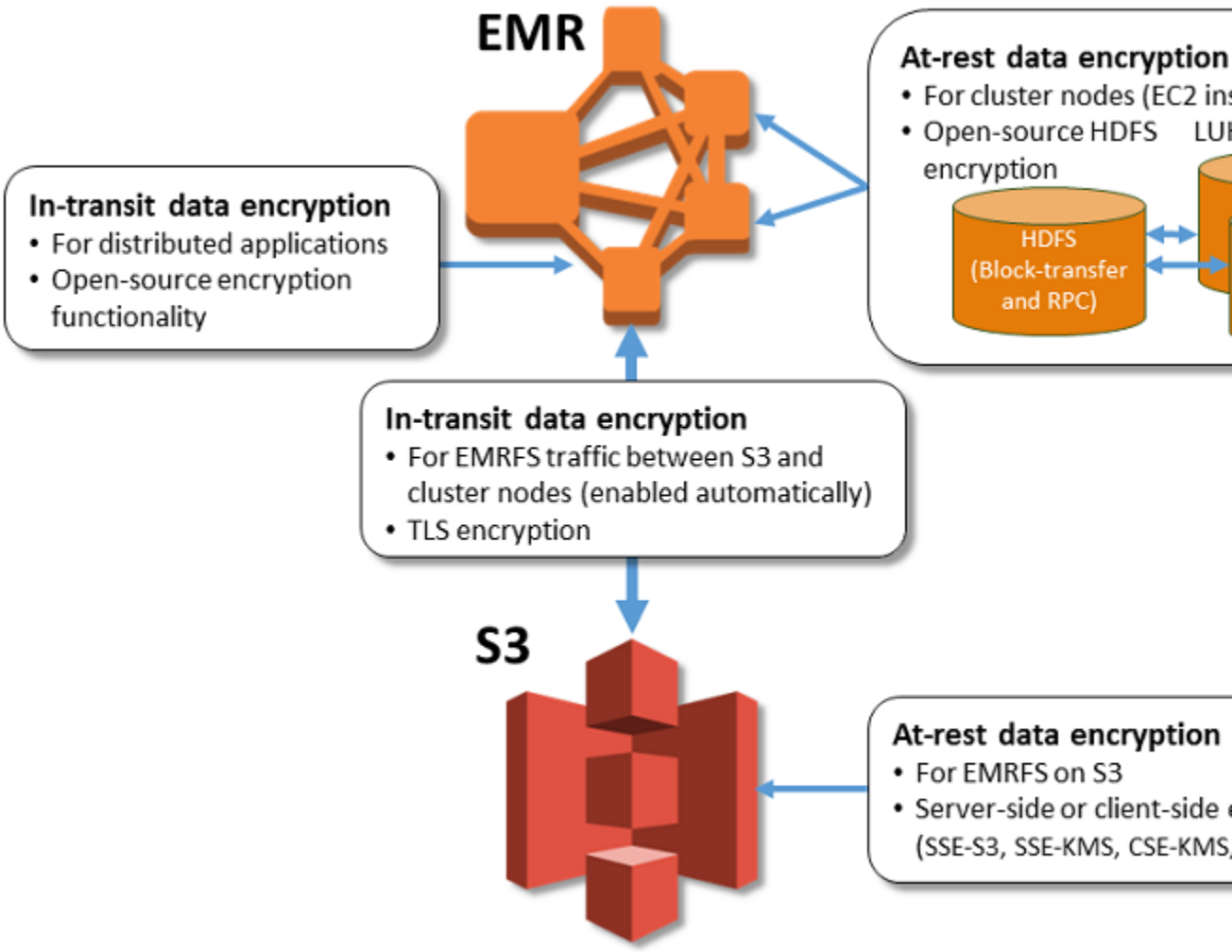
Data encryption works in tandem with access control. A solid defense strategy includes both components. For more information about setting up access control, see Configure Access to the Cluster in the *Amazon EMR Management Guide*.

Topics

Understanding Encryption Options with Amazon EMR

Amazon EMR enables you to use a security configuration to specify settings for Amazon S3 encryption with EMR file system (EMRFS), local disk encryption, and in-transit encryption. You create a security configuration that specifies encryption settings and then use the security configuration when you create a cluster.

The following diagram shows the different data encryption options available with security configurations.



You can use a security configuration to encrypt data at-rest, data in-transit, or both. Each security configuration is stored in Amazon EMR rather than in the cluster configuration, so you can easily reuse a configuration to specify data encryption settings whenever a cluster is created.

Data encryption requires keys and certificates. A security configuration gives you the flexibility to choose from several options, including keys managed by AWS Key Management Service, keys managed by Amazon S3, and keys and certificates from custom providers that you supply.

When using AWS KMS as your key provider, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

You can use the Amazon EMR console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs to create security configurations and to enable encryption options when a cluster is created. Before you specify encryption options, decide on the key and certificate management systems you want to use, so you can first create the keys and certificates or the custom providers that you specify as part of encryption settings.

Amazon S3 encryption and local disk encryption options are specified together when you configure at-rest encryption. You can choose to enable only at-rest encryption, only in-transit encryption, or both.

At-rest Encryption for Amazon S3 with EMRFS

Amazon S3 encryption works with EMR File System (EMRFS) objects read from and written to Amazon S3. You specify Amazon S3 server-side encryption (SSE) or client-side encryption (CSE) when you enable at-rest encryption. Amazon S3 SSE and CSE encryption with EMRFS are mutually exclusive; you can choose either but not both. Regardless of whether Amazon S3 encryption is enabled, transport layer security (TLS) encrypts the EMRFS objects in-transit between Amazon EMR cluster nodes and Amazon S3. For in-depth information about Amazon S3 encryption, see Protecting Data Using Encryption in the *Amazon Simple Storage Service Developer Guide*.

Amazon S3 Server-Side Encryption

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

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

- **SSE-S3**: Amazon S3 manages keys for you.
- **SSE-KMS**: You use an AWS KMS customer master key (CMK) set up with policies suitable for Amazon EMR. When you use AWS KMS, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

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

Amazon S3 Client-Side Encryption

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

> **Note**
> Amazon S3 CSE only ensures that EMRFS data exchanged with Amazon S3 is encrypted; not all data on cluster instance volumes is encrypted. Furthermore, because Hue does not use EMRFS, objects that the Hue S3 File Browser writes to Amazon S3 are not encrypted. These are important considerations if you use an Amazon EMR version earlier than 4.8.0. In later versions, Amazon S3 encryption is enabled as part of at-rest encryption, which includes local disk encryption. For more information, see Local Disk Encryption below.

At-rest Encryption for Local Disks

Two mechanisms work together to encrypt cluster instance volumes when you enable at-rest data encryption:

- **Open-source HDFS Encryption**: HDFS exchanges data between cluster instances during distributed processing, and also reads from and writes data to instance store volumes and the Elastic Block Store (EBS) volumes attached to instances. The following open-source Hadoop encryption options are activated when you enable local-disk encryption:
  - Secure Hadoop RPC is set to "Privacy", which uses Simple Authentication Security Layer (SASL).

  - Data encryption on HDFS block data transfer is set to true and is configured to use AES 256 encryption.

    Note
    You can activate additional Apache Hadoop encryption by enabling in-transit encryption (see In-Transit Data Encryption (p. 149)). These encryption settings do not activate HDFS transparent encryption, which you can configure manually. For more information, see Transparent Encryption in HDFS on Amazon EMR (p. 168).
- **LUKS**. In addition to HDFS encryption, the Amazon EC2 instance store volumes (except boot volumes) and the attached Amazon EBS volumes of cluster instances are encrypted using LUKS. For more information about LUKS encryption, see the LUKS on-disk specification..

  For your key provider, you can use an AWS KMS CMK set up with policies suitable for Amazon EMR, or a custom Java class that provides the encryption artifacts. When you use AWS KMS, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

In-Transit Data Encryption

Several encryption mechanisms are enabled with in-transit encryption. These are open-source features, are application-specific, and may vary by Amazon EMR release. In this release, the following application-specific encryption features can be enabled using security configurations:

- Hadoop (for more information, see Hadoop in Secure Mode in Apache Hadoop documentation):
  - Hadoop MapReduce Encrypted Shuffle uses TLS.
  - Secure Hadoop RPC is set to "Privacy" and uses SASL (activated in Amazon EMR when at-rest encryption is enabled).
  - Data encryption on HDFS block data transfer uses AES 256 (activated in Amazon EMR when at-rest encryption is enabled in the security configuration).
- Tez:
  - Tez Shuffle Handler uses TLS (`tez.runtime.ssl.enable`).
- Spark (for more information, see Spark security settings):
  - Akka protocol (file and broadcast server) uses TLS.
  - Block transfer service uses SASL and 3DES.
  - External shuffle service uses SASL. Applications that are not set up to use SASL encryption will fail to connect to the shuffle service.

You specify the encryption artifacts used for in-transit encryption in one of two ways: either by providing a zipped file of certificates that you upload to Amazon S3, or by referencing a custom Java class that provides encryption artifacts. For more information, see Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption (p. 152).

Enabling Data Encryption with Amazon EMR

You have two ways to enable encryption and specify options in Amazon EMR. The preferred method is to use security configurations, which are available beginning with Amazon EMR version 4.8.0 and later, or you can use a cluster configuration to specify Amazon S3 encryption with EMR file system (EMRFS). For information about using security configurations, see Specifying Encryption Options Using a Security Configuration.

Important
We recommend against specifying Amazon S3 encryption options individually with a cluster configuration. Using security configurations simplifies setup, allows you to reuse security

configurations, and provides additional encryption options. If you configure Amazon S3 encryption using both a cluster configuration and a security configuration, the security configuration overrides the cluster configuration.

Before you specify encryption options, decide on the provider you want to use for keys and encryption artifacts (for example, AWS KMS or a custom provider that you create) and create the keys or key provider as required.

Topics

Providing Keys for At-Rest Data Encryption with Amazon EMR

You can use AWS Key Management Service (AWS KMS) or a custom key provider for at-rest data encryption in Amazon EMR. When you use AWS KMS, charges apply for the storage and use of encryption keys. For more information, see AWS KMS Pricing.

This topic provides key policy details for an AWS KMS CMK to be used with Amazon EMR, as well as guidelines and code examples for writing a custom key provider class for Amazon S3 encryption. For more information about creating keys, see Creating Keys in the *AWS Key Management Service Developer Guide*.

Using AWS KMS Customer Master Keys (CMKs) for Encryption

When using SSE-KMS or CSE-KMS, the AWS KMS encryption key must be created in the same region as your Amazon EMR cluster instance and the Amazon S3 buckets used with EMRFS. In addition, the Amazon EC2 instance role used for a cluster must have permission to use the CMK you specify. The default instance role is `EMR_EC2_DefaultRole`. If you have assigned a different instance role to a cluster, make sure that the role is added as a key user, which gives the role permission to use the CMK. For more information, see Using Key Policies in the *AWS Key Management Service Developer Guide* and Create and Use IAM Roles for Amazon EMR in the *Amazon EMR Management Guide*. Although you may use the same AWS KMS customer master key (CMK) for Amazon S3 data encryption as you use for local disk encryption, using separate keys is recommended.

You can use the AWS Management Console to add your instance profile or EC2 role to the list of key users for the specified AWS KMS CMK, or you can use the AWS CLI or an AWS SDK to attach an appropiate key policy.

Add the EMR Instance Role to an AWS KMS CMK

The following procedure describes how to add the default EMR instance role, `EMR_EC2_DefaultRole` as a *key user* using the AWS Management Console. It assumes you have already created a CMK. To create a new CMK, see Creating Keys in the *AWS Key Management Service Developer Guide*.

To add the default instance role for Amazon EC2 to the list of encryption key users

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).

3. Choose the alias of the CMK to modify.
4. On the key details page under **Key Users**, choose **Add**.
5. In the **Attach** dialog box, choose the appropriate role. The default name for the role is `EMR_EC2_DefaultRole`.
6. Choose **Attach**.

Creating a Custom Key Provider

When you create a custom key provider, the application is expected to implement the EncryptionMaterialsProvider interface, which is available in the AWS SDK for Java version 1.11.0 and later. The implementation can use any strategy to provide encryption materials. You may, for example, choose to provide static encryption materials or integrate with a more complex key management system. You must use a different provider class name for local disk encryption and Amazon S3 encryption.

The EncryptionMaterialsProvider class gets encryption materials by encryption context, which is used when Amazon EMR fetches encryption materials to encrypt data. Amazon EMR populates encryption context information at runtime to help the caller determine which encryption materials to return.

Example: Using a Custom Key Provider for Amazon S3 Encryption with EMRFS

When Amazon EMR fetches the encryption materials from the EncryptionMaterialsProvider class to perform encryption, EMRFS optionally populates the materialsDescription argument with two fields: the Amazon S3 URI for the object and the JobFlowId of the cluster, which can be used by the EncryptionMaterialsProvider class to return encryption materials selectively.

For example, the provider may return different keys for different Amazon S3 URI prefixes. Note that it is the description of the returned encryption materials that is eventually stored with the Amazon S3 object rather than the materialsDescription value that is generated by EMRFS and passed to the provider. While decrypting an Amazon S3 object, the encryption materials description is passed to the EncryptionMaterialsProvider class, so that it can, again, selectively return the matching key to decrypt the object.

An EncryptionMaterialsProvider reference implementation is provided below. Another custom provider, EMRFSRSAEncryptionMaterialsProvider, is available from GitHub.

```
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.EncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.KMSEncryptionMaterials;
import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;

import java.util.Map;

/**
 * Provides KMSEncryptionMaterials according to Configuration
 */
public class MyEncryptionMaterialsProviders implements
 EncryptionMaterialsProvider, Configurable{
  private Configuration conf;
  private String kmsKeyId;
  private EncryptionMaterials encryptionMaterials;

  private void init() {
    this.kmsKeyId = conf.get("my.kms.key.id");
    this.encryptionMaterials = new KMSEncryptionMaterials(kmsKeyId);
```

```
  }

  @Override
  public void setConf(Configuration conf) {
    this.conf = conf;
    init();
  }

  @Override
  public Configuration getConf() {
    return this.conf;
  }

  @Override
  public void refresh() {

  }

  @Override
  public EncryptionMaterials getEncryptionMaterials(Map<String, String>
 materialsDescription) {
    return this.encryptionMaterials;
  }

  @Override
  public EncryptionMaterials getEncryptionMaterials() {
    return this.encryptionMaterials;
  }
}
```

Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption

For in-transit data encryption, you have to two options to specify the artifacts used for encryption:

- You can manually create PEM certificates, include them in a zip file, and then reference the zip file in Amazon S3.
- You can implement a custom certificate provider as a Java class. You specify the JAR file of the application in Amazon S3, and then provide the full class name of the provider as declared in the application. The class must implement the TLSArtifactsProvider interface available beginning with AWS SDK for Java version 1.11.0.

Amazon EMR automatically downloads artifacts to each node in the cluster and later uses them to implement the open-source, in-transit encryption features. For more information about available options, see In-Transit Data Encryption (p. 149).

Using PEM Certificates

When you specify a zip file for in-transit encryption, the security configuration expects PEM files within the zip file to be named exactly as they appear below:

In-transit encryption certificates

File name	Required/optional	Details
privateKey.pem	Required	Private key

File name	Required/optional	Details
certificateChain.pem	Required	Certificate chain
trustedCertificates.pem	Optional	Required if the provided certificate is signed neither by the Java default trusted root certification authority (CA) nor an intermediate CA that can link to the Java default trusted root CA. The Java default trusted root CAs can be found in `jre/lib/security/cacerts`.

You likely want to configure the private key PEM file to be a wildcard certificate that enables access to the Amazon VPC domain in which your cluster instances reside. For example, if your cluster resides in us-east-1, you may choose to specify a common name in the certificate configuration that allows access to the cluster by specifying `CN=*.ec2.internal` in the certificate subject definition. For more information about Amazon EMR cluster configuration within Amazon VPC, see Select an Amazon VPC Subnet for the Cluster.

The following example demonstrates how to use OpenSSL to generate a self-signed X.509 certificate with a 1024-bit RSA private key that allows access to the issuer's Amazon EMR cluster instances in the US East (N. Virginia) region. This is identified by the `*.ec2.internal` domain name as the common name. Other optional subject items—such as country (C), state (S), Locale (L), etc.—are specified. Because a self-signed certificate is generated, the example then copies the `certificateChain.pem` file to the `trustedCertificates.pem` file. The `zip` command is then used to create the `my-certs.zip` file that contains the certificates.

Important
This example is a proof-of-concept demonstration only. Using self-signed certificates is not recommended and presents a potential security risk. For production systems, use a trusted certification authority (CA) to issue certificates.

```
$ openssl req -x509 -newkey rsa:1024 -keyout privateKey.pem -out
 certificateChain.pem -days 365 -nodes -subj '/C=US/S=Washington/L=Seattle/
O=MyOrg/OU=MyDept/CN=*.ec2.internal'
$ cp certificateChain.pem trustedCertificates.pem
$ zip -r -X my-certs.zip certificateChain.pem privateKey.pem
```

Using a Custom Certificate Provider

Custom certificate providers must implement the TLSArtifacts provider interface.

Specifying Amazon EMR Encryption Options Using a Security Configuration

Using a security configuration to specify cluster encryption settings is a two-step process. First, you create a security configuration, which you can use for any number of clusters. Then you specify the security configuration to use when you create a cluster. Before you create a security configuration, decide on the key and certificate management systems you want to use and create the keys and certificates. For more information, see Providing Keys for At-Rest Data Encryption with Amazon EMR (p. 150) and Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption (p. 152).

Creating a Security Configuration

When you create a security configuration, you specify two sets of encryption options: at-rest data encryption and in-transit data encryption. At-rest data encryption options include both Amazon S3 with EMRFS and local-disk encryption. In-transit encryption options enable the open-source encryption features for certain applications that support transport layer security (TLS). At-rest options and in-transit options can be enabled together or separately. You can use the AWS Management Console, the AWS CLI, or the AWS SDKs to create a security configuration.

Creating a Security Configuration Using the Console

To create a security configuration:

1. Sign in to the AWS Management Console and open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. In the navigation pane, choose **Security Configurations**, **Create security configuration**.
3. Type a **Name** for the security configuration.
4. Choose **At rest encryption** to encrypt data stored within the file system. This also enables Hadoop Distributed File System (HDFS) block-transfer encryption and RPC encryption, which need no further configuration.
5. Under **S3 data encryption**, choose a value for **Encryption mode**, which determines how Amazon EMR encrypts Amazon S3 data with EMRFS.

   What you do next depends on the encryption mode you chose:

   - **SSE-S3**

     Specifies Server-side encryption with Amazon S3-managed encryption keys. You don't need to do anything more because Amazon S3 handles keys for you.
   - **SSE-KMS** or **CSE-KMS**

     Specifies server-side encryption with AWS KMS-managed keys (SSE-KMS) or client-side encryption with AWS KMS-managed keys (CSE-KMS). For **AWS KMS Key**, select a key. The key must exist in the same region as your Amazon EMR cluster. For key requirements, see Using AWS KMS Customer Master Keys (CMKs) for Encryption (p. 150).
   - **CSE-Custom**

     Specifies client-side encryption using a custom client-side master key (CSE-Custom). In the **S3 object** box, enter the location in Amazon S3, or the Amazon S3 ARN, of your custom key-provider JAR file. Then, in the **Key provider class** field, enter the full class name of a class declared in your application that implements the EncryptionMaterialsProvider interface.
6. Under **Local disk encryption**, choose a value for **Key provider type**. Amazon EMR uses this key for Linux Unified Key System (LUKS) encryption for the local volumes (except boot volumes) attached to your cluster nodes.

   - **AWS KMS**

     Select this option to specify an AWS KMS customer master key (CMK). For **AWS KMS Key**, select a key. The key must exist in the same region as your Amazon EMR cluster. For more information about key requirements, see Using AWS KMS Customer Master Keys (CMKs) for Encryption (p. 150).
   - **Custom**

     Select this option to specify a custom key provider. For **S3 object**, enter the location in Amazon S3, or the Amazon S3 ARN, of your custom key-provider JAR file. For **Key provider class**, enter the full class name of a class declared in your application that implements the

EncryptionMaterialsProvider interface. The class name you provide here must be different from the class name provided for CSE-Custom.

7. Choose **In-transit encryption** to enable the open-source TLS encryption features for in-transit data. Choose a **Certificate provider type** according to the following guidelines:

   - **PEM**

     Select this option to use PEM files that you provide within a zip file. Two artifacts are required within the zip file: privateKey.pem and certificateChain.pem. A third file, trustedCertificates.pem, is optional. See Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption (p. 152) for details. Specify the location in Amazon S3, or the Amazon S3 ARN, of the zip file in the **S3 object** box.

   - **Custom**

     Select this option to specify a custom certificate provider and then, for **S3 object**, enter the location in Amazon S3, or the Amazon S3 ARN, of your custom certificate-provider JAR file. For **Key provider class**, enter the full class name of a class declared in your application that implements the TLSArtifactsProvider interface.

8. Click **Create**.

Creating a Security Configuration Using the AWS CLI

To create a security configuration with the AWS CLI, use the following command:

```
aws emr create-security-configuration --name "SecConfigName" --security-
configuration SecConfigDef
```

- **--name *SecConfigName*** specifies the name of the security configuration, which you will specify when you create a cluster.
- `--security-configuration '`*`SecConfigDef`*`'` specifies a JSON blob (examples below) or the path to a JSON file in Amazon S3 (such as *`file://./MySecConfig.json`*) that defines encryption parameters.

The sections that follow use sample scenarios to illustrate well-formed **--security-configuration** JSON for different configurations and key providers, as well as a reference for JSON parameters.

Example In-Transit Data Encryption Options

The example below illustrates the following scenario:

- In-transit data encryption is enabled and at-rest data encryption is disabled.
- A zip file with certificates in Amazon S3 is used as the key provider (see Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption (p. 152) for certificate requirements.

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : true,
  "EnableAtRestEncryption" : false,
  "InTransitEncryptionConfiguration" : {
   "TLSCertificateConfiguration" : {
```

```
    "CertificateProviderType" : "PEM",
    "S3Object" : "s3://MyConfigStore/artifacts/MyCerts.zip"
   }
  }
 }
}'
```

The example below illustrates the following scenario:

- In-transit data encryption is enabled and at-rest data encryption is disabled.
- A custom key provider is used (see Providing Certificates for In-Transit Data Encryption with Amazon EMR Encryption (p. 152) for certificate requirements).

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : true,
  "EnableAtRestEncryption" : false,
  "InTransitEncryptionConfiguration" : {
   "TLSCertificateConfiguration" : {
    "CertificateProviderType" : "Custom",
    "S3Object" : "s3://MyConfig/artifacts/MyCerts.jar",
    "CertificateProviderClass" : "com.mycompany.MyCertProvider"
   }
  }
  }
}'
```

Example At-rest Data Encryption Options

The example below illustrates the following scenario:

- In-transit data encryption is disabled and at-rest data encryption is enabled
- SSE-S3 is used for Amazon S3 encryption
- Local disk encryption uses AWS KMS as the key provider

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : false,
  "EnableAtRestEncryption" : true,
  "AtRestEncryptionConfiguration" : {
   "S3EncryptionConfiguration" : {
    "EncryptionMode" : "SSE-S3"
   },
   "LocalDiskEncryptionConfiguration" : {
    "EncryptionKeyProviderType" : "AwsKms",
    "AwsKmsKey" : "arn:aws:kms:us-
east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
   }
  }
  }
}'
```

The example below illustrates the following scenario:

- In-transit data encryption is enabled and references a zip file with PEM certificates in Amazon S3, using the ARN
- SSE-KMS is used for Amazon S3 encryption
- Local disk encryption uses AWS KMS as the key provider

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : true,
  "EnableAtRestEncryption" : true,
  "InTransitEncryptionConfiguration" : {
   "TLSCertificateConfiguration" : {
    "CertificateProviderType" : "PEM",
    "S3Object" : "arn:aws:s3:::MyConfigStore/artifacts/MyCerts.zip"
   }
  },
  "AtRestEncryptionConfiguration" : {
   "S3EncryptionConfiguration" : {
    "EncryptionMode" : "SSE-KMS",
    "AwsKmsKey" : "arn:aws:kms:us-
east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
   },
   "LocalDiskEncryptionConfiguration" : {
    "EncryptionKeyProviderType" : "AwsKms",
    "AwsKmsKey" : "arn:aws:kms:us-
east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
   }
  }
 }
}'
```

The example below illustrates the following scenario:

- In-transit data encryption is enabled and references a zip file with PEM certificates in Amazon S3
- CSE-KMS is used for Amazon S3 encryption
- Local disk encryption uses a custom key provider referenced by its ARN

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : true,
  "EnableAtRestEncryption" : true,
  "InTransitEncryptionConfiguration" : {
   "TLSCertificateConfiguration" : {
    "CertificateProviderType" : "PEM",
    "S3Object" : "s3://MyConfigStore/artifacts/MyCerts.zip"
   }
  },
  "AtRestEncryptionConfiguration" : {
   "S3EncryptionConfiguration" : {
    "EncryptionMode" : "CSE-KMS",
```

```
    "AwsKmsKey" : "arn:aws:kms:us-
east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
   },
   "LocalDiskEncryptionConfiguration" : {
    "EncryptionKeyProviderType" : "Custom",
    "S3Object" : "arn:aws:s3:::artifacts/MyKeyProvider.jar",
    "EncryptionKeyProviderClass" : "com.mycompany.MyKeyProvider.jar"
   }
  }
 }
}'
```

The example below illustrates the following scenario:

- In-transit data encryption is enabled with a custom key provider
- CSE-Custom is used for Amazon S3 data
- Local disk encryption uses a custom key provider

```
aws emr create-security-configuration --name "MySecConfig" --security-
configuration '{
 "EncryptionConfiguration": {
  "EnableInTransitEncryption" : "true",
  "EnableAtRestEncryption" : "true",
  "InTransitEncryptionConfiguration" : {
   "TLSCertificateConfiguration" : {
    "CertificateProviderType" : "Custom",
    "S3Object" : "s3://MyConfig/artifacts/MyCerts.jar",
    "CertificateProviderClass" : "com.mycompany.MyCertProvider"
   }
  },
  "AtRestEncryptionConfiguration" : {
   "S3EncryptionConfiguration" : {
    "EncryptionMode" : "CSE-Custom",
    "S3Object" : "s3://MyConfig/artifacts/MyCerts.jar",
    "EncryptionKeyProviderClass" : "com.mycompany.MyKeyProvider"
    },
   "LocalDiskEncryptionConfiguration" : {
    "EncryptionKeyProviderType" : "Custom",
    "S3Object" : "s3://MyConfig/artifacts/MyCerts.jar",
    "EncryptionKeyProviderClass" : "com.mycompany.MyKeyProvider"
   }
  }
 }
}'
```

AWS CLI Security Configuration JSON Reference

The following table lists the JSON parameters for encryption settings and provides a description of acceptable values for each parameter.

Parameter	Description
`"EnableInTransitEncryption" : true|false`	Specify `true` to enable in-transit encryption and `false` to disable it. If omitted, `false` is assumed, and in-transit encryption is disabled.

Parameter	Description
`"EnableAtRestEncryption" : true|false`	Specify `true` to enable at-rest encryption and `false` to disable it. If omitted, `false` is assumed and at-rest encryption is disabled.
In-transit encryption parameters	
`"InTransitEncryptionConfiguration" :`	Specifies a collection of values used to configure in-transit encryption when `EnableInTransitEncryption` is `true`.
`"CertificateProviderType" : "PEM"|"Custom"`	Specifies whether to use `PEM` certificates referenced with a zipped file, or a `Custom` certificate provider. If `PEM` is specified, `S3Object` must be a reference to the location in Amazon S3 of a zip file containing the certificates. If Custom is specified, `S3Object` must be a reference to the location in Amazon S3 of a JAR file, followed by a `CertificateProviderClass` entry.
`"S3Object" : "`*`ZipLocation`*`"|"`*`JarLocation`*`"`	Provides the location in Amazon S3 to a zip file when `PEM` is specified, or to a JAR file when `Custom` is specified. The format can be a path (for example, `s3://MyConfig/articfacts/CertFiles.zip`) or an ARN (for example, `arn:aws:s3:::Code/MyCertProvider.jar`). If a zip file is specified, it must contain files named exactly `privateKey.pem` and `certificateChain.pem`. A file named `trustedCertificates.pem` is optional.
`"CertificateProviderClass" : "`*`MyClassID`*`"`	Required only if `Custom` is specified for `CertificateProviderType`. *`MyClassID`* specifies a full class name declared in the JAR file, which implements the TLSArtifactsProvider interface. For example, `com.mycompany.MyCertProvider`.
At-rest encryption parameters	
`"AtRestEncryptionConfiguration" :`	Specifies a collection of values for at-rest encryption when `EnableAtRestEncryption` is `true`, including Amazon S3 encryption and local disk encryption.
Amazon S3 encryption parameters	
`"S3EncryptionConfiguration" :`	Specifies a collection of values used for Amazon S3 encryption with the EMR File System (EMRFS).

Parameter	Description
"EncryptionMode" : "SSE-S3" | "SSE-KMS" | "CSE-KMS" | "CSE-Custom"	Specifies the type of Amazon S3 encryption to use. If SSE-S3 is specified, no further S3 encryption values are required. If either SSE-KMS or CSE-KMS is specified, an AWS KMS customer master key (CMK) ARN must be specified as the AwsKmsKey value. If CSE-Custom is specified, S3Object and EncryptionKeyProviderClass values must be specified.
"AwsKmsKey" : "*MyKeyARN*"	Required only when either SSE-KMS or CSE-KMS is specified for EncryptionMode. *MyKeyARN* must be a fully specified ARN to a key (for example, arn:aws:kms:us-east-1:123456789012:key/12345678-1234-1234-1234-1234
"S3Object" : "*JarLocation*"	Required only when CSE-Custom is specified for CertificateProviderType. *JarLocation* provides the location in Amazon S3 to a JAR file. The format can be a path (for example, s3://MyConfig/articfacts/MyKeyProvider.jar) or an ARN (for example, arn:aws:s3:::Code/MyKeyProvider.jar).
"EncryptionKeyProviderClass" : "*MyS3KeyClassID*"	Required only when CSE-Custom is specified for EncryptionMode. *MyS3KeyClassID* specifies a full class name of a class declared in the application that implements the EncryptionMaterialsProvider interface; for example, *com.mycompany.MyS3KeyProvider*.
Local disk encryption parameters	
"LocalDiskEncryptionKeyProvider"	Specifies the key provider and corresponding values to be used for local disk encryption.
"Type" : "AwsKms" | "Custom"	Specifies the key provider. If AwsKms is specified, an AWS KMS CMK ARN must be specified as the AwsKmsKey value. If **Custom** is specified, S3Object and EncryptionKeyProviderClass values must be specified.
"AwsKmsKey : "*MyKeyARN*"	Required only when AwsKms is specified for Type. *MyKeyARN* must be a fully specified ARN to a key (for example, arn:aws:kms:us-east-1:123456789012:key/12345678-1234-1234-1234-4567
"S3Object" : "*JarLocation*"	Required only when CSE-Custom is specified for CertificateProviderType. *JarLocation* provides the location in Amazon S3 to a JAR file. The format can be a path (for example, s3://MyConfig/articfacts/MyKeyProvider.jar) or an ARN (for example, arn:aws:s3:::Code/MyKeyProvider.jar).

Parameter	Description
`"EncryptionKeyProviderClass" : "MyLocalDiskKeyClassID"`	Required only when `Custom` is specified for `Type`. *`MyLocalDiskKeyClassID`* specifies a full class name of a class declared in the application that implements the EncryptionMaterialsProvider interface; for example, *`com.mycompany.MyLocalDiskKeyProvider`*.

Using a Security Configuration To Specify Cluster Encryption Settings

You can specify encryption settings when you create a cluster by specifying the security configuration. You can use the AWS Management Console or the AWS CLI.

Specifying a Security Configuration Using the Console

When using the AWS console to create an Amazon EMR cluster, you choose the security configuration during **Step 4: Security** of the advanced options creation process.

1. Sign in to the AWS Management Console and open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose Create cluster, **Go to advanced options**.
3. On the**Step 1: Software and Steps** screen, from the **Release** list, choose **emr-4.8.0** or a more recent release. Choose the settings you want and choose Next.
4. On the **Step 2: Hardware** screen, choose the settings you want and choose Next. Do the same for **Step 3: General Cluster Settings**.
5. On the **Step 4: Security** screen, under **Encryption Options**, choose a value for **Security configuration**.
6. Configure other security options as desired and choose Create cluster.

Specifying a Security Configuration Using the CLI

When you use `aws emr create-cluster`, you can optionally apply a security configuration using `--security-configuration` *`MySecConfig`*, where *`MySecConfig`* is the name of the security configuration, as shown in the following example. The `--release-label` specified must be 4.8.0 or later and the `--instance-type` can be any available.

```
aws emr create-cluster --instance-type m3.xlarge --release-label emr-5.0.0 --
security-configuration mySecConfig
```

Specifying Amazon S3 Encryption with EMRFS Using a Cluster Configuration

When you create a cluster, you can specify Amazon S3 server-side encryption (SSE) or client-side encryption (CSE) using the `emrfs-site` classification. Amazon S3 SSE and CSE are mutually exclusive; you can choose either but not both. For more information about Amazon S3 encryption options, see Amazon S3 Server-Side Encryption (p. 148). Beginning with Amazon EMR release 4.8.0, you can use security configurations to apply encryption settings more easily and with more options.

Important
Although you can still use cluster configurations to apply encryption with current versions of Amazon EMR, it is not recommended. If you configure Amazon S3 encryption in the cluster configuration and in a security configuration, the security configuration overrides the cluster configuration.

For information about how to create security configurations, see Amazon EMR Data Encryption with Security Configurations.

Specifying Amazon S3 Server-Side Encryption

Amazon EMR supports server-side encryption with Amazon S3-provided keys (SSE-S3) and with AWS KMS-managed encryption keys (SSE-KMS). Amazon EMR does not support the Amazon S3 option to use SSE with customer-provided encryption keys (SSE-C). For more information about these options, see At-rest Encryption for Amazon S3 with EMRFS (p. 148).

Creating a Cluster with Amazon S3 SSE-S3 Enabled

To configure SSE-S3 as part of a cluster configuration, you can use the AWS Management Console or the AWS CLI. You can also use the `configure-hadoop` bootstrap action to set `fs.s3.enableServerSideEncryption` to `true`.

Note
The following AWS Management Console procedure is not available beginning with Amazon EMR version 4.8.0. Use a security configuration to specify encryption options. For more information, see Specifying a Security Configuration Using the Console (p. 161).

To create a cluster with SSE-S3 enabled using the console

1. Choose Create Cluster.
2. Navigate to the **File System Configuration** section.
3. To use **Server-side encryption**, choose **Enabled**.
4. Choose **Create cluster**.

To create a cluster with SSE-S3 enabled using the AWS CLI

- Type the following command:

```
aws emr create-cluster --release-label emr-5.2.1 \
--instance-count 3 --instance-type m1.large --emrfs Encryption=ServerSide
```

Creating a Cluster with Amazon S3 SSE-KMS Enabled

To configure SSE-KMS as part of a cluster configuration, you must use the AWS CLI or the AWS SDKs. There is no SSE-KMS configuration for Amazon EMR using the AWS Management Console. You enable SSE-KMS much the same as you do for SSE-S3, but you also provide an AWS KMS CMK ID or ARN (Amazon Resource Name) using the `fs.s3.serverSideEncryption.kms.keyId` setting in the `emrfs-site` configuration classification.

To create a cluster with SSE-KMS enabled using the AWS CLI

- Type the following AWS CLI command to create a cluster with SSE-KMS, where keyID is an AWS KMS customer master key (CMK):

```
aws emr create-cluster --release-label emr-4.5.0 --instance-count 3 \
--instance-type m1.xlarge --use-default-roles \
```

```
--emrfs
 Encryption=ServerSide,Args=[fs.s3.serverSideEncryption.kms.keyId=keyId]
```

--OR--

Type the following AWS CLI command using the configuration API and providing a configuration JSON file with contents as shown (`myConfig.json` in the example):

```
aws emr create-cluster --release-label emr-4.5.0 --instance-count 3 \
--instance-type m1.xlarge --applications Name=Hadoop \
--configurations file://./myConfig.json --use-default-roles
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

myConfig.json

```
[
  {
    "Classification":"emrfs-site",
    "Properties": {
       "fs.s3.enableServerSideEncryption": "true",

 "fs.s3.serverSideEncryption.kms.keyId":"a4567b8-9900-12ab-1234-123a45678901"
    }
  }
]
```

`emrfs-site.xml` Properties for SSE-S3 and SSE-KMS

Property	Default value	Description
`fs.s3.enableServerSideEncryption`	**`false`**	When set to **`true`**, objects stored in Amazon S3 are encrypted using server-side encryption. If no key is specified, SSE-S3 is used.
`fs.s3.serverSideEncryption.kms.keyId`	**`n/a`**	Specifies an AWS KMS key ID or ARN. If a key is specified, SSE-KMS is used.

Specifying Amazon S3 Client-Side Encryption

Amazon EMR supports Amazon S3 client-side encryption (CSE) using an AWS KMS-managed CMK or using a custom client-side master key you provide in a Java class implementation. For more information about Amazon S3 CSE, see Protecting Data Using Client-Side Encryption in the *Amazon Simple Storage Service Developer Guide*.

Enabling Amazon S3 Client-Side Encryption in the Console

To configure client-side encryption using the console

1. Choose Create Cluster.
2. Fill in the fields as appropriate for **Cluster Configuration** and **Tags**.

3. For the **Software Configuration** field, choose **AMI 3.6.0** or later.
4. In the **File System Configuration** section, select one of the following client-side encryption types for the **Encryption** field: **S3 client-side encryption with AWS Key Management Service (KMS)** or **S3 client-side encryption with custom encryption materials provider**.

   a. If you chose **S3 client-side encryption with AWS Key Management Service (KMS)**, select the master key alias from the list of master keys that you have previously configured. Alternately, you can choose **Enter a Key ARN** and enter the ARN of an AWS KMS master key that belongs to a different account, provided that you have permissions to use that key. If you have assigned an instance profile to your EMR cluster, make sure that the role in that profile has permissions to use the key.

   b. If you chose **S3 client-side encryption with custom encryption materials provider**, provide the full class name and Amazon S3 location of your EncryptionMaterialsProvider class. Amazon EMR automatically downloads your provider to each node in your cluster when it is created.
5. Fill in the fields as appropriate for **Hardware Configuration**, **Security and Access**, **Bootstrap Actions**, and **Steps**.
6. Choose **Create cluster**.

Selecting a Master Key Stored in AWS KMS Using an SDK or CLI

When you enable Amazon S3 client-side encryption and specify keys stored in AWS KMS, you provide the KeyId value, key alias, or ARN of the key that Amazon EMR will use to encrypt objects written to Amazon S3. For decryption, EMRFS tries to access whichever key encrypted the object. You create the key using the IAM console, AWS CLI, or the AWS SDKs.

If you have assigned an instance profile to your EMR cluster, make sure that the role in that profile has permission to use the key. AWS KMS charges apply for API calls during each encryption or decryption activity and for storing your key. For more information, see the AWS KMS pricing page.

To use an AWS KMS master key for Amazon S3 encryption, provide the master key by reference using any of three possible identifiers:

- KeyId (a 32-character GUID)
- Alias mapped to the KeyId value (you must include the `alias/` prefix in this value)
- Full ARN of the key, which includes the region, account ID, and KeyId value

`MyKMSKeyId` in the example below can be any of the three values:

```
aws emr create-cluster --release-label emr-5.2.1 \
--emrfs Encryption=ClientSide,ProviderType=KMS,KMSKeyId=MyKMSKeyId
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

Note
You must use the ARN of the AWS KMS master key to use a key owned by an account different than the one you are using to configure Amazon EMR.

Configuring Amazon S3 Client-side Encryption Using a Custom Provider

To use the AWS CLI, pass the `Encryption`, `ProviderType`, `CustomProviderClass`, and `CustomProviderLocation` arguments to the `emrfs` option.

```
aws emr create-cluster --instance-type m3.xlarge --release-label emr-5.2.1 \
```

```
--emrfs
 Encryption=ClientSide,ProviderType=Custom,CustomProviderLocation=s3://
mybucket/myfolder/provider.jar,CustomProviderClass=classname
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

Setting `Encryption` to `ClientSide` enables client-side encryption, `CustomProviderClass` is the name of your EncryptionMaterialsProvider object, and `CustomProviderLocation` is the local or Amazon S3 location from which Amazon EMR copies `CustomProviderClass` to each node in the cluster and places it in the classpath.

Custom EncryptionMaterialsProvider with Arguments

You may need to pass arguments directly to the provider, so you can use a configuration to supply arguments using `emrfs-site.xml`. Here is the configuration:

```
[
    {
      "Classification": "emrfs-site",
      "Properties": {
        "myProvider.arg1":"value1",
  "myProvider.arg2":"value2"
      }
    }
 ]
```

Then, use the configuration with the CLI:

```
aws emr create-cluster --release-label emr-5.2.1 \
--instance-type m3.xlarge --instance-count 2 --configurations file://./
myConfig.json --emrfs Encryption=ClientSide,CustomProviderLocation=s3://
mybucket/myfolder/myprovider.jar,CustomProviderClass=classname
```

To use an SDK, you can set the property `fs.s3.cse.encryptionMaterialsProvider.uri` to download the custom EncryptionMaterialsProvider class that you store in Amazon S3 to each node in your cluster. You configure this in `emrfs-site.xml` file along with CSE enabled and the proper location of the custom provider.

For example, in the AWS SDK for Java using RunJobFlowRequest, your code might look like the following:

```
<snip>
  Map<String,String> emrfsProperties = new HashMap<String,String>();
      emrfsProperties.put("fs.s3.cse.encryptionMaterialsProvider.uri","s3://
mybucket/MyCustomEncryptionMaterialsProvider.jar");
      emrfsProperties.put("fs.s3.cse.enabled","true");
      emrfsProperties.put("fs.s3.consistent","true");

 emrfsProperties.put("fs.s3.cse.encryptionMaterialsProvider","full.class.name.of.Encryption

  Configuration myEmrfsConfig = new Configuration()
      .withClassification("emrfs-site")
      .withProperties(emrfsProperties);

  RunJobFlowRequest request = new RunJobFlowRequest()
```

```
   .withName("Custom EncryptionMaterialsProvider")
   .withReleaseLabel("emr-5.2.1")
   .withApplications(myApp)
   .withConfigurations(myEmrfsConfig)
   .withServiceRole("EMR_DefaultRole")
   .withJobFlowRole("EMR_EC2_DefaultRole")
   .withLogUri("s3://myLogUri/")
   .withInstances(new JobFlowInstancesConfig()
    .withEc2KeyName("myEc2Key")
    .withInstanceCount(2)
    .withKeepJobFlowAliveWhenNoSteps(true)
    .withMasterInstanceType("m3.xlarge")
    .withSlaveInstanceType("m3.xlarge")
   );

  RunJobFlowResult result = emr.runJobFlow(request);
</snip>
```

For more information about a list of configuration key values to use to configure `emrfs-site.xml`, see `emrfs-site.xml` Properties for SSE-S3 and SSE-KMS (p. 163).

Reference Implementation of Amazon S3 EncryptionMaterialsProvider

When fetching the encryption materials from the EncryptionMaterialsProvider class to perform encryption, EMRFS optionally populates the materialsDescription argument with two fields: the Amazon S3 URI for the object and the JobFlowId of the cluster, which can be used by the EncryptionMaterialsProvider class to return encryption materials selectively. You can enable this behavior by setting `fs.s3.cse.materialsDescription.enabled` to `true` in `emrfs-site.xml`. For example, the provider may return different keys for different Amazon S3 URI prefixes. Note that it is the description of the returned encryption materials that is eventually stored with the Amazon S3 object rather than the materialsDescription value that is generated by EMRFS and passed to the provider. While decrypting an Amazon S3 object, the encryption materials description is passed to the EncryptionMaterialsProvider class, so that it can, again, selectively return the matching key to decrypt the object.

The following EncryptionMaterialsProvider reference implementation is provided below. Another custom provider, EMRFSRSAEncryptionMaterialsProvider, is available from GitHub.

```
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.EncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.KMSEncryptionMaterials;
import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;

import java.util.Map;

/**
 * Provides KMSEncryptionMaterials according to Configuration
 */
public class MyEncryptionMaterialsProviders implements
 EncryptionMaterialsProvider, Configurable{
  private Configuration conf;
  private String kmsKeyId;
  private EncryptionMaterials encryptionMaterials;

  private void init() {
    this.kmsKeyId = conf.get("my.kms.key.id");
    this.encryptionMaterials = new KMSEncryptionMaterials(kmsKeyId);
```

```
  }

  @Override
  public void setConf(Configuration conf) {
    this.conf = conf;
    init();
  }

  @Override
  public Configuration getConf() {
    return this.conf;
  }

  @Override
  public void refresh() {

  }

  @Override
  public EncryptionMaterials getEncryptionMaterials(Map<String, String>
 materialsDescription) {
    return this.encryptionMaterials;
  }

  @Override
  public EncryptionMaterials getEncryptionMaterials() {
    return this.encryptionMaterials;
  }
}
```

`emrfs-site.xml` Properties for Amazon S3 Client-side Encryption

Property	Default value	Description
`fs.s3.cse.enabled`	**`false`**	When set to **`true`**, objects stored in Amazon S3 are encrypted using client-side encryption.
`fs.s3.cse.encryptionMaterialsProvider.uri`	**N/A**	The Amazon S3 URI where the JAR with the EncryptionMaterialsProvider is located. When you provide this URI, Amazon EMR automatically downloads the JAR to all nodes in the cluster.
`fs.s3.cse.encryptionMaterialsProvider`	**N/A**	The EncryptionMaterialsProvider class path used with client-side encryption. **Note** For AWS KMS, use `com.amazon.ws.emr.hadoop.fs.cse.KMSEn`
`fs.s3.cse.materialsDescription.enabled`	**`false`**	Enabling populates the materialsDescription of encrypted objects with the Amazon S3 URI for the object and the JobFlowId.

Property	Default value	Description
`fs.s3.cse.kms.keyId`	**`N/A`**	The value of the KeyId field for the AWS KMS encryption key that you are using with EMRFS encryption. **Note** This property also accepts the ARN and key alias associated with the key.
`fs.s3.cse.cryptoStorageMode`	**`ObjectMetadata`**	The Amazon S3 storage mode. By default, the description of the encryption information is stored in the object metadata. You can also store the description in an instruction file. Valid values are ObjectMetadata and InstructionFile. For more information, see Client-Side Data Encryption with the AWS SDK for Java and Amazon S3.

Transparent Encryption in HDFS on Amazon EMR

Note
This feature is only available in Amazon EMR 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 in Amazon EMR

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 configure the encryption zone.

- Create a cluster with the following command:

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge
 --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the 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:

```
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\""] \
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
For Windows, replace the above Linux line continuation character (\) with the 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. 168). 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

Classification	Filename
hadoop-kms-site	`kms-site.xml`
hadoop-kms-acls	`kms-acls.xml`
hadoop-kms-env	`kms-env.sh`
hadoop-kms-log4j	`kms-log4j.properties`

To set Hadoop KMS ACLs using the CLI

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

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge
 --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.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:

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge
 --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.json`:

```
[
    {
      "Classification": "hadoop-kms-site",
      "Properties": {
        "hadoop.kms.cache.enable": "false"
      }
    }
  ]
```

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:

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge
 --instance-count 2 \
--applications Name=App1 Name=App2 --configurations https://
s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

`myConfig.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.

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 Amazon Kinesis stream in a Hive query, Pig script, or MapReduce application and then operate on that data.

Topics

EMR File System (EMRFS) (Optional)

The EMR File System (EMRFS) and the Hadoop Distributed File System (HDFS) are both installed as components in the release. EMRFS is an implementation of HDFS which allows clusters to store data on Amazon S3. You can enable Amazon S3 server-side and client-side encryption as well as consistent view for EMRFS using the AWS Management Console, AWS CLI, or you can use a bootstrap action (with CLI or SDK) to configure additional settings for EMRFS.

Enabling Amazon S3 server-side encryption allows you to encrypt objects written to Amazon S3 by EMRFS. EMRFS support for Amazon S3 client-side encryption allows your cluster to work with S3 objects that were previously encrypted using an Amazon S3 encryption client. Consistent view provides consistency checking for list and read-after-write (for new put requests) for objects in Amazon S3. Enabling consistent view requires you to store EMRFS metadata in Amazon DynamoDB. If the metadata is not present, it is created for you.

Topics

Consistent View

EMRFS consistent view monitors Amazon S3 list consistency for objects written by or synced with EMRFS, delete consistency for objects deleted by EMRFS, and read-after-write consistency for new objects written by EMRFS.

Amazon S3 is designed for eventual consistency. For instance, buckets in all regions provide read-after-write consistency for put requests of new objects and eventual consistency for overwrite of put and delete requests. Therefore, if you are listing objects in an Amazon S3 bucket quickly after putting new objects, Amazon S3 does not provide a guarantee to return a consistent listing and it may be incomplete. This is more common in quick sequential MapReduce jobs which use Amazon S3 as a data store.

EMRFS includes a command line utility on the master node, `emrfs`, which allows administrator to perform operations on metadata such as import, delete, and sync. For more information about the EMRFS CLI, see the section called “EMRFS CLI Reference” (p. 183).

For a given path, EMRFS returns the set of objects listed in the EMRFS metadata and those returned directly by Amazon S3. Because Amazon S3 is still the “source of truth” for the objects in a path, EMRFS ensures that everything in a specified Amazon S3 path is being processed regardless of whether it is tracked in the metadata. However, EMRFS consistent view only ensures that the objects in the folders which you are tracking are being checked for consistency. The following topics give further details about how to enable and use consistent view.

> **Note**
> If you directly delete objects from Amazon S3 that are being tracked in the EMRFS metadata, EMRFS sees an entry for that object in the metadata but not the object in a Amazon S3 list or get request. Therefore, EMRFS treats the object as inconsistent and throws an exception after it has exhausted retries. You should use EMRFS to delete objects in Amazon S3 that are being tracked in the consistent view, purge the entries in the metadata for objects directly deleted in Amazon S3, or sync the consistent view with Amazon S3 immediately after you delete objects directly from Amazon S3.

To read an article about EMRFS consistency, see the Ensuring Consistency When Using Amazon S3 and Amazon Elastic MapReduce for ETL Workflows post on the AWS Big Data blog.

Topics

How to Enable Consistent View

You can enable Amazon S3 server-side encryption or consistent view for EMRFS using the AWS Management Console, AWS CLI, or the `emrfs-site` configuration classification..

To configure consistent view using the console

1. Choose Create Cluster.
2. Navigate to the **File System Configuration** section.
3. To enable **Consistent view**, choose **Enabled**.
4. For **EMRFS Metadata store**, type the name of your metadata store. The default value is **`EmrFSMetadata`**. If the EmrFSMetadata table does not exist, it is created for you in DynamoDB.

   > **Note**
   > Amazon EMR does not automatically remove the EMRFS metadata from DynamoDB when the cluster is terminated.

5. For **Number of retries**, type an integer value. This value represents the number of times EMRFS retries calling Amazon S3 if an inconsistency is detected. The default value is **5**.
6. For **Retry period (in seconds)**, type an integer value. This value represents the amount of time that lapses before EMRFS retries calling Amazon S3. The default value is **10**.

   Note
   Subsequent retries use an exponential backoff.

To launch a cluster with consistent view enabled using the AWS CLI

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

Type the following command to launch an Amazon EMR cluster with consistent view enabled.

```
aws emr create-cluster --instance-type m1.large --instance-count 3 --emrfs
 Consistent=true \
--release-label emr-5.2.1 --ec2-attributes KeyName=myKey
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

To check if consistent view is enabled using the AWS Management Console

To check whether consistent view is enabled in the console, navigate to the **Cluster List** and select your cluster name to view **Cluster Details**. The "EMRFS consistent view" field has a value of `Enabled` or `Disabled`.

To check if consistent view is enabled by examining the `emrfs-site.xml` file

You can check if consistency is enabled by inspecting the `emrfs-site.xml` configuration file on the master node of the cluster. If the Boolean value for `fs.s3.consistent` is set to `true` then consistent view is enabled for file system operations involving Amazon S3.

Objects Tracked By EMRFS

EMRFS creates a consistent view of objects in Amazon S3 by adding information about those objects to the EMRFS metadata. EMRFS adds these listings to its metadata when:

- An object written by EMRFS during the course of an Amazon EMR job.
- An object is synced with or imported to EMRFS metadata by using the EMRFS CLI.

Objects read by EMRFS are not automatically added to the metadata. When a object is deleted by EMRFS, a listing still remains in the metadata with a deleted state until that listing is purged using the EMRFS CLI. To learn more about the CLI, see the section called “EMRFS CLI Reference” (p. 183). For more information about purging listings in the EMRFS metadata, see the section called “EMRFS Metadata” (p. 177).

For every Amazon S3 operation, EMRFS checks the metadata for information about the set of objects in consistent view. If EMRFS finds that Amazon S3 is inconsistent during one of these operations, it will retry the operation according to parameters defined in `emrfs-site.xml`. After retries are exhausted, it will either throw a `ConsistencyException` or log the exception and continue the workflow. For more information about this retry logic, see the section called “Retry Logic” (p.). You can find `ConsistencyExceptions` in your logs, for example:

- listStatus: No s3 object for metadata item `/S3_bucket/dir/object`

- getFileStatus: Key `dir/file` is present in metadata but not s3

If you delete an object that is being tracked in the EMRFS consistent view directly from Amazon S3, EMRFS will treat that object as inconsistent because it will still be listed in the metadata as present in Amazon S3. If your metadata becomes out of sync with the objects it is tracking in Amazon S3, you can use the **sync** subcommand on the EMRFS CLI to reset the listings in the metadata to reflect what is currently in Amazon S3. To find if there is a discrepancy between the metadata and Amazon S3, you can use the **diff** subcommand on the EMRFS CLI to compare them. Finally, EMRFS only has a consistent view of the objects referenced in the metadata; there can be other objects in the same Amazon S3 path that are not being tracked. When EMRFS lists the objects in an Amazon S3 path, it will return the superset of the objects being tracked in the metadata and those in that Amazon S3 path.

Retry Logic

EMRFS will try to verify list consistency for objects tracked in its metadata for a specific number of retries. The default is 5. In the case where the number of retries is exceeded the originating job returns a failure unless `fs.s3.consistent.throwExceptionOnInconsistency` is set to `false`, where it will only log the objects tracked as inconsistent. EMRFS uses an exponential backoff retry policy by default but you can also set it to a fixed policy. Users may also want to retry for a certain period of time before proceeding with the rest of their job without throwing an exception. They can achieve this by setting `fs.s3.consistent.throwExceptionOnInconsistency` to `false`, `fs.s3.consistent.retryPolicyType` to `fixed`, and `fs.s3.consistent.retryPeriodSeconds` for the desired value. The following example will create a cluster with consistency enabled, which will log inconsistencies and set a fixed retry interval of 10 seconds:

Setting retry period to a fixed amount

```
aws emr create-cluster --release-label emr-5.2.1 \
--instance-type m3.xlarge --instance-count 1 \
--emrfs
 Consistent=true,Args=[fs.s3.consistent.throwExceptionOnInconsistency=false,
 fs.s3.consistent.retryPolicyType=fixed,fs.s3.consistent.retryPeriodSeconds=10]
 --ec2-attributes KeyName=myKey
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

For more information, see the section called “Configuring Consistent View” (p.).

EMRFS Metadata

Note
In order to use consistent view, your data is tracked in a DynamoDB database. Therefore, you will incur the cost of using that database while it exists.

Amazon EMR tracks consistency using a DynamoDB table to store object state. EMRFS consistent view creates and uses EMRFS metadata stored in a DynamoDB table to maintain a consistent view of Amazon S3 and this consistent view can be shared by multiple clusters. EMRFS creates and uses this metadata to track objects in Amazon S3 folders which have been synced with or created by EMRFS. The metadata is used to track all operations (read, write, update, and copy), and no actual content is stored in it. This metadata is used to validate whether the objects or metadata received from Amazon S3 matches what is expected. This confirmation gives EMRFS the ability to check list consistency and read-after-write consistency for new objects EMRFS writes to Amazon S3 or objects synced with EMRFS.

How to add entries to metadata

You can use the `sync` or `import` subcommands to add entries to metadata. `sync` will simply reflect the state of the Amazon S3 objects in a path while `import` is used strictly to add new entries to the metadata. For more information, see the section called “EMRFS CLI Reference” (p. 183).

How to check differences between metadata and objects in Amazon S3

To check for differences between the metadata and Amazon S3, use the `diff` subcommand of the EMRFS CLI. For more information, see the section called “EMRFS CLI Reference” (p. 183).

How to know if metadata operations are being throttled

EMRFS sets default throughput capacity limits on the metadata for its read and write operations at 400 and 100 units, respectively. Large numbers of objects or buckets may cause operations to exceed this capacity, at which point they will be throttled by DynamoDB. For example, an application may cause EMRFS to throw a `ProvisionedThroughputExceededException` if you are performing an operation that exceeds these capacity limits. Upon throttling the EMRFS CLI tool will attempt to retry writing to the DynamoDB table using exponential backoff until the operation finishes or when it reaches the maximum retry value for writing objects from EMR to Amazon S3.

You can also view Amazon CloudWatch metrics for your EMRFS metadata in the DynamoDB console where you can see the number of throttled read and/or write requests. If you do have a non-zero value for throttled requests, your application may potentially benefit from increasing allocated throughput capacity for read or write operations. You may also realize a performance benefit if you see that your operations are approaching the maximum allocated throughput capacity in reads or writes for an extended period of time.

Throughput characteristics for notable EMRFS operations

The default for read and write operations is 400 and 100 throughput capacity units, respectively. The following performance characteristics will give you an idea of what throughput is required for certain operations. These tests were performed using a single-node `m3.large` cluster. All operations were single threaded. Performance will differ greatly based on particular application characteristics and it may take experimentation to optimize file system operations.

Operation	Average read-per-second	Average write-per-second
create (object)	26.79	6.70
delete (object)	10.79	10.79
delete (directory containing 1000 objects)	21.79	338.40
getFileStatus (object)	34.70	0
getFileStatus (directory)	19.96	0
listStatus (directory containing 1 object)	43.31	0
listStatus (directory containing 10 objects)	44.34	0
listStatus (directory containing 100 objects)	84.44	0
listStatus (directory containing 1,000 objects)	308.81	0
listStatus (directory containing 10,000 objects)	416.05	0

Operation	Average read-per-second	Average write-per-second
listStatus (directory containing 100,000 objects)	823.56	0
listStatus (directory containing 1M objects)	882.36	0
mkdir (continuous for 120 seconds)	24.18	4.03
mkdir	12.59	0
rename (object)	19.53	4.88
rename (directory containing 1000 objects)	23.22	339.34

To submit a step that purges old data from your metadata store

Users may wish to remove particular entries in the DynamoDB-based metadata. This can help reduce storage costs associated with the table. Users have the ability to manually or programmatically purge particular entries by using the EMRFS CLI `delete` subcommand. However, if you delete entries from the metadata, EMRFS no longer makes any checks for consistency.

Programmatically purging after the completion of a job can be done by submitting a final step to your cluster which executes a command on the EMRFS CLI. For instance, type the following command to submit a step to your cluster to delete all entries older than two days.

```
aws emr add-steps --cluster-id j-2AL4XXXXXX5T9 --steps
 Name="emrfsCLI",Jar="command-runner.jar",Args=["emrfs","delete","--
time","2","time-unit","days"]
{
    "StepIds": [
        "s-B12345678902"
    ]
}
```

Use the StepId value returned to check the logs for the result of the operation.

Configuring Consistency Notifications for CloudWatch and Amazon SQS

You can enable CloudWatch metrics and Amazon SQS messages in EMRFS for Amazon S3 eventual consistency issues.

CloudWatch

When CloudWatch metrics are enabled, a metric named **Inconsistency** is pushed each time a `FileSystem` API call fails due to Amazon S3 eventual consistency.

To view CloudWatch metrics for Amazon S3 eventual consistency issues

To view the **Inconsistency** metric in the CloudWatch console, select the EMRFS metrics and then select a **JobFlowId**/**Metric Name** pair. For example: `j-162XXXXXXM2CU ListStatus`, `j-162XXXXXXM2CU GetFileStatus`, and so on.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the **Dashboard**, in the **Metrics** section, choose **EMRFS**.
3. In the **Job Flow Metrics** pane, select one or more **JobFlowId**/**Metric Name** pairs. A graphical representation of the metrics appears in the window below.

Amazon SQS

When Amazon SQS notifications are enabled, an Amazon SQS queue with the name `EMRFS-Inconsistency-<jobFlowId>` is created when EMRFS is initialized. Amazon SQS messages are pushed into the queue when a `FileSystem` API call fails due to Amazon S3 eventual consistency. The message contains information such as JobFlowId, API, a list of inconsistent paths, a stack trace, and so on. Messages can be read using the Amazon SQS console or using the EMRFS `read-sqs` command.

To manage Amazon SQS messages for Amazon S3 eventual consistency issues

Amazon SQS messages for Amazon S3 eventual consistency issues can be read using the EMRFS CLI. To read messages from an EMRFS Amazon SQS queue, type the `read-sqs` command and specify an output location on the master node's local file system for the resulting output file.

You can also delete an EMRFS Amazon SQS queue using the `delete-sqs` command.

1. To read messages from an Amazon SQS queue, type the following command. Replace *queuename* with the name of the Amazon SQS queue that you configured and replace */path/filename* with the path to the output file:

   ```
   emrfs read-sqs -queue-name queuename -output-file /path/filename
   ```

   For example, to read and output Amazon SQS messages from the default queue, type:

   ```
   emrfs read-sqs -queue-name EMRFS-Inconsistency-j-162XXXXXXM2CU -output-file /path/filename
   ```

   Note
   You can also use the `-q` and `-o` shortcuts instead of `-queue-name` and `-output-file` respectively.
2. To delete an Amazon SQS queue, type the following command:

   ```
   emrfs delete-sqs -queue-name queuename
   ```

   For example, to delete the default queue, type:

   ```
   emrfs delete-sqs -queue-name EMRFS-Inconsistency-j-162XXXXXXM2CU
   ```

   Note
   You can also use the `-q` shortcut instead of `-queue-name`.

Configuring Consistent View

You can configure additional settings for consistent view by providing them for the `/home/hadoop/conf/emrfs-site.xml` file by either using AWS CLI or a bootstrap action. For example, you can choose a different default DynamoDB throughput by supplying the following arguments to the CLI `--emrfs` option or bootstrap action:

Changing default metadata read and write values at cluster launch

```
aws emr create-cluster --release-label emr-5.2.1 --instance-type m3.xlarge \
--emrfs Consistent=true,Args=[fs.s3.consistent.metadata.read.capacity=600,\
fs.s3.consistent.metadata.write.capacity=300] --ec2-attributes KeyName=myKey
```

Alternatively, use the following configuration file and save it locally or in Amazon S3:

```
[
    {
      "Classification": "emrfs-site",
      "Properties": {
        "fs.s3.consistent.metadata.read.capacity": "600",
        "fs.s3.consistent.metadata.write.capacity": "300"
      }
    }
 ]
```

Use the configuration you created with the following syntax:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=Hive \
--instance-type m3.xlarge --instance-count 2 --configurations file://./
myConfig.json
```

Note
For Windows, replace the above Linux line continuation character (\) with the caret (^).

The following options can be set using configurations or AWS CLI `--emrfs` arguments. For information about those arguments, see the AWS Command Line Interface Reference.

`emrfs-site.xml` properties for consistent view

Property	Default value	Description
`fs.s3.consistent`	**`false`**	When set to **`true`**, this property configures EMRFS to use DynamoDB to provide consistency.
`fs.s3.consistent.retryPolicyType`	**`exponential`**	This property identifies the policy to use when retrying for consistency issues. Options include: exponential, fixed, or none.
`fs.s3.consistent.retryPeriodSeconds`	**`10`**	This property sets the length of time to wait between consistency retry attempts.
`fs.s3.consistent.retryCount`	**`5`**	This property sets the maximum number of retries when inconsistency is detected.
`fs.s3.consistent.throwExceptionOnInconsistency`	**`true`**	This property determines whether to throw or log a consistency exception. When set to **`true`**, a `ConsistencyException` is thrown.

Property	Default value	Description
fs.s3.consistent.metadata.autoCreate	**true**	When set to **true**, this property enables automatic creation of metadata tables.
fs.s3.consistent.metadata.tableName	**EmrFSMetadata**	This property specifies the name of the metadata table in DynamoDB.
fs.s3.consistent.metadata.read.capacity	**400**	This property specifies the DynamoDB read capacity to provision when the metadata table is created.
fs.s3.consistent.metadata.write.capacity	**100**	This property specifies the DynamoDB write capacity to provision when the metadata table is created.
fs.s3.consistent.fastList	**true**	When set to **true**, this property uses multiple threads to list a directory (when necessary). Consistency must be enabled in order to use this property.
fs.s3.consistent.fastList.prefetchMetadata	**false**	When set to **true**, this property enables metadata prefetching for directories containing more than 20,000 items.
fs.s3.consistent.notification.CloudWatch	**false**	When set to **true**, CloudWatch metrics are enabled for FileSystem API calls that fail due to Amazon S3 eventual consistency issues.
fs.s3.consistent.notification.SQS	**false**	When set to **true**, eventual consistency notifications are pushed to an Amazon SQS queue.
fs.s3.consistent.notification.SQS.queueName	**EMRFS-Inconsistency-<jobFlowId>**	Changing this property allows you to specify your own SQS queue name for messages regarding Amazon S3 eventual consistency issues.
fs.s3.consistent.notification.SQS.customMsg	**none**	This property allows you to specify custom information included in SQS messages regarding Amazon S3 eventual consistency issues. If a value is not specified for this property, the corresponding field in the message is empty.
fs.s3.consistent.dynamodb.endpoint	**none**	This property allows you to specify a custom DynamoDB endpoint for your consistent view metadata.

EMRFS CLI Reference

The EMRFS CLI is installed by default on all cluster master nodes created using AMI 3.2.1 or greater. You use the EMRFS CLI to manage the metadata, which tracks when objects have a consistent view.

> **Note**
> The **emrfs** command is only supported with VT100 terminal emulation. However, it may work with other terminal emulator modes.

The **emrfs** top-level command supports the following structure.

```
emrfs [[describe-metadata | set-metadata-capacity | delete-metadata | create-
metadata | \
list-metadata-stores | diff | delete | sync | import ]] [[options]]
 [[arguments]]
```

emrfs command

The **emrfs** command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-a` *AWS_ACCESS_KEY_ID* `| --access-key` *AWS_ACCESS_KEY_ID*	The AWS access key you use to write objects to Amazon S3 and to create or access a metadata store in DynamoDB. By default, *AWS_ACCESS_KEY_ID* is set to the access key used to create the cluster.	No
`-s` *AWS_SECRET_ACCESS_KEY* `| --secret-key` *AWS_SECRET_ACCESS_KEY*	The AWS secret key associated with the access key you use to write objects to Amazon S3 and to create or access a metadata store in DynamoDB. By default, *AWS_SECRET_ACCESS_KEY* is set to the secret key associated with the access key used to create the cluster.	No
`-v | --verbose`	Makes output verbose.	No
`-h | --help`	Displays the help message for the `emrfs` command with a usage statement.	No

describe-metadata sub-command

The **describe-metadata** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *METADATA_NAME* `| --metadata-name` *METADATA_NAME*	*METADATA_NAME* is the name of the DynamoDB metadata table. If the *METADATA_NAME* argument is not supplied, the default value is `EmrFSMetadata`.	No

describe-metadata example

The following example describes the default metadata table.

```
$ emrfs describe-metadata
EmrFSMetadata
  read-capacity: 400
```

```
  write-capacity: 100
  status: ACTIVE
  approximate-item-count (6 hour delay): 12
```

set-metadata-capacity sub-command

The **set-metadata-capacity** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *`METADATA_NAME`* `| --metadata-name` *`METADATA_NAME`*	*`METADATA_NAME`* is the name of the DynamoDB metadata table. If the *`METADATA_NAME`* argument is not supplied, the default value is `EmrFSMetadata`.	No
`-r` *`READ_CAPACITY`* `| --read-capacity` *`READ_CAPACITY`*	The requested read throughput capacity for the metadata table. If the *`READ_CAPACITY`* argument is not supplied, the default value is `400`.	No
`-w` *`WRITE_CAPACITY`* `| --write-capacity` *`WRITE_CAPACITY`*	The requested write throughput capacity for the metadata table. If the *`WRITE_CAPACITY`* argument is not supplied, the default value is `100`.	No

set-metadata-capacity example

The following example sets the read throughput capacity to `600` and the write capacity to `150` for a metadata table named `EmrMetadataAlt`.

```
$ emrfs set-metadata-capacity --metadata-name EmrMetadataAlt  --read-capacity
 600 --write-capacity 150
  read-capacity: 400
  write-capacity: 100
  status: UPDATING
  approximate-item-count (6 hour delay): 0
```

delete-metadata sub-command

The **delete-metadata** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *`METADATA_NAME`* `| --metadata-name` *`METADATA_NAME`*	*`METADATA_NAME`* is the name of the DynamoDB metadata table. If the *`METADATA_NAME`* argument is not supplied, the default value is `EmrFSMetadata`.	No

delete-metadata example

The following example deletes the default metadata table.

```
$ emrfs delete-metadata
```

create-metadata sub-command

The **create-metadata** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *METADATA_NAME* `| --metadata-name` *METADATA_NAME*	*METADATA_NAME* is the name of the DynamoDB metadata table. If the *METADATA_NAME* argument is not supplied, the default value is `EmrFSMetadata`.	No
`-r` *READ_CAPACITY* `| --read-capacity` *READ_CAPACITY*	The requested read throughput capacity for the metadata table. If the *READ_CAPACITY* argument is not supplied, the default value is `400`.	No
`-w` *WRITE_CAPACITY* `| --write-capacity` *WRITE_CAPACITY*	The requested write throughput capacity for the metadata table. If the *WRITE_CAPACITY* argument is not supplied, the default value is `100`.	No

create-metadata example

The following example creates a metadata table named `EmrFSMetadataAlt`.

```
$ emrfs create-metadata -m EmrFSMetadataAlt
Creating metadata: EmrFSMetadataAlt
EmrFSMetadataAlt
  read-capacity: 400
  write-capacity: 100
  status: ACTIVE
  approximate-item-count (6 hour delay): 0
```

list-metadata-stores sub-command

The **list-metadata-stores** sub-command has no [[options]].

list-metadata-stores example

The following example lists your metadata tables.

```
$ emrfs list--metadata-stores
  EmrFSMetadata
```

diff sub-command

The **diff** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *METADATA_NAME* `| --metadata-name` *METADATA_NAME*	*METADATA_NAME* is the name of the DynamoDB metadata table. If the *METADATA_NAME* argument is not supplied, the default value is `EmrFSMetadata`.	No
[s3://*s3Path*]	The path to the Amazon S3 bucket you are tracking for consistent view that you wish to compare to the metadata table. Buckets sync recursively.	Yes

diff example

The following example compares the default metadata table to an Amazon S3 bucket.

```
$ emrfs diff s3://elasticmapreduce/samples/cloudfront
BOTH | MANIFEST ONLY | S3 ONLY
```

```
DIR elasticmapreduce/samples/cloudfront
DIR elasticmapreduce/samples/cloudfront/code/
DIR elasticmapreduce/samples/cloudfront/input/
DIR elasticmapreduce/samples/cloudfront/logprocessor.jar
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-14.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-15.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-16.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-17.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-18.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-19.WxYz1234
DIR elasticmapreduce/samples/cloudfront/input/
XABCD12345678.2009-05-05-20.WxYz1234
DIR elasticmapreduce/samples/cloudfront/code/cloudfront-loganalyzer.tgz
```

delete sub-command

The **delete** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *`METADATA_NAME`* `| --metadata-name` *`METADATA_NAME`*	*`METADATA_NAME`* is the name of the DynamoDB metadata table. If the *`METADATA_NAME`* argument is not supplied, the default value is `EmrFSMetadata`.	No
[s3://*`s3Path`*]	The path to the Amazon S3 bucket you are tracking for consistent view. Buckets sync recursively.	Yes
`-t` *`TIME`* `| --time` *`TIME`*	The expiration time (interpreted using the time unit argument). All metadata entries older than the *`TIME`* argument are deleted for the specified bucket.	
`-u` *`UNIT`* `| --time-unit` *`UNIT`*	The measure used to interpret the time argument (nanoseconds, microseconds, milliseconds, seconds, minutes, hours, or days). If no argument is specified, the default value is `days`.	
`--read-consumption` *`READ_CONSUMPTION`*	The requested amount of available read throughput used for the **delete** operation. If the *`READ_CONSUMPTION`* argument is not specified, the default value is `400`.	No
`--write-consumption` *`WRITE_CONSUMPTION`*	The requested amount of available write throughput used for the **delete** operation. If the *`WRITE_CONSUMPTION`* argument is not specified, the default value is `100`.	No

delete example

The following example removes all objects in an Amazon S3 bucket from the tracking metadata for consistent view.

```
$ emrfs delete s3://elasticmapreduce/samples/cloudfront
```

```
entries deleted: 11
```

import sub-command

The **import** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *`METADATA_NAME`* `| --metadata-name` *`METADATA_NAME`*	*`METADATA_NAME`* is the name of the DynamoDB metadata table. If the *`METADATA_NAME`* argument is not supplied, the default value is `EmrFSMetadata`.	No
[s3://*`s3Path`*]	The path to the Amazon S3 bucket you are tracking for consistent view. Buckets sync recursively.	Yes
`--read-consumption` *`READ_CONSUMPTION`*	The requested amount of available read throughput used for the **delete** operation. If the *`READ_CONSUMPTION`* argument is not specified, the default value is `400`.	No
`--write-consumption` *`WRITE_CONSUMPTION`*	The requested amount of available write throughput used for the **delete** operation. If the *`WRITE_CONSUMPTION`* argument is not specified, the default value is `100`.	No

import example

The following example imports all objects in an Amazon S3 bucket with the tracking metadata for consistent view. All unknown keys are ignored.

```
$ emrfs import s3://elasticmapreduce/samples/cloudfront
```

sync sub-command

The **sync** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-m` *`METADATA_NAME`* `| --metadata-name` *`METADATA_NAME`*	*`METADATA_NAME`* is the name of the DynamoDB metadata table. If the *`METADATA_NAME`* argument is not supplied, the default value is `EmrFSMetadata`.	No
[s3://*`s3Path`*]	The path to the Amazon S3 bucket you are tracking for consistent view. Buckets sync recursively.	Yes
`--read-consumption` *`READ_CONSUMPTION`*	The requested amount of available read throughput used for the **delete** operation. If the *`READ_CONSUMPTION`* argument is not specified, the default value is `400`.	No
`--write-consumption` *`WRITE_CONSUMPTION`*	The requested amount of available write throughput used for the **delete** operation. If the *`WRITE_CONSUMPTION`* argument is not specified, the default value is `100`.	No

sync example

The following example imports all objects in an Amazon S3 bucket with the tracking metadata for consistent view. All unknown keys are deleted.

```
$ emrfs sync s3://elasticmapreduce/samples/cloudfront
Synching samples/cloudfront                                    0 added | 0
 updated | 0 removed | 0 unchanged
Synching samples/cloudfront/code/                              1 added | 0
 updated | 0 removed | 0 unchanged
Synching samples/cloudfront/                                   2 added | 0
 updated | 0 removed | 0 unchanged
Synching samples/cloudfront/input/                             9 added | 0
 updated | 0 removed | 0 unchanged
Done synching s3://elasticmapreduce/samples/cloudfront         9 added | 0
 updated | 1 removed | 0 unchanged
creating 3 folder key(s)
folders written: 3
```

read-sqs sub-command

The **read-sqs** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-q` *`QUEUE_NAME`* | `--queue-name` *`QUEUE_NAME`*	*`QUEUE_NAME`* is the name of the Amazon SQS queue configured in `emrfs-site.xml`. The default value is **`EMRFS-Inconsistency-<jobFlowId>`**.	Yes
`-o` *`OUTPUT_FILE`* | `--output-file` *`OUTPUT_FILE`*	*`OUTPUT_FILE`* is the path to the output file on the master node's local file system. Messages read from the queue are written to this file.	Yes

delete-sqs sub-command

The **delete-sqs** sub-command accepts the following [[options]] (with or without arguments).

Option	Description	Required
`-q` *`QUEUE_NAME`* | `--queue-name` *`QUEUE_NAME`*	*`QUEUE_NAME`* is the name of the Amazon SQS queue configured in `emrfs-site.xml`. The default value is **`EMRFS-Inconsistency-<jobFlowId>`**.	Yes

Submitting EMRFS CLI Commands as Steps

To add an Amazon S3 bucket to the tracking metadata for consistent view (AWS SDK for Python)

The following example shows how to use the `emrfs` utility on the master node by leveraging the AWS CLI or API and the `script-runner.jar` to run the `emrfs` command as a step. The example uses the AWS SDK for Python (Boto) to add a step to a cluster which adds objects in an Amazon S3 bucket to the default EMRFS metadata table.

```
from boto.emr import EmrConnection,connect_to_region,JarStep

emr=EmrConnection()
connect_to_region("us-east-1")
```

```
myStep = JarStep(name='Boto EMRFS Sync',
                 jar='s3://elasticmapreduce/libs/script-runner/script-
runner.jar',
                 action_on_failure="CONTINUE",
                 step_args=['/home/hadoop/bin/emrfs',
                            'sync',
                            's3://elasticmapreduce/samples/cloudfront'])


stepId = emr.add_jobflow_steps("j-2AL4XXXXXX5T9",
                           steps=[myStep]).stepids[0].value
```

You can use the stepId value returned to check the logs for the result of the operation.

Creating an AWSCredentialsProvider for EMRFS

You can create a custom credentials provider which implements both the AWSCredentialsProvider and the Hadoop Configurable classes for use with EMRFS when it makes calls to Amazon S3. Creating a custom credentials provider in this way is useful when an Amazon EMR user may need to provide different credentials depending on which Amazon S3 bucket is being accessed.

You must specify the full class name of the provider by setting `fs.s3.customAWSCredentialsProvider` in the `emrfs-site` configuration classification. You set this property at cluster creation time using the AWS CLI. For example, the following code sets `fs.s3.customAWSCredentialsProvider` to *`MyAWSCredentialsProvider`*.

Use the following configuration file and save it locally or in Amazon S3:

```
[
    {
      "Classification": "emrfs-site",
      "Properties": {
        "fs.s3.customAWSCredentialsProvider":"MyAWSCredentialsProvider"
      }
    }
 ]
```

Use the configuration you created with the following syntax:

```
aws emr create-cluster --release-label emr-5.2.1 --applications Name=Hive \
--instance-type m3.xlarge --instance-count 2 --configurations file://./
myConfig.json
```

An example implementation follows:

```
public class MyAWSCredentialsProvider implements AWSCredentialsProvider,
 Configurable {


    private Configuration conf;
    private String accessKey;
    private String secretKey;
```

```
    private void init() {
      accessKey = conf.get("my.accessKey");
      secretKey = conf.get("my.secretKey");
    }

    @Override
    public AWSCredentials getCredentials() {
      return new BasicAWSCredentials(accessKey, secretKey);
    }

    @Override
    public void refresh() {

    }

    @Override
    public void setConf(Configuration configuration) {
      this.conf = configuration;
      init();
    }

    @Override
    public Configuration getConf() {
      return this.conf;
    }
  }
```

An alternative implementation allows you to provide the bucket URI when creating an object. That follows this signature:

```
class MyCustomCredentialProvider implements AWSCredentialsProvider ,
 Configurable {
    public MyCustomCredentialProvider (Uri uri , Configuration configuration)
 {
    }
}
```

EMRFS Endpoint Resolution

EMRFS now resolves to the regional endpoints in which a cluster is running when it makes calls to Amazon S3. Cross-region calls between clusters and buckets not co-located in the same region may fail. This may affect scripts and other artifacts that use older non-regionalized URIs. For more information about region-specific Amazon S3 endpoints, see AWS Regions and Endpoints in the Amazon Web Services General Reference.

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. 87)

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 an 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. 203) or Write Percent Setting (p. 203) 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

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/.

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.

```
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 seconds = 14.56 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. 196).

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.

Hive type	DynamoDB type
string	string (S)
bigint or double	number (N)
binary	binary (B)
array	number set (NS), string set (SS), or binary set (BS)

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.

Hive Options	Description
dynamodb.throughput.read.percent	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

Hive Options	Description
	oversubscribe by setting it up to 1.5 if you believe there are unused input/output operations available.
dynamodb.throughput.write.percent	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.
dynamodb.endpoint	Specify the endpoint in case you have tables in different regions. For more information about the available DynamoDB endpoints, see Regions and Endpoints.
dynamodb.max.map.tasks	Specify the maximum number of map tasks when reading data from DynamoDB. This value must be equal to or greater than 1.
dynamodb.retry.duration	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.

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 '\t' 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>)
```

```
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 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. 34). 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.

```
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 seconds = 14.56 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. 194).

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. 194).

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, seeHive Options (p. 194).

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, go to Configure Hadoop (p. 34). There, you will find a "Task Configuration" section for each of the supported configurations.

```
{
    "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.

Amazon 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 Amazon 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:

Configuration Setting	Description
kinesis.checkpoint.metastore.table.name	DynamoDB table name where checkpoint information will be stored
kinesis.checkpoint.metastore.hash.key.name	Hash key name for the DynamoDB table
kinesis.checkpoint.metastore.hash.range.name	Range key name for the DynamoDB table

Configuration Setting	Description
kinesis.checkpoint.logical.name	A logical name for current processing
kinesis.checkpoint.iteration.no	Iteration number for processing associated with the logical name
kinesis.rerun.iteration.without.wait	Boolean value that indicates if a failed iteration can be rerun without waiting for timeout; the default is `false`

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*s/5`

For Write Capacity Units: `j*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

Topics

Note

The name of the command for S3DistCp in Amazon EMR 4.x or later is `s3-dist-cp`.

Apache DistCp is an open-source tool you can use to copy large amounts of data. DistCp 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, go to http://hadoop.apache.org/docs/stable/hadoop-distcp/DistCp.html.

S3DistCp is an extension of DistCp that is optimized to work with AWS, particularly Amazon S3. You use S3DistCp by adding it 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.

During a copy operation, S3DistCp stages a temporary copy of the output in HDFS on the cluster. There must be sufficient 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, therefore you must manually purge the temporary files. 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.

Option	Description	Required
`--src=LOCATION`	Location of the data to copy. This can be either an HDFS or Amazon S3 location. Example: `--src=s3://myawsbucket/logs/j-3GYXXXXXX9IOJ/node` **Important** S3DistCp does not support Amazon S3 bucket names that contain the underscore character.	Yes
`--dest=LOCATION`	Destination for the data. This can be either an HDFS or Amazon S3 location. Example: `--dest=hdfs:///output` **Important** S3DistCp does not support Amazon S3 bucket names that contain the underscore character.	Yes
`--srcPattern=PATTERN`	A regular expression that filters the copy operation to a subset of the data at `--src`. If neither `--srcPattern` nor `--groupBy` is specified, all data at `--src` is copied to `--dest`. If the regular expression argument contains special characters, such as an asterisk (*), either the regular	No

Option	Description	Required
	expression or the entire `--args` string must be enclosed in single quotes ('). Example: `--srcPattern=.*daemons.*-hadoop-.*`	
`--groupBy=PATTERN`	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. 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. If the regular expression argument contains special characters, such as an asterisk (*), either the regular expression or the entire `--args` string must be enclosed in single quotes ('). When `--groupBy` is specified, only files that match the specified pattern are copied. You do not need to specify `--groupBy` and `--srcPattern` at the same time. Example: `--groupBy=.*subnetid.*([0-9]+-[0-9]+-[0-9]+-[0-9]+).*`	No
`--targetSize=SIZE`	The size, in mebibytes (MiB), of the files to create based on the `--groupBy` option. This value must be an integer. When `--targetSize` 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 `--groupBy` are larger than the value of `--targetSize`, they are broken up into part files, and named sequentially with a numeric value appended to the end. For example, a file concatenated into `myfile.gz` would be broken into parts as: `myfile0.gz`, `myfile1.gz`, etc. Example: `--targetSize=2`	No

Option	Description	Required
`--appendToLastFile`	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 `--appendToLastFile` with `--groupBy`, new data is appended to files which match the same groups. This option also respects the `--targetSize` behavior when used with `--groupBy`.	No
`--outputCodec=CODEC`	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 `--outputCodec`, the files are copied over with no change in their compression. Example: `--outputCodec=lzo`	No
`--s3ServerSideEncryption`	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: `--s3ServerSideEncryption`	No
`--deleteOnSuccess`	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`	No
`--disableMultipartUpload`	Disables the use of multipart upload. Example: `--disableMultipartUpload`	No
`--multipartUploadChunkSize=SIZE`	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`	No
`--numberFiles`	Prepends output files with sequential numbers. The count starts at 0 unless a different value is specified by `--startingIndex`. Example: `--numberFiles`	No

Option	Description	Required
`--startingIndex=INDEX`	Used with `--numberFiles` to specify the first number in the sequence. Example: `--startingIndex=1`	No
`--outputManifest=FILENAME`	Creates a text file, compressed with Gzip, that contains a list of all the files copied by S3DistCp. Example: `--outputManifest=manifest-1.gz`	No
`--previousManifest=PATH`	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`	No
`--requirePreviousManifest`	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.	No
`--copyFromManifest`	Reverses the behavior of `--previousManifest` to cause S3DistCp to use the specified manifest file as a list of files to copy, instead of a list of files to exclude from copying. Example: `--copyFromManifest --previousManifest=/usr/bin/manifest-1.gz`	No
`--s3Endpoint=ENDPOINT`	Specifies the Amazon S3 endpoint to use when uploading a file. This option sets the endpoint for both the source and destination. If not set, the default endpoint is `s3.amazonaws.com`. For a list of the Amazon S3 endpoints, see Regions and Endpoints. Example: `--s3Endpoint=s3-eu-west-1.amazonaws.com`	No
`--storageClass=CLASS`	The storage class to use when the destination is Amazon S3. Valid values are STANDARD and REDUCED_REDUNDANCY. If this option is not specified, S3DistCp tries to preserve the storage class. Example: `--storageClass=STANDARD`	No

Option	Description	Required
`--srcPrefixesFile=PATH`	a text file in Amazon S3 (s3://), HDFS (hdfs:///) or local file system (file:/) that contains a list of `src` prefixes, one prefix per line. If `srcPrefixesFile` is provided, S3DistCp will not list the src path. Instead, it generates a source list as the combined result of listing all prefixes specified in this file. The relative path as compared to src path, instead of these prefixes, will be used to generate the destination paths. If `srcPattern` is also specified, it will be applied to the combined list results of the source prefixes to further filter the input. If `copyFromManifest` is used, objects in the manifest will be copied and `srcPrefixesFile` will be ignored. Example: `--srcPrefixesFile=PATH`	No

In addition to the options above, S3DistCp implements the Tool interface which means that it supports the generic options.

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. 207).

  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-3GYXXXXXX9IOK`* with your cluster ID and replace *`mybucket`* with your Amazon S3 bucket name.

```
[
    {
        "Name":"S3DistCp step",
```

```
        "Args":["s3-dist-cp","--s3Endpoint=s3.amazonaws.com","--src=s3://
mybucket/logs/j-3GYXXXXXX9IOJ/node/","--dest=hdfs:///output","--
srcPattern=.*[a-zA-Z,]+"],
        "ActionOnFailure":"CONTINUE",
        "Type":"CUSTOM_JAR",
        "Jar":"command-runner.jar"
    }
]
```

```
aws emr add-steps --cluster-id j-3GYXXXXXX9IOK --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-3GYXXXXXX9IOK* with your cluster ID and replace *mybucket* with your Amazon S3 bucket name.

```
[
    {
        "Name":"S3DistCp step",
        "Args":["s3-dist-cp","--s3Endpoint=s3.amazonaws.com","--
src=s3://mybucket/logs/j-3GYXXXXXX9IOJ/node/","--dest=hdfs:///output","--
srcPattern=.*daemons.*-hadoop-.*"],
        "ActionOnFailure":"CONTINUE",
        "Type":"CUSTOM_JAR",
        "Jar":"command-runner.jar"
    }
]
```

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..."]
```

Links to All Release Guides

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 5.2.1 (this guide)
- Release 5.2.0
- Release 5.1.0
- Release 5.0.3
- Release 5.0.0
- 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: January 3, 2017

Change	Description	Release Date
Amazon EMR Release 5.2.1	This guide supports the Amazon EMR 5.2.1 release.	January 3, 2017