Amazon Elastic MapReduce

Amazon EMR Release Guide

API Version 2009-03-31

Amazon Elastic MapReduce: Amazon EMR Release Guide

Table of Contents

About Amazon EMR Releases 1
- Applications 1
- Components 2
- Learn More 3

Differences Introduced in 4.x 4
- AMI Version vs. Release Label 4
- Installing Applications on the Cluster 5
- Configurations Replace Predefined Bootstrap Actions 6
  - Install Steps Are Deprecated 7
- Application Environment 7
- Service Ports 8
- Users 9
- Installation Sequence, Installed Artifact, and Log File Locations 10
- Command Runner 11

Configuring Applications 12

Apache Hadoop 17
- Create or Run a Hadoop Application 17
  - Build Binaries Using Amazon EMR 18
  - Run a Script in a Cluster 20
  - Process Data with Streaming 20
  - Process Data with a Custom JAR 24
- Configure Hadoop 25
  - Hadoop Daemon Settings 25
  - HDFS Configuration 35
  - Task Configuration 36

Apache Hive 48
- How Amazon EMR Hive Differs from Apache Hive 49
  - Combine Splits Input Format 49
  - Hive Authorization 49
  - Hive File Merge Behavior with Amazon S3 49
  - ACID Transactions and Amazon S3 49
  - Additional Features of Hive in Amazon EMR 49
- Use the Hive JDBC Driver 55

Apache Pig 57
- Submit Pig Work 57
  - Submit Pig Work Using the Amazon EMR Console 58
  - Submit Pig Work Using the AWS CLI 58
- Call User Defined Functions from Pig 59
  - Call JAR files from Pig 59
  - Call Python/Jython Scripts from Pig 59

Apache Spark 61
- Create a Cluster With Spark 62
- Configure Spark 63
  - Manually adjusting executor settings 63
  - Automatically configure executors with maximum resource allocation 64
  - Enabling Dynamic Allocation of Executors 65
- Access the Spark Shell 65
- Write a Spark Application 67
  - Scala 67
  - Java 68
  - Python 69
- Adding a Spark Step 69
  - Overriding Spark Default Configuration Settings 71

Apache Mahout 73

Amazon EMR Connectors and Utilities 74

EMR File System (EMRFS) (Optional) 74
Consistent View 75
Creating an AWSCredentialsProvider for EMRFS 89
Encryption in EMRFS 90
Export, Query, and Join Tables in DynamoDB 97
Set Up a Hive Table to Run Hive Commands 98
Hive Command Examples for Exporting, Importing, and Querying Data 102
Optimizing Performance 109
Amazon Kinesis 111
What Can I Do With Amazon EMR and Amazon Kinesis Integration? 111
Checkpointed Analysis of Amazon Kinesis Streams 112
Performance Considerations 113
Schedule Amazon Kinesis Analysis with Amazon EMR 113
Distributed Copy Using S3DistCp 113
S3DistCp Options 114
Adding S3DistCp as a Step in a Cluster 118
Document History 120

About Amazon EMR Releases

This document provides information about Amazon EMR software releases 4.0.0 or greater. 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 (p. 1)
- Components (p. 2)
- Learn More (p. 3)

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 currently supported for this release: Hadoop, Hive, Mahout, Pig, and Spark.

For information about applications and their associated components, see the following sections:

- Apache Hadoop (p. 17)
- Apache Hive (p. 48)
- Apache Mahout (p. 73)
- Apache Pig (p. 57)
- Apache Spark (p. 61)

Applications are essentially bundles of components.

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 will have a release version like the following:

communityVersion*-amzn-*emrReleaseVersion

As an example, assume that a component that has not been modified by Amazon EMR is Apache Mahout and the version is 0.10.0, which is the community version. However, another component, hive-client, is modified and its Amazon EMR release version is 1.0.0-amzn-0. Amazon components will just have one version number. For example, a emr-ddb version is 2.1.0.

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. For an example of using Hive to query DynamoDB, see Amazon EMR Hive queries to accommodate partial DynamoDB schemas (p. 53).

The following components are included with Amazon EMR:

- emr-ddb—DynamoDB connector for Hadoop ecosystem applications.

  Version: 3.0.0
- emr-goodies—Extra convenience libraries for the Hadoop ecosystem.

  Version: 2.0.0
- emr-kinesis—Amazon Kinesis connector for Hadoop ecosystem applications.

  Version: 3.0.0
- emr-s3-dist-cp—Distributed copy application optimized for Amazon S3.

  Version: 2.0.0
- emrfs—Amazon S3 connector for Hadoop ecosystem applications.

  Version: 2.0.0
- hadoop-client—Hadoop command-line clients such as 'hdfs', 'hadoop', or 'yarn'.

  Version: 2.6.0-amzn-0
- hadoop-hdfs-datanode—HDFS node-level service for storing blocks.

  Version: 2.6.0-amzn-0
- hadoop-hdfs-namenode—HDFS service for tracking file names and block locations.

  Version: 2.6.0-amzn-0
- hadoop-httpfs-server—HTTP endpoint for HDFS operations.

  Version: 2.6.0-amzn-0
- hadoop-mapred—MapReduce execution engine libraries for running a MapReduce application.

  Version: 2.6.0-amzn-0
- hadoop-yarn-nodemanager—YARN service for managing containers on an individual node.

  Version: 2.6.0-amzn-0

- hadoop-yarn-resourcemanager—YARN service for allocating and managing cluster resources and distributed applications.

  Version: 2.6.0-amzn-0
- hive-client—Hive command line client.

  Version: 1.0.0-amzn-0
- hive-metastore-server—Service for accessing the Hive metastore, a semantic repository storing metadata for SQL on Hadoop operations.

  Version: 1.0.0-amzn-0
- hive-server—Service for accepting Hive queries as web requests.

  Version: 1.0.0-amzn-0
- mahout-client—Library for machine learning.

  Version: 0.10.0
- mysql-server—MySQL database server.

  Version: 5.5
- pig-client—Pig command-line client.

  Version: 0.14.0-amzn-0
- spark-client—Spark command-line clients.

  Version: 1.4.1
- spark-history-server—Web UI for viewing logged events for the lifetime of a completed Spark application.

  Version: 1.4.1
- spark-on-yarn—In-memory execution engine for YARN.

  Version: 1.4.1
- spark-yarn-slave—Apache Spark libraries needed by YARN slaves.

  Version: 1.4.1

Learn More

If you are looking for additional information, we recommend 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 Elastic MapReduce API Reference
- AWS SDKs and other tools
- AWS Command Line Interface Reference
- Information about Amazon EMR AMI versions 2.x and 3.x — Amazon Elastic MapReduce Developer Guide

Differences Introduced in 4.x

We have made a series of changes to Amazon EMR releases that introduce differences between previous versions and the 4.0.0 release. The scope of changes range from how you create and configure your cluster to the ports and directory structure of applications on your cluster. The following sections detail these changes.

Topics

- AMI Version vs. Release Label (p. 4)
- Installing Applications on the Cluster (p. 5)
- Configurations Replace Predefined Bootstrap Actions (p. 6)
- Application Environment (p. 7)
- Service Ports (p. 8)
- Users (p. 9)
- Installation Sequence, Installed Artifact, and Log File Locations (p. 10)
- Command Runner (p. 11)

AMI Version vs. Release Label

Before Amazon EMR release 4.0.0, Amazon EMR software was referenced by its *AMI versions*. With Amazon EMR release 4.0.0 and later, releases are now referenced by their *release label*.

The following are ways of specifying release:

Console

Previously, in **Version** you chose the **AMI Version** and still do for 2.x and 3.x releases.

Choose **EMR release** for 4.x or later releases.

CLI

For AMI version releases 2.x and 3.x, specify `--ami-version 3.x.x`.

For EMR releases emr-4.0.0 or later, use `--release-label emr-4.x.x`.

API and SDK

In the API you provide either AmiVersion or ReleaseLabel depending on the respective releases.

In the Java SDK, the following RunJobFlowRequest call specifies an AMI version:

```
RunJobFlowRequest request = new RunJobFlowRequest()
   .withName("AmiVersion Cluster")
   .withAmiVersion("3.8.0")
   .withInstances(new JobFlowInstancesConfig()
    .withEc2KeyName("myKeyPair")
    .withInstanceCount(1)
    .withKeepJobFlowAliveWhenNoSteps(true)
    .withMasterInstanceType("m3.xlarge")
    .withSlaveInstanceType("m3.xlarge");
```

The following RunJobFlowRequest call uses a release label instead:

```
RunJobFlowRequest request = new RunJobFlowRequest()
   .withName("ReleaseLabel Cluster")
   .withReleaseLabel("emr-4.0.0")
   .withInstances(new JobFlowInstancesConfig()
    .withEc2KeyName("myKeyPair")
    .withInstanceCount(1)
    .withKeepJobFlowAliveWhenNoSteps(true)
    .withMasterInstanceType("m3.xlarge")
    .withSlaveInstanceType("m3.xlarge");
```

Installing Applications on the Cluster

In AMI versions 2.x and 3.x, applications were installed in any number of ways including: the NewSupportedProducts parameter for the RunJobFlow action, using bootstrap actions, and the Step action. With Amazon EMR release 4.x, there is a new, simpler way to install applications on your cluster:

Console

On the **Quick Create** page, applications are grouped in bundles. In emr-4.0.0 you can choose from All Applications (Hadoop, Spark, Pig, Hive, and Mahout), Core Hadoop (Hadoop, Hive, and Pig), and Spark (Spark and Hadoop-YARN). On the **Advanced cluster configuration** page, you can select the exact applications you want to install, and optionally edit the default configuration for each application. For more information about editing application configuration, see Configuring Applications (p. 12).

CLI

Installing applications is not changed using the CLI although you no longer provide application configuration with the Args parameter. You instead use the Configurations parameter to provide the path to a JSON-formatted file containing a set of configuration objects. You can store the file locally or in Amazon S3. For more information, see Configuring Applications (p. 12).

Java SDK

With emr-4.0.0, the preferred way to install applications using Java is to supply a list of Applications to RunJobFlowRequest. Using the AWS SDK for Java, this looks like:

```
List<Application> myApps = new ArrayList<Application>();
```

```
myApps.add("Spark","Hive","Mahout");

RunJobFlowRequest request = new RunJobFlowRequest()
 .withName("My EMR Cluster")
 .withReleaseLabel("emr-4.0.0")
 .withApplications(myApps)
 .withInstances(new JobFlowInstancesConfig()
     .withEc2KeyName("myKeyName")
     .withInstanceCount(1)
     .withKeepJobFlowAliveWhenNoSteps(true)
     .withMasterInstanceType("m3.xlarge")
     .withSlaveInstanceType("m3.xlarge");
);
```

Configurations Replace Predefined Bootstrap Actions

Application configuration is simplified with emr-4.0.0. Every application that you are able to specify in the Applications parameter supplied to the RunJobFlow action can be configured using a Configuration object. Furthermore, native applications are no longer configured by bootstrap actions. For example, this method replaces the configure-hadoop and configure-daemons bootstrap actions, which were used to configure certain applications. Those bootstrap actions are replaced by configurations. 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 object.

Important
If you try to use one of the previous bootstrap actions supported by Amazon EMR this causes a webservice error when attempting to launch with releases greater than emr-4.0.0. Custom bootstrap actions that do not attempt to configure native applications continue to work. The following bootstrap actions are no longer supported: configure-daemons, configure-hadoop, and s3Get.

Instead of using the s3get bootstrap action to copy objects to each node, use a custom bootstrap action which will run AWS CLI on each node. The syntax would look like:

```
aws s3 cp s3://mybucket/myfolder/myobject myFolder/
```

In the AWS SDK for Java, you do this with a ScriptBootstrapActionConfig:

```
ScriptBootstrapActionConfig s3Config = new ScriptBootstrapActionConfig()
 .withPath("file:///usr/bin/aws")
 .withArgs("s3", "cp","s3://mybucket/myfolder/myobject","myFolder/");
```

With the AWS CLI, you can launch the cluster with the bootstrap action using the following command:

```
aws emr create-cluster --release-label emr-4.0.0 --instance-type m3.xlarge --
instance-count 1 --bootstrap-actions Path=file:///usr/bin/aws,Name="copy
ToAll",Args="s3","cp","s3://mybucket/myfolder/myobject","myFolder/" --use-de
fault-roles
```

For more information, see Configuring Applications (p. 12)

Install Steps Are Deprecated

Certain predefined steps, such as those used to install Hive and Pig, are deprecated. Please use the configuration interface instead.

Application Environment

In Amazon EMR AMI versions 2.x and 3.x, there was a `hadoop-user-env.sh` script which was not part of standard Hadoop and was used along with the configure-daemons bootstrap action to configure Hadoop environment. The script did the following:

```
#!/bin/bash
export HADOOP_USER_CLASSPATH_FIRST=true;
echo "HADOOP_CLASSPATH=/path/to/my.jar" >> /home/hadoop/conf/hadoop-user-env.sh
```

In Amazon EMR release 4.0.0 and later, you can do the same now with the hadoop-env configurations:

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

         },
         "Configurations":[
            {
               "Classification":"export",
               "Properties":{
                  "HADOOP_USER_CLASSPATH_FIRST":"true",
                  "HADOOP_CLASSPATH":"/path/to/my.jar"
               }
            }
         ]
      }
   ]
```

You may have previously used a bootstrap action `configure-daemons` to pass environment. For example, if you set `--namenode-heap-size=2048` and `--namenode-opts=-XX:GCTimeRatio=19` with `configure-daemons`, the equivalent JSON would look like:

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

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

```
        ]
    }
]
```

Other application environment variables are no longer defined in `/home/hadoop/.bashrc`. Instead, they are primarily set in `/etc/default` files per component or application e.g. `/etc/default/hadoop`. Wrapper scripts in `/usr/bin/` installed by application RPMs may also set additional environment variables before involving the actual bin script.

Service Ports

In Amazon EMR AMI versions 2.x and 3.x, some services used custom ports. emr-4.0.0 hosts these services on open source community defaults in most cases.

Changes in Port Settings

Setting	AMI Version 3.x	Release Label emr-4.x
fs.default.name	hdfs://emrDeterminedIP:9000	default (hdfs://*emrDetermi-nedIP*:8020)
dfs.datanode.address	0.0.0.0:9200	default (0.0.0.0:50010)
dfs.datanode.http.address	0.0.0.0:9102	default (0.0.0.0:50075)
dfs.datanode.https.address	0.0.0.0:9402	default (0.0.0.0:50475)
dfs.datanode.ipc.address	0.0.0.0:9201	default (0.0.0.0:50020)
dfs.http.address	0.0.0.0:9101	default (0.0.0.0:50070)
dfs.https.address	0.0.0.0:9202	default (0.0.0.0:50470)
dfs.secondary.http.address	0.0.0.0:9104	default (0.0.0.0:50090)
yarn.nodemanager.address	0.0.0.0:9103	default (${yarn.nodemanager.host-name}:0)
yarn.nodemanager.localizer.ad-dress	0.0.0.0:9033	default (${yarn.nodemanager.host-name}:8040)
yarn.nodemanager.webapp.ad-dress	0.0.0.0:9035	default (${yarn.nodemanager.host-name}:8042)
yarn.resourcemanager.address	*emrDeterminedIP*:9022	default (${yarn.resourceman-ager.hostname}:8032)
yarn.resourcemanager.admin.ad-dress	*emrDeterminedIP*:9025	default (${yarn.resourceman-ager.hostname}:8033)
yarn.resourcemanager.resource-tracker.address	*emrDeterminedIP*:9023	default (${yarn.resourceman-ager.hostname}:8031)
yarn.resourcemanager.sched-uler.address	*emrDeterminedIP*:9024	default (${yarn.resourceman-ager.hostname}:8030)
yarn.resourceman-ager.webapp.address	0.0.0.0:9026	default (${yarn.resourceman-ager.hostname}:8088)

Setting	AMI Version 3.x	Release Label emr-4.x
yarn.web-proxy.address	*emrDeterminedIP*:9046	default (no-value)
yarn.resourcemanager.hostname	0.0.0.0 (default)	*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.

Users

In AMI versions 2.x and 3.x, the user `hadoop` ran all processes and owned all files. In Amazon EMR release 4.x, users exist at the application and component level. For example, here is a process status that demonstrates this user ownership model:

```
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 -Dha
doop.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 -Dha
doop.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 -Dh
base.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 -Dh
base.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 -Dha
doop.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 -Dha
doop.log.dir=/var/log/hadoop-yarn -Dyarn.log.dir=/var/log/hadoop-yarn -Dha
doop.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 -Dha
doop.log.dir=/var/log/hadoop-yarn -Dyarn.log.dir=/var/log/hadoop-yarn -Dha
doop.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 -Dha
doop.log.dir=/usr/lib/hadoop/logs -Dhadoop.log.file=hadoop.log -Dha
doop.home.dir=/usr/lib/hadoop
```

Installation Sequence, Installed Artifact, and Log File Locations

In AMI versions 2.x and 3.x, application artifacts and their configuration directories were previously installed to `/home/hadoop/application`. For example, if you installed Hive the directory would be `/home/hadoop/hive`. In EMR release 4.0.0 and later, application artifacts are installed in `/usr/lib/application`, so Hive would be located in /usr/lib/hive. Most configuration files are stored in `/etc/application/conf`, so hive-site, Hive's configuration file, would be located at `/etc/hive/conf`.

Previously, log files were found in various places. In Amazon EMR 4.x and later, they are now all located under `/var/log/component`.

Locations for log files pushed to Amazon S3 have changed as follows:

Changes in Log Locations on Amazon S3

Daemon or Application	AMI 3.x	emr-4.0.0
instance-state	node/*instance-id*/instance-state/	node/*instance-id*/instance-state/
hadoop-hdfs-namenode	daemons/*instance-id*/hadoop-hadoop-namenode.log	node/*instance-id*/applications/hadoop-hdfs/hadoop-hdfs-namenode-ip-*ipAddress*.log
hadoop-hdfs-datanode	daemons/*instance-id*/hadoop-hadoop-datanode.log	node/*instance-id*/applications/hadoop-hdfs/hadoop-hdfs-datanode-ip-*ipAddress*.log
hadoop-yarn (ResourceManager)	daemons/*instance-id*/yarn-hadoop-resourcemanager	node/*instance-id*/applications/hadoop-yarn/yarn-yarn-resourcemanager-ip-*ipAddress*.log
hadoop-yarn (Proxy Server)	daemons/*instance-id*/yarn-hadoop-proxyserver	node/*instance-id*/applications/hadoop-yarn/yarn-yarn-proxyserver-ip-*ipAddress*.log
mapred-historyserver	daemons/*instance-id*/	node/*instance-id*/applications/hadoop-mapreduce/mapred-mapred-historyserver-ip-*ipAddress*.log
httpfs	daemons/*instance-id*/httpfs.log	node/*instance-id*/applications/hadoop-httpfs/httpfs.log
hive-server	node/*instance-id*/hive-server/hive-server.log	node/*instance-id*/applications/hive/hive-server.log
hive-metastore	node/*instance-id*/apps/hive.log	node/*instance-id*/applications/hive/hive-metastore.log
Hive CLI	node/*instance-id*/apps/hive.log	node/*instance-id*/applications/hive/tmp/$username/hive.log
YARN applications user logs and container logs	task-attempts/	containers/

Daemon or Application	AMI 3.x	emr-4.0.0
Mahout	N/A	node/*instance-id*/applications/mahout
Pig	N/A	node/*instance-id*/applications/pig/pig.log
spark-historyserver	N/A	node/*instance-id*/applications/spark/spark-historyserver.log
mapreduce job history files	jobs/	hadoop-mapred/history/

Command Runner

Many scripts or programs, like `/home/hadoop/contrib/streaming/hadoop-streaming.jar`, are now 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 also 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 Run
ner",Jar="command-runner.jar",Args=["spark-submit","Args..."]
```

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. 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": [

        ]
      }
    ]
  }
]
```

Bootstrap actions that were previously used to configure Hadoop and other applications are replaced by configurations. The following tables give classifications for components and applications and corollary bootstrap actions for each. If the classification matches a file documented in the application project, see that respective project documentation for more details.

Hadoop

Filename	AMI Version Bootstrap Action	Release Label Classification
core-site.xml	configure-hadoop -c	core-site
log4j.properties	configure-hadoop -l	hadoop-log4j
hdfs-site.xml	configure-hadoop -s	hdfs-site
mapred-site.xml	configure-hadoop -m	mapred-site

Filename	AMI Version Bootstrap Action	Release Label Classification
yarn-site.xml	configure-hadoop -y	yarn-site
httpfs-site.xml	configure-hadoop -t	httpfs-site
capacity-scheduler.xml	configure-hadoop -z	capacity-scheduler
hadoop-env.sh	configure-daemons --client-opts	hadoop-env
httpfs-env.sh	n/a	httpfs-env
mapred-env.sh	n/a	mapred-env
yarn-env.sh	configure-daemons --resourcemanager-opts	yarn-env

Spark

Filename	AMI Version Bootstrap Action	Release Label Classification
spark-defaults.conf	n/a	spark-defaults
spark-env.sh	n/a	spark-env
log4j.properties	n/a	spark-log4j

Hive

Filename	AMI Version Bootstrap Action	Release Label Classification
hive-env.sh	n/a	hive-env
hive-site.xml	hive-script --install-hive-site ${MY_HIVE_SITE_FILE}	hive-site
hive-exec-log4j.properties	n/a	hive-exec-log4j
hive-log4j.properties	n/a	hive-log4j

Pig

Filename	AMI Version Bootstrap Action	Release Label Classification
pig.properties	n/a	pig-properties
log4j.properties	n/a	pig-log4j

EMRFS

Filename	AMI Version Bootstrap Action	Release Label Classification
emrfs-site.xml	configure-hadoop -e	emrfs-site

Filename	AMI Version Bootstrap Action	Release Label Classification
n/a	`s3get -s s3://custom-provider.jar -d /usr/share/aws/emr/auxlib/`	`emrfs-site` (with new setting `fs.s3.cse.encryptionMaterialsProvider.uri`)

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 maximum resources of each node

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 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-4.0.0 --instance-type m3.xlarge --
instance-count 2 --applications Name=Hive --configurations https://s3.amazon
aws.com/mybucket/myfolder/myConfig.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();
 hive.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-4.0.0")
 .withApplications(hive)
 .withConfigurations(myHiveConfig)
 .withServiceRole("EMR_DefaultRole")
 .withJobFlowRole("EMR_EC2_DefaultRole")
 .withInstances(new JobFlowInstancesConfig()
  .withEc2KeyName("myKey")
  .withInstanceCount(1)
  .withKeepJobFlowAliveWhenNoSteps(true)
  .withMasterInstanceType("m3.xlarge")
  .withSlaveInstanceType("m3.xlarge")
 );
```

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 not only increased processing and storage capacity, but also high availability.

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

> **Note**
> Hadoop Key Management Server (KMS) and the YARN Timeline Server are not supported in this release.

Release Information

Apache Hadoop 2.6.0-amzn-0

Amazon EMR Release Label: emr-4.0.0

Components Installed with Apache Hadoop

If you install Hive as an application in Amazon EMR, the following components will be installed:

emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager

Topics

- Create or Run a Hadoop Application (p. 17)
- Configure Hadoop (p. 25)

Create or Run a Hadoop Application

Topics

- Build Binaries Using Amazon EMR (p. 18)
- Run a Script in a Cluster (p. 20)
- Process Data with Streaming (p. 20)

- Process Data with a Custom JAR (p. 24)

Build Binaries Using Amazon EMR

You can use Amazon Elastic MapReduce (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/ElasticMapReduce/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, go to 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 Elastic MapReduce (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, go to Submit Work to a Cluster.

If you want to run a script before step processing begins, use a bootstrap action. For more information on bootstrap actions, go to (Optional) Create Bootstrap Actions to Install Additional Software.

Submitting a Custom JAR Step Using the AWS CLI

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. `Script-runner.jar` is located at `s3://elasticmapreduce/libs/script-runner/script-runner.jar`.

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 Amazon 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-4.0.0 -
-applications Name=Hive Name=Pig --use-default-roles --ec2-attributes Key
Name=myKey --instance-type m3.xlarge --instance-count 3 --steps Type=CUS
TOM_JAR,Name=CustomJAR,ActionOnFailure=CONTINUE,Jar=s3://elasticmapre
duce/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 will 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 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 Elastic MapReduce (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 a complete description of 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

Parameter	Description	Required
-cacheArchive	JAR file to extract into the working directory Type: String Default: None Constraints: [URI]#[symlink directory name to create in working directory	No
-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. Once 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**, click the name of your cluster.
3. Scroll to the **Steps** section and expand it, then click **Add step**.
4. In the **Add Step** dialog:

   - 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 Amazon 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 Amazon 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. Click **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, click 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-4.0.0 -
-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,-in
put,pathtoinputdata,-output,pathtooutputbucket]
```

  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.

  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.

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 on compiling a JAR file, see Build Binaries Using Amazon EMR (p. 18).

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**, click the name of your cluster.
3. Scroll to the **Steps** section and expand it, then click **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. Click **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, click 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-4.0.0 -
-applications Name=Hue Name=Hive Name=Pig --use-default-roles --ec2-attributes
 KeyName=myKey --instance-type m3.xlarge --instance-count 3 --steps
Type=CUSTOM_JAR,Name="Custom JAR Step",ActionOnFailure=CONTINUE,Jar=pathto
jarfile,Args=["pathtoinputdata","pathtooutputbucket","arg1","arg2"]
```

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.

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.

Configure Hadoop

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

Topics

- Hadoop Daemon Settings (p. 25)
- HDFS Configuration (p. 35)
- Task Configuration (p. 36)

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_RESOURCEMAN-AGER_HEAPSIZE	384
YARN_PROXYSERVER_HEAPSIZE	192
YARN_NODEMANAGER_HEAPSIZE	256
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	256
HADOOP_NAMENODE_HEAPSIZE	384
HADOOP_DATANODE_HEAPSIZE	192

m1.large

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	768
YARN_PROXYSERVER_HEAPSIZE	384
YARN_NODEMANAGER_HEAPSIZE	512

Parameter	Value
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	512
HADOOP_NAMENODE_HEAPSIZE	768
HADOOP_DATANODE_HEAPSIZE	384

m1.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	1024
YARN_PROXYSERVER_HEAPSIZE	512
YARN_NODEMANAGER_HEAPSIZE	768
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1024
HADOOP_NAMENODE_HEAPSIZE	2304
HADOOP_DATANODE_HEAPSIZE	384

m2.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	1536
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1024
HADOOP_NAMENODE_HEAPSIZE	3072
HADOOP_DATANODE_HEAPSIZE	384

m2.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	1536
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1536

Parameter	Value
HADOOP_NAMENODE_HEAPSIZE	6144
HADOOP_DATANODE_HEAPSIZE	384

m2.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

m3.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2396
YARN_PROXYSERVER_HEAPSIZE	2396
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2396
HADOOP_NAMENODE_HEAPSIZE	1740
HADOOP_DATANODE_HEAPSIZE	757

m3.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2703
YARN_PROXYSERVER_HEAPSIZE	2703
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2703
HADOOP_NAMENODE_HEAPSIZE	3276
HADOOP_DATANODE_HEAPSIZE	1064

c1.medium

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	192
YARN_PROXYSERVER_HEAPSIZE	96
YARN_NODEMANAGER_HEAPSIZE	128
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	128
HADOOP_NAMENODE_HEAPSIZE	192
HADOOP_DATANODE_HEAPSIZE	96

c1.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	768
YARN_PROXYSERVER_HEAPSIZE	384
YARN_NODEMANAGER_HEAPSIZE	512
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	512
HADOOP_NAMENODE_HEAPSIZE	768
HADOOP_DATANODE_HEAPSIZE	384

c3.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2124
YARN_PROXYSERVER_HEAPSIZE	2124
YARN_NODEMANAGER_HEAPSIZE	2124
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2124
HADOOP_NAMENODE_HEAPSIZE	972
HADOOP_DATANODE_HEAPSIZE	588

c3.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2396
YARN_PROXYSERVER_HEAPSIZE	2396
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2396
HADOOP_NAMENODE_HEAPSIZE	1740
HADOOP_DATANODE_HEAPSIZE	757

c3.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2703
YARN_PROXYSERVER_HEAPSIZE	2703
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2703
HADOOP_NAMENODE_HEAPSIZE	3276
HADOOP_DATANODE_HEAPSIZE	1064

c3.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	3317
YARN_PROXYSERVER_HEAPSIZE	3317
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	3317
HADOOP_NAMENODE_HEAPSIZE	6348
HADOOP_DATANODE_HEAPSIZE	1679

cc2.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

cg1.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	3840
HADOOP_DATANODE_HEAPSIZE	384

cr1.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

d2.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

d2.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

d2.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

d2.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	7089
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

hi1.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	3328
YARN_PROXYSERVER_HEAPSIZE	3328
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	3328
HADOOP_NAMENODE_HEAPSIZE	6400
HADOOP_DATANODE_HEAPSIZE	1689

hs1.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2048
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1536
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1536
HADOOP_NAMENODE_HEAPSIZE	12288
HADOOP_DATANODE_HEAPSIZE	384

i2.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

i2.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

i2.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

i2.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	7086
HADOOP_NAMENODE_HEAPSIZE	25190
HADOOP_DATANODE_HEAPSIZE	4096

g2.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	1536
YARN_PROXYSERVER_HEAPSIZE	1024
YARN_NODEMANAGER_HEAPSIZE	1024
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	1024
HADOOP_NAMENODE_HEAPSIZE	2304
HADOOP_DATANODE_HEAPSIZE	384

r3.xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	2713
YARN_PROXYSERVER_HEAPSIZE	2713
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	2713
HADOOP_NAMENODE_HEAPSIZE	3328
HADOOP_DATANODE_HEAPSIZE	1075

r3.2xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	3338
YARN_PROXYSERVER_HEAPSIZE	3338
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	3338
HADOOP_NAMENODE_HEAPSIZE	6451
HADOOP_DATANODE_HEAPSIZE	1699

r3.4xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	4587
YARN_PROXYSERVER_HEAPSIZE	4587
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_HEAPSIZE	4587
HADOOP_NAMENODE_HEAPSIZE	12697
HADOOP_DATANODE_HEAPSIZE	2949

r3.8xlarge

Parameter	Value
YARN_RESOURCEMAN-AGER_HEAPSIZE	7086
YARN_PROXYSERVER_HEAPSIZE	7086
YARN_NODEMANAGER_HEAPSIZE	2048
HADOOP_JOB_HISTORYSERV-ER_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	This determines how many copies of each block to store for durability. For small clusters, we 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 a configure-hadoop bootstrap action.	1 for clusters < four nodes 2 for clusters < ten nodes 3 for all other clusters

Task Configuration

Topics

- Task JVM Memory Settings (p. 36)

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 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.

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
mapreduce.map.java.opts	-Xmx512m

Configuration Option	Default Value
mapreduce.reduce.java.opts	-Xmx1024m
mapreduce.map.memory.mb	768
mapreduce.reduce.memory.mb	1536
yarn.app.mapreduce.am.resource.mb	1536
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	3072
yarn.nodemanager.resource.memory-mb	5120

m1.xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx512m
mapreduce.reduce.java.opts	-Xmx1536m
mapreduce.map.memory.mb	768
mapreduce.reduce.memory.mb	2048
yarn.app.mapreduce.am.resource.mb	2048
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	8192
yarn.nodemanager.resource.memory-mb	12288

m2.xlarge

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

m2.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1536
mapreduce.reduce.memory.mb	2560
yarn.app.mapreduce.am.resource.mb	2560
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	8192
yarn.nodemanager.resource.memory-mb	30720

m3.xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1440
mapreduce.reduce.memory.mb	2880
yarn.scheduler.minimum-allocation-mb	1440
yarn.scheduler.maximum-allocation-mb	11520
yarn.nodemanager.resource.memory-mb	11520

m3.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1440
mapreduce.reduce.memory.mb	2880
yarn.scheduler.minimum-allocation-mb	1440
yarn.scheduler.maximum-allocation-mb	23040
yarn.nodemanager.resource.memory-mb	23040

m2.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1536
mapreduce.reduce.memory.mb	2560
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	8192
yarn.nodemanager.resource.memory-mb	61440

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.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	512
yarn.nodemanager.resource.memory-mb	1024

c1.xlarge

Configuration Option	Default Value
io.sort.mb	150
mapreduce.map.java.opts	-Xmx864m
mapreduce.reduce.java.opts	-Xmx1536m
mapreduce.map.memory.mb	1024
mapreduce.reduce.memory.mb	2048
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	2048
yarn.nodemanager.resource.memory-mb	5120

c3.xlarge

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

c3.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1440
mapreduce.reduce.memory.mb	2880
yarn.scheduler.minimum-allocation-mb	1440
yarn.scheduler.maximum-allocation-mb	11520
yarn.nodemanager.resource.memory-mb	11520

c3.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1152m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1440
mapreduce.reduce.memory.mb	2880
yarn.scheduler.minimum-allocation-mb	1440
yarn.scheduler.maximum-allocation-mb	23040
yarn.nodemanager.resource.memory-mb	23040

c3.8xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1331m

Configuration Option	Default Value
mapreduce.reduce.java.opts	-Xmx2662m
mapreduce.map.memory.mb	1664
mapreduce.reduce.memory.mb	3328
yarn.scheduler.minimum-allocation-mb	1664
yarn.scheduler.maximum-allocation-mb	53248
yarn.nodemanager.resource.memory-mb	53248

cg1.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx1280m
mapreduce.reduce.java.opts	-Xmx2304m
mapreduce.map.memory.mb	1536
mapreduce.reduce.memory.mb	2560
yarn.app.mapreduce.am.resource.mb	2560
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	5120
yarn.nodemanager.resource.memory-mb	20480

cc2.8xlarge

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

cr1.8xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx6042m

Configuration Option	Default Value
mapreduce.reduce.java.opts	-Xmx12084m
mapreduce.map.memory.mb	7552
mapreduce.reduce.memory.mb	15104
yarn.scheduler.minimum-allocation-mb	7552
yarn.scheduler.maximum-allocation-mb	241664
yarn.nodemanager.resource.memory-mb	241664

d2.xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m
mapreduce.map.memory.mb	2928
mapreduce.reduce.memory.mb	5856
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	23424
yarn.nodemanager.resource.memory-mb	23424

d2.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m
mapreduce.map.memory.mb	3392
mapreduce.reduce.memory.mb	6784
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	54272
yarn.nodemanager.resource.memory-mb	54272

d2.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2918m
mapreduce.reduce.java.opts	-Xmx5836m
mapreduce.map.memory.mb	3648

Configuration Option	Default Value
mapreduce.reduce.memory.mb	7296
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	116736
yarn.nodemanager.resource.memory-mb	116736

d2.8xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2685m
mapreduce.reduce.java.opts	-Xmx5370m
mapreduce.map.memory.mb	3356
mapreduce.reduce.memory.mb	6712
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	241664
yarn.nodemanager.resource.memory-mb	241664

g2.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx512m
mapreduce.reduce.java.opts	-Xmx1536m
mapreduce.map.memory.mb	768
mapreduce.reduce.memory.mb	2048
yarn.app.mapreduce.am.resource.mb	2048
yarn.scheduler.minimum-allocation-mb	256
yarn.scheduler.maximum-allocation-mb	8192
yarn.nodemanager.resource.memory-mb	12288

hi1.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2688m
mapreduce.reduce.java.opts	-Xmx5376m
mapreduce.map.memory.mb	3360
mapreduce.reduce.memory.mb	6720

Configuration Option	Default Value
yarn.scheduler.minimum-allocation-mb	3360
yarn.scheduler.maximum-allocation-mb	53760
yarn.nodemanager.resource.memory-mb	53760

hs1.8xlarge

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

i2.xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m
mapreduce.map.memory.mb	2928
mapreduce.reduce.memory.mb	5856
yarn.scheduler.minimum-allocation-mb	2928
yarn.scheduler.maximum-allocation-mb	23424
yarn.nodemanager.resource.memory-mb	23424

i2.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m
mapreduce.map.memory.mb	3392
mapreduce.reduce.memory.mb	6784
yarn.scheduler.minimum-allocation-mb	3392

Configuration Option	Default Value
yarn.scheduler.maximum-allocation-mb	54272
yarn.nodemanager.resource.memory-mb	54272

i2.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2918m
mapreduce.reduce.java.opts	-Xmx5836m
mapreduce.map.memory.mb	3648
mapreduce.reduce.memory.mb	7296
yarn.scheduler.minimum-allocation-mb	3648
yarn.scheduler.maximum-allocation-mb	116736
yarn.nodemanager.resource.memory-mb	116736

i2.8xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx3021m
mapreduce.reduce.java.opts	-Xmx6042m
mapreduce.map.memory.mb	15974
mapreduce.reduce.memory.mb	16220
yarn.scheduler.minimum-allocation-mb	3776
yarn.scheduler.maximum-allocation-mb	241664
yarn.nodemanager.resource.memory-mb	241664

r3.xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2342m
mapreduce.reduce.java.opts	-Xmx4684m
mapreduce.map.memory.mb	2982
mapreduce.reduce.memory.mb	5856
yarn.scheduler.minimum-allocation-mb	2928
yarn.scheduler.maximum-allocation-mb	23424
yarn.nodemanager.resource.memory-mb	23424

r3.2xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx2714m
mapreduce.reduce.java.opts	-Xmx5428m
mapreduce.map.memory.mb	3392
mapreduce.reduce.memory.mb	6784
yarn.scheduler.minimum-allocation-mb	3392
yarn.scheduler.maximum-allocation-mb	54272
yarn.nodemanager.resource.memory-mb	54272

r3.4xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx5837m
mapreduce.reduce.java.opts	-Xmx11674m
mapreduce.map.memory.mb	7296
mapreduce.reduce.memory.mb	14592
yarn.scheduler.minimum-allocation-mb	7296
yarn.scheduler.maximum-allocation-mb	116736
yarn.nodemanager.resource.memory-mb	116736

r3.8xlarge

Configuration Option	Default Value
mapreduce.map.java.opts	-Xmx6042m
mapreduce.reduce.java.opts	-Xmx12084m
mapreduce.map.memory.mb	7552
mapreduce.reduce.memory.mb	15104
yarn.scheduler.minimum-allocation-mb	7552
yarn.scheduler.maximum-allocation-mb	241664
yarn.nodemanager.resource.memory-mb	241664

You can start a new JVM for every task, which provides better task isolation, or you can share JVMs between tasks, providing lower framework overhead. If you are processing many small files, it makes sense to reuse the JVM many times to amortize the cost of start-up. However, if each task takes a long time or processes a large amount of data, then you might choose to not reuse the JVM to ensure that all memory is freed for subsequent tasks.

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

To modify JVM settings using the AWS CLI

To modify JVM settings using the AWS CLI, type the `--bootstrap-action` parameter and specify the settings in the arguments list.

- To configure infinite JVM reuse, 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-4.0.0 -
--use-default-roles --ec2-attributes KeyName=myKey --instance-count 3 --ap
plications Name=Hadoop --configurations Classification=mapred-site,Proper
ties=["mapred.job.reuse.jvm.num.tasks"="-1"]
```

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 with a bootstrap action. A value of `-1` means infinite reuse within a single job, and `1` means do not reuse tasks.

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

Apache Hive

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

Hive extends the SQL paradigm by including serialization formats and the ability to invoke mapper and reducer scripts. 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 a more information on Hive, go to http://hive.apache.org/.

Release Information

Apache Hive 1.0.0-amzn-0

Amazon EMR Release Label: emr-4.0.0

Components Installed with Hive

If you install Hive as an application in Amazon EMR, the following components will be installed:

emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hive-client, hive-metastore-server, hive-server, mysql-server

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

- How Amazon EMR Hive Differs from Apache Hive (p. 49)
- Use the Hive JDBC Driver (p. 55)

How Amazon EMR Hive Differs from Apache Hive

Topics

- Combine Splits Input Format (p. 49)
- Hive Authorization (p. 49)
- Hive File Merge Behavior with Amazon S3 (p. 49)
- ACID Transactions and Amazon S3 (p. 49)
- Additional Features of Hive in Amazon EMR (p. 49)

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

Combine Splits Input Format

If you have many GZip files in your Hive cluster, you can optimize performance by passing multiple files to each mapper. This reduces the number of mappers needed in your cluster and can help your clusters complete faster. You do this by specifying that Hive use the `HiveCombineSplitsInputFormat` input format and setting the split size, in bytes. This is shown in the following example

```
set hive.hadoop.supports.splittable.combineinputformat=true;
```

Hive Authorization

Amazon EMR supports Hive Authorization for HDFS but not for EMRFS and Amazon S3. Amazon EMR clusters run with authorization disabled by default.

Hive File Merge Behavior with Amazon S3

Apache Hive merges small files at the end of a map-only job if `hive.merge.mapfiles` is true and the merge is triggered only if the average output size of the job is less than the `hive.merge.smallfiles.avgsize` setting. Amazon EMR Hive has exactly the same behavior if the final output path is in HDFS; however, if the output path is in Amazon S3, the `hive.merge.smallfiles.avgsize` parameter is ignored. In that situation, the merge task is always triggered if `hive.merge.mapfiles` is set to `true`.

ACID Transactions and Amazon S3

ACID (Atomicity, Consistency, Isolation, Durability) transactions are not supported with Hive data stored in Amazon S3. If you attempt to create a transactional table in Amazon S3, this will cause an exception.

Additional Features of Hive in 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. For information about which versions of Hive support these additional features, see Hive Patches (p. 54).

Topics

- Write Data Directly to Amazon S3 (p. 50)

- Use Hive to Access Resources in Amazon S3 (p. 50)
- Variables in Hive (p. 51)
- Amazon EMR Hive queries to accommodate partial DynamoDB schemas (p. 53)
- Copy data between DynamoDB tables in different AWS regions (p. 54)
- Set DynamoDB throughput values per table (p. 54)
- Hive Patches (p. 54)

Write Data Directly to Amazon S3

The Hadoop Distributed File System (HDFS) and Amazon S3 are handled differently within Amazon EMR and Hive. The version of Hive installed with Amazon EMR is extended with the ability to write directly to Amazon S3 without the use of temporary files. This produces a significant performance improvement but it means that HDFS and Amazon S3 behave differently within Hive.

A consequence of Hive writing directly to Amazon S3 is that you cannot read and write to the same table within the same Hive statement if that table is located in Amazon S3. The following example shows how to use multiple Hive statements to update a table in Amazon S3.

To update a table in Amazon S3 using Hive

1. From a Hive prompt or script, create a temporary table in the cluster's local HDFS filesystem.
2. Write the results of a Hive query to the temporary table.
3. Copy the contents of the temporary table to Amazon S3. This is shown in the following example.

```
CREATE TEMPORARY TABLE tmp LIKE my_s3_table;
INSERT OVERWRITE TABLE tmp SELECT ....;
INSERT OVERWRITE TABLE my_s3_table SELECT * FROM tmp;
```

Use Hive to Access Resources in Amazon S3

The version of Hive installed in Amazon EMR enables you to reference resources, such as JAR files, located in Amazon S3.

```
add jar s3://elasticmapreduce/samples/hive-ads/libs/jsonserde.jar
```

You can also reference scripts located in Amazon S3 to execute custom map and reduce operations. This is shown in the following example.

```
from logs select transform (line)
using 's3://mybucket/scripts/parse-logs.pl' as
(time string, exception_type string, exception_details string)
```

Variables in Hive

You can include variables in your scripts by using the dollar sign and curly braces.

```
add jar ${LIB}/jsonserde.jar
```

You pass the values of these variables to Hive on the command line using the `-d` parameter, as in the following example:

```
-d LIB=s3://elasticmapreduce/samples/hive-ads/lib
```

You can also pass the values into steps that execute Hive scripts.

To pass variable values into Hive steps using the console

1. Open the Amazon Elastic MapReduce console at https://console.aws.amazon.com/elasticmapreduce/.
2. Click **Create cluster**.
3. In the **Steps** section, for **Add Step**, choose **Hive Program** from the list and click **Configure and add**.
4. In the **Add Step** dialog, specify the parameters using the following table as a guide, and then click **Add**.

Field	Action
Script S3 location*	Specify the URI where your script resides in Amazon S3. The value must be in the form *BucketName*/*path*/*ScriptName*. For example: `s3://elasticmapreduce/samples/hive-ads/libs/response-time-stats.q`.
Input S3 location	Optionally, specify the URI where your input files reside in Amazon S3. The value must be in the form *BucketName*/*path*/. If specified, this will be passed to the Hive script as a parameter named `INPUT`. For example: `s3://elasticmapreduce/samples/hive-ads/tables/`.
Output S3 location	Optionally, specify the URI where you want the output stored in Amazon S3. The value must be in the form *BucketName*/*path*. If specified, this will be passed to the Hive script as a parameter named `OUTPUT`. For example: `s3://mybucket/hive-ads/output/`.

Arguments	Optionally, enter a list of arguments (space-separated strings) to pass to Hive. If you defined a path variable in your Hive script named ${SAMPLE}, for example:
	`CREATE EXTERNAL TABLE logs (requestBeginTime STRING, requestEndTime STRING, hostname STRING) PARTITIONED BY (dt STRING) \`
	`ROW FORMAT serde 'com.amazon.elasticmapreduce.JsonSerde'`
	`WITH SERDEPROPERTIES ( 'paths'='requestBeginTime, requestEndTime, hostname' ) LOCATION '${SAMPLE}/tables/impressions';`
	To pass a value for the variable, type the following in the **Arguments** window:
	`-d SAMPLE=s3://elasticmapreduce/samples/hive-ads/`.

Field	Action
Action on Failure	This determines what the cluster does in response to any errors. The possible values for this setting are: • **Terminate cluster**: If the step fails, terminate the cluster. If the cluster has termination protection enabled AND keep alive enabled, it will not terminate. • **Cancel and wait**: If the step fails, cancel the remaining steps. If the cluster has keep alive enabled, the cluster will not terminate. • **Continue**: If the step fails, continue to the next step.

5. Select values as necessary and choose **Create cluster**.

To pass variable values into Hive steps using the AWS CLI

To pass variable values into Hive steps using the AWS CLI, use the `--steps` parameter and include an arguments list.

- To pass a variable into a Hive step using the AWS CLI, type the following command, replace *myKey* with the name of your EC2 key pair, and replace *mybucket* with your bucket name. In this example, `SAMPLE` is a variable value preceded by the `-d` switch. This variable is defined in the Hive script as: `${SAMPLE}`.

  •

```
aws emr create-cluster --name "Test cluster" --release-label emr-4.0.0 --applications Name=Hive Name=Pig --use-default-roles --ec2-attributes KeyName=myKey --instance-type m3.xlarge --instance-count 3 --steps Type=Hive,Name="Hive Program",ActionOnFailure=CONTINUE,Args=[-f,s3://elasticmapreduce/samples/hive-ads/libs/response-time-stats.q,-d,INPUT=s3://elasticmapreduce/samples/hive-ads/tables,-d,OUTPUT=s3://mybucket/hive-ads/output/,-d,SAMPLE=s3://elasticmapreduce/samples/hive-ads/]
```

  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.

  > **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.

To pass variable values into Hive steps using the Java SDK

- The following example demonstrates how to pass variables into steps using the SDK. For more information, see Class StepFactory in the *AWS SDK for Java API Reference*.

```
StepFactory stepFactory = new StepFactory();

   StepConfig runHive = new StepConfig()
     .withName("Run Hive Script")
     .withActionOnFailure("TERMINATE_JOB_FLOW")
     .withHadoopJarStep(stepFactory.newRunHiveScriptStep(“s3://mybuck
et/script.q”,
      Lists.newArrayList(“-d”,”LIB= s3://elasticmapreduce/samples/hive-
ads/lib”));
```

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 record
TimeStamp < "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.

Query Example	Result Description
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");
```

Hive Patches

The Amazon EMR team has created the following patches for Hive.

The following patches were applied to Hive 1.0 for the EMR 4.0 release:

HIVE-10319
: This patch provides a performance improvement to Hive CLI startup when the Hive metastore has a large amount of databases.

HIVE-2777
: This patch provides the ability to atomically add and drop partitions.

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: http://amazon-odbc-jdbc-drivers.s3.amazonaws.com/public/HiveJDBC.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
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
```

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

```
com.amazon.hive.jdbc4.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.

Apache Pig

Amazon Elastic MapReduce (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/. Amazon EMR supports several versions of Pig.

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 MapReduce jobs. You do not have to write complex MapReduce 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

Apache Pig 0.14.0

Amazon EMR Release Label: emr-4.0.0

Components Installed with Apache Pig

If you install Apache Pig as an application in Amazon EMR, the following components will be installed:

emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, pig-client

Topics

- Submit Pig Work (p. 57)
- Call User Defined Functions from Pig (p. 59)

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.

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-4.0.0 --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]
```

  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 on 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. 57).
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, 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 my
functions;
```

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);
```

Apache Spark

Apache Spark is a cluster framework and programming model that helps you process data. 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

Apache Spark 1.4.1

Amazon EMR Release Label: emr-4.0.0

Components Installed with Apache Spark

If you install Apache Spark as an application in Amazon EMR, the following components will be installed:

emrfs, emr-goodies, emr-s3-dist-cp, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, spark-client, spark-history-server, spark-on-yarn, spark-yarn-shuffle

Topics

- Create a Cluster With Spark (p. 62)
- Configure Spark (p. 63)
- Access the Spark Shell (p. 65)

- Write a Spark Application (p. 67)
- Adding a Spark Step (p. 69)

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 the **Software Configuration** field, choose **Amazon Release Version emr-4.0.0** or later.
4. In the **Select Applications** field, 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. 63).

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-4.0.0 --
applications Name=Spark --ec2-attributes KeyName=myKey --instance-type
m3.xlarge --instance-count 3 --use-default-roles
```

To launch a cluster with Spark installed using the AWS 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(creden
tials);

Application sparkApp = new Application()
    .withName("Spark");
Applications myApps = new Applications();
myApps.add(sparkApp);

RunJobFlowRequest request = new RunJobFlowRequest()
    .withName("Spark Cluster")
    .withApplications(myApps)
    .withReleaseLabel("emr-4.0.0")
    .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 any of the options listed in the Spark Configuration topic in the Apache Spark documentation using the `spark-defaults` file. These settings reflect the community defaults. It is also possible to configure Spark dynamically at the time of each application submission. Additionally, we have introduced a new setting to automatically maximize the resource allocation for an executor, and it is available using the `spark` configuration file introduced by Amazon EMR. For more information, see the section called “Overriding Spark Default Configuration Settings” (p. 71).

Topics

- Manually adjusting executor settings (p. 63)
- Automatically configure executors with maximum resource allocation (p. 64)
- Enabling Dynamic Allocation of Executors (p. 65)

Manually adjusting executor settings

The following procedures show how to set executor 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 1024m, using the following:

```
aws emr create-cluster --release-label emr-4.0.0 --instance-type m3.xlarge
 --instance-count 2 --applications Name=Spark --configurations ht
tps://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

`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-4.0.0 --instance-type
m3.xlarge --instance-count 3 --applications Name=Spark --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 emr-4.0.0** 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**.

Automatically configure executors with maximum resource allocation

You can configure your executors to utilize the maximum resources possible on each node in your cluster by enabling the maximizeResourceAllocation option when creating your 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. It also sets the number of executors—by setting a value for `spark.executor.instances`—to the initial core nodes specified when creating your 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-4.0.0 --instance-type m3.xlarge
 --instance-count 2 --applications Name=Spark --configurations file://./my
Config.json
```

Or using Amazon S3:

```
aws emr create-cluster --release-label emr-4.0.0 --instance-type m3.xlarge
 --instance-count 2 --applications Name=Spark --configurations ht
tps://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

`myConfig.json`:

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

Enabling Dynamic Allocation of Executors

Spark on YARN has the ability to dynamically scale the number of executors used for a Spark application. You still need to set the memory and cores used for an executor but YARN will automatically allocate executors to the Spark application as needed. To enable dynamic allocation of executors, set `spark.dynamicAllocation.enabled` to true in the `spark-defaults` configuration file.

The setting `spark.shuffle.service.enabled` is automatically set to true by default, which is required by the dynamic allocation feature. If you enable dynamic allocation when you create your cluster, you must then disable it if you also pass a value for `spark.executor.instances` with a **spark-submit** command. Conversely, you cannot enable dynamic allocation if you set a value for `spark.executor.instances` in `spark-defaults` when you create your cluster. To learn more about dynamic allocation, see the Dynamically Loading Spark Properties topic in the Apache Spark documentation.

To create a cluster with dynamic allocation of executors

- Create a cluster with Spark installed and `spark.dynamicAllocation.enabled` set to `true`, `spark.executor.memory` set to `2G`, and `spark.executor.cores` set to `2` using the following:

```
aws emr create-cluster --release-label emr-4.0.0 --instance-type m3.xlarge
 --instance-count 3 --applications Name=Spark --configurations file://./my
Config.json
```

`myConfig.json`:

```
[
  {
    "Classification": "spark-defaults",
    "Properties": {
      "spark.dynamicAllocation.enabled": "true",
      "spark.executor.memory": "2G",
      "spark.executor.cores": "2"
    }
  }
]
```

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-4.0.0 --instance-type
m3.xlarge --instance-count 3 --applications Name=Spark --configurations
 https://s3.amazonaws.com/mybucket/myfolder/myConfig.json
```

Access the Spark Shell

The Spark shell is based on the Scala REPL (Read-Eval-Print-Loop). It allows you to create Spark programs interactively and submit work to the framework. You can access the Spark shell by connecting to the master node with SSH and invoking `spark-shell`. For more information about connecting to the master node, see Connect to the Master Node Using SSH in the Amazon EMR Management Guide . The following examples use Apache HTTP Server access logs stored in Amazon S3.

Note
The bucket used in these examples is available to clients that can access US East (N. Virginia).

By default, the Spark shell creates its own SparkContext object called `sc`. You can use this context if it is required within the REPL. sqlContext is also available in the shell and it is a HiveContext.

Example Using the Spark shell to count the occurrences of a string in a file stored in Amazon S3

This example uses `sc` to read a textFile in Amazon S3.

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@404721db

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("car
toonnetwork.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/im
pressions/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 examplecounts 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 will run 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.

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-
4.0.0 --applications Name=Spark --ec2-attributes KeyName=myKey --instance-
type m3.xlarge --instance-count 3 --steps Type=Spark,Name="Spark Program",Ac
tionOnFailure=CONTINUE,Args=[--class,org.apache.spark.ex
amples.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-
4.0.0 --applications Name=Spark --ec2-attributes KeyName=myKey --instance-
type m3.xlarge --instance-count 3 --steps Type=CUSTOM_JAR,Name="Spark Pro
gram",Jar="command-runner.jar",ActionOnFailure=CONTIN
UE,Args=[/usr/lib/spark/bin/run-example,SparkPi,10] --use-default-roles
```

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.ex
amples.SparkPi,/usr/lib/spark/lib/spark-examples-*.jar,10] --use-default-
roles
```

   An alternative using `command-runner.jar`:

```
aws emr add-steps --cluster-id j-2AXXXXXXGAPLF --steps Type=CUS
TOM_JAR,Name="Spark Program",Jar="command-runner.jar",ActionOnFailure=CON
TINUE,Args=[/usr/lib/spark/bin/run-example,SparkPi,10
```

To submit work to Spark using the AWS SDK for Java

- To submit work to a cluster, use a step to run the `spark-submit` script on your EMR cluster. You 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(creden
tials);
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-ex
amples.jar","10");

StepConfig sparkStep = new StepConfig()
   .withName("Spark Step")
   .withActionOnFailure("CONTINUE")
   .withHadoopJarStep(sparkStepConf);

stepConfigs.add(sparkStep);
req.withSteps(stepConfigs);
AddJobFlowStepsResult result = emr.addJobFlowSteps(req);
```

Overriding Spark Default Configuration Settings

It is probable that you will want to override Spark default configuration values on a per-application basis. You can do this when you submit applications using a step, which is 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:

```
/home/hadoop/spark/bin/spark-submit --executor-memory 1g --class
org.apache.spark.examples.SparkPi /home/hadoop/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 /home/ha
doop/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.

Apache Mahout

Amazon Elastic MapReduce (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 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

Apache Mahout 0.10.0

Amazon EMR Release Label: emr-4.0.0

Components Installed with Apache Mahout

If you install Apache Mahout as an application in Amazon EMR, the following components will be installed:

emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, mahout-client

Amazon EMR 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) (p. 74)
- Export, Import, Query, and Join Tables in DynamoDB Using Amazon EMR (p. 97)
- Amazon Kinesis (p. 111)
- Distributed Copy Using S3DistCp (p. 113)

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.

Release Information

EMR File System (EMRFS) 2.0.0

Amazon EMR Release Label: emr-4.0.0

Topics

- Consistent View (p. 75)

- Creating an AWSCredentialsProvider for EMRFS (p. 89)
- Encryption in EMRFS (p. 90)

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 the US East (N. Virginia) provide eventual consistency on read-after-write and read-after-overwrite requests. Amazon S3 buckets in the US West (Oregon), US West (N. California), EU (Ireland), EU (Frankfurt), Asia Pacific (Singapore), Asia Pacific (Tokyo), Asia Pacific (Sydney), and South America (Sao Paulo)regions provide read-after-write consistency for put requests of new objects and eventual consistency for overwrite 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. 83).

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 (p. 76)
- Objects Tracked By EMRFS (p. 76)
- Retry Logic (p. 77)
- EMRFS Metadata (p. 77)
- Configuring Consistency Notifications for CloudWatch and Amazon SQS (p. 79)
- Configuring Consistent View (p. 81)
- EMRFS CLI Reference (p. 83)

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 you can use a bootstrap action to configure additional settings for EMRFS.

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 http://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-4.0.0 --ec2-attributes KeyName=myKey
```

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. 83). For more information about purging listings in the EMRFS metadata, see the section called “EMRFS Metadata” (p. 77).

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-4.0.0 --instance-type m3.xlarge -
-instance-count 1 --emrfs Consistent=true,Args=[fs.s3.consistent.throwExcep
tionOnInconsistency=false, fs.s3.consistent.retryPolicyType=fixed,fs.s3.consist
ent.retryPeriodSeconds=10] --ec2-attributes KeyName=myKey
```

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. 83).

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. 83).

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 500 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 500 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

Operation	Average read-per-second	Average write-per-second
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
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="com
mand-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-4.0.0 --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-4.0.0 --instance-type m3.xlarge --
instance-count 2 --applications Name=Hive --configurations file://./myConfig.json
```

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.

Property	Default value	Description
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.
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	**500**	This property specifies the DynamoDB read capacity to provision when the metadata table is created.
fs.s3.consistent.metadata.write.capacity	**250**	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.

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: 500
  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 `500`.	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: 500
  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
  [no output]
```

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 `500`.	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: 500
  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

Option	Description	Required
[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`.	

Option	Description	Required
`--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 `500`.	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 `500`.	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 `500`.	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. You must specify the full class name of the provider by setting `fs.s3.customAWSCredentialsProvider` in `/home/hadoop/conf/emrfs-site.xml`. 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-4.0.0 --instance-type m3.xlarge --
instance-count 2 --applications Name=Hive --configurations file://./myConfig.json
```

Additionally, you will need to place the JAR file of the AWSCredentialsProvider class in `/usr/share/aws/emr/auxlib`. An example implementation follows:

```
public class MyAWSCredentialsProvider implements AWSCredentialsProvider, Config
urable {
    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;
    }
  }
```

Encryption in EMRFS

You can use either server-side or client-side encryption to protect the data you store in Amazon S3. With server-side encryption, Amazon S3 encrypts your data after you upload it. With client-side encryption, you manage the encryption and keys. You can use AWS Key Management Service (AWS KMS) to manage your keys used for encryption. You enable both of these options at cluster creation time.

Note
Client-side encryption and server-side encryption are mutually exclusive global settings. When either option is enabled, all Amazon S3 write actions that happen through EMRFS use the form of encryption chosen.
EMRFS implements Amazon S3 encryption. If you write files locally on your cluster (in HDFS or local file systems volumes), those files are not encrypted.

Create a Cluster With Amazon S3 Server-Side Encryption Enabled

Amazon S3 server-side encryption (SSE) is supported with Amazon EMR on AMIs 3.2.1 or later. To launch a cluster with server-side encryption, you can use the AWS Management Console, AWS CLI, or the `configure-hadoop` bootstrap action to set `fs.s3.enableServerSideEncryption` to "true". You can specify the encryption key and/or algorithm used. However, if you provide your own key, you must have access to that same key to later access objects stored and encrypted.

To configure server-side encryption 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 launch a cluster with Amazon S3 server-side encryption enabled

Type the following command to launch an Amazon EMR cluster with Amazon S3 server-side encryption enabled.

```
aws emr create-cluster --release-label emr-4.0.0 --instance-count 3 --instance-
type m1.large --emrfs Encryption=ServerSide
```

To launch a cluster with Amazon S3 server side encryption enabled that specifies an encryption algorithm

Type the following command to launch an Amazon EMR cluster with Amazon S3 server-side encryption enabled that specifies AES256 as the encryption algorithm.

```
aws emr create-cluster --instance-type m3.xlarge --release-label emr-4.0.0 --
emrfs Encryption=ServerSide,Args=[fs.s3.serverSideEncryptionAlgorithm=AES256]
```

`emrfs-site.xml` properties for server-side encryption

Property	Default value	Description
`fs.s3.enableServerSideEncryption`	**`false`**	When set to **`true`**, objects stored in Amazon S3 are encrypted using server-side encryption.
`fs.s3.serverSideEncryptionAlgorithm`	**AES256**	When using server-side encryption, this property determines the algorithm used to encrypt data.

Using Amazon S3 Client-Side Encryption in EMRFS

EMRFS support for Amazon S3 client-side encryption enables your EMR cluster to work with S3 objects that were previously encrypted using an Amazon S3 encryption client. When Amazon S3 client-side encryption is enabled, EMRFS supports the decryption of objects encrypted using keys in AWS KMS or from your own key management system. Amazon S3 client-side encryption in EMRFS also supports re-encrypting the output from your EMR cluster using keys from either AWS KMS or your own key management system.

> **Note**
> EMRFS client-side encryption only ensures that output written from an enabled cluster to Amazon S3 will be encrypted. Data written to the local file systems and HDFS on the cluster are not encrypted. Furthermore, because Hue does not use EMRFS, objects written to Amazon S3 using the Hue S3 File Browser are not encrypted. For more information about security controls available for applications running on EC2 instances, see the "Overview of Security Processes" whitepaper.

EMRFS support for Amazon S3 client-side encryption uses a process called *envelope encryption*, with keys stored in a location of your choosing, to encrypt and decrypt data stored in Amazon S3. In contrast to Amazon S3 server-side encryption, the decryption and encryption actions in Amazon S3 client-side encryption take place in the EMRFS client on your EMR cluster; the encrypted object streams from Amazon S3 to your EMR cluster in an encrypted form to be decrypted by the client on the cluster. Output from the cluster is then encrypted by the client before being written to Amazon S3.

The envelope encryption process uses a one-time symmetric data key generated by the encryption client, unique to each object, to encrypt data. The data key is then encrypted by your master key (stored in AWS KMS or your custom provider) and stored with the associated object in Amazon S3. When decrypting data on the client (e.g., an EMRFS client or your own Amazon S3 encryption client retrieving data for post-processing), the reverse process occurs: the encrypted data key is retrieved from the metadata of the object in Amazon S3. It is decrypted using the master key and then the client uses the data key to decrypt the object data. When Amazon S3 client-side encryption is enabled, the EMRFS client on the cluster can read either encrypted or unencrypted objects in Amazon S3.

When Amazon S3 client-side encryption in EMRFS is enabled, the behavior of the encryption client depends on the provider specified and the metadata of the object being decrypted or encrypted. When EMRFS encrypts an object before writing it to Amazon S3, the provider (e.g., AWS KMS or your custom provider) that you specified at cluster creation time is always used to supply the encryption key. When EMRFS reads an object from Amazon S3, it checks the object metadata for information about the master key used to encrypt the data key. If there is an AWS KMS key ID, EMRFS attempts to decrypt the object using AWS KMS. If there is metadata containing an EncryptionMaterialsDescription instance, EMRFS tries to fetch the key using the EncryptionMaterialsProvider instance. The provider uses this description to determine which key should be used and to retrieve it. If you do not have access to the required key, this raises an exception and causes an error. If there is no EncryptionMaterialsDescription instance in the Amazon S3 object metadata, EMRFS assumes that the object is unencrypted.

Amazon S3 client-side encryption in EMRFS provides two methods to supply the master keys for decryption when reading from Amazon S3 and encryption when writing to Amazon S3:

1. With a built-in AWS KMS provider, which can use a master key stored in AWS KMS. You specify the key to use for encryption, but EMRFS can use any AWS KMS key for decryption, assuming your cluster has permission to access it. AWS KMS charges apply for the storage and use of encryption keys.
2. With a custom Java class implementing both the Amazon S3 EncryptionMaterialsProvider and Hadoop Configurable classes. The EncryptionMaterialsProvider class is used to provide the materials description, detailing how and where to get the master keys.

For more information about Amazon S3 client-side encryption see, Protecting Data Using Client-Side Encryption. For more information about how to use the AWS SDK for Java with Amazon S3 client-side encryption, see the article Client-Side Data Encryption with the AWS SDK for Java and Amazon S3.

For information about how to create and manage keys in AWS KMS and associated pricing, see AWS KMS Frequently Asked Questions and the AWS Key Management Service Developer Guide.

Topics

- Enabling Amazon S3 Client-Side Encryption in the Console (p. 93)
- Selecting a Master Key Stored in AWS KMS using an SDK or CLI (p. 93)
- Configuring Amazon S3 Client-side Encryption Using a Custom Provider (p. 94)
- emrfs-site.xml Properties for Amazon S3 Client-side Encryption (p. 96)

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 a 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 a 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 in EMRFS 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 encryption with EMRFS, 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-4.0.0 --emrfs Encryption=Client
Side,ProviderType=KMS,KMSKeyId=MyKMSKeyId
```

Note

Note: You must use the ARN of the AWS KMS master key if you want 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-4.0.0 --
emrfs Encryption=ClientSide,ProviderType=Custom,CustomProviderLocation=s3://my
bucket/myfolder/provider.jar,CustomProviderClass=classname
```

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-4.0.0 --instance-type m3.xlarge --
instance-count 2 --configurations file://./myConfig.json --emrfs Encryption=Cli
entSide,CustomProviderLocation=s3://mybucket/myfolder/myprovider.jar,CustomPro
viderClass=classname
```

To use an SDK, you can set the property `fs.s3.cse.encryptionMaterialsProvider.uri` to download the custom EncryptionMaterialsProvider class 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://my
bucket/MyCustomEncryptionMaterialsProvider.jar");
      emrfsProperties.put("fs.s3.cse.enabled","true");
      emrfsProperties.put("fs.s3.consistent","true");
      emrfsProperties.put("fs.s3.cse.encryptionMaterialsPro
vider","full.class.name.of.EncryptionMaterialsProvider");

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

  RunJobFlowRequest request = new RunJobFlowRequest()
   .withName("Custom EncryptionMaterialsProvider")
   .withReleaseLabel("emr-4.0.0")
   .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 the section called "`emrfs-site.xml` Properties for Amazon S3 Client-side Encryption" (p. ?).

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 EncryptionMaterialsPro
vider, 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> materi
alsDescription) {
    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`	**`N/A`**	The EncryptionMaterialsProvider class path used with client-side encryption.

Property	Default value	Description
`fs.s3.cse.materialsDescription.enabled`	**`false`**	Enabling will populate the materialsDescription of encrypted objects with the Amazon S3 URI for the object and the JobFlowId.
`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.

Export, Import, Query, and Join Tables in DynamoDB Using Amazon EMR

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. 48)

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. 110) or Write Percent Setting (p. 110) to a lower value to throttle the Amazon EMR operations. For information about DynamoDB throughput settings, see Provisioned Throughput.

Topics

- Set Up a Hive Table to Run Hive Commands (p. 98)
- Hive Command Examples for Exporting, Importing, and Querying Data in DynamoDB (p. 102)
- Optimizing Performance for Amazon EMR Operations in DynamoDB (p. 109)

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_attrib
ute1_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 ar
ray<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. 102).

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 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.

Hive Options	Description
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 on creating and deleting tables in DynamoDB, go to Working with Tables in DynamoDB.

> **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 ar
ray<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 ar
ray<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 ar
ray<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 ar
ray<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 ar
ray<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 hiveTable
Name;
```

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 ar
ray<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 ar
ray<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. 25). 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 ar
ray<string>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';

CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 ar
ray<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 ar
ray<string>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 'hdfs:///directoryName';

CREATE EXTERNAL TABLE hiveTableName (col1 string, col2 bigint, col3 ar
ray<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_pur
chased: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_pur
chased: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_pur
chased: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,custom
erAddress: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_pur
chased: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. 101).

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. 101).

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. 101).

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. 25). 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 details, 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.check-point.metastore.hash.range.name	Range key name for the DynamoDB table
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*.

Distributed Copy Using S3DistCp

Topics

- S3DistCp Options (p. 114)
- Adding S3DistCp as a Step in a Cluster (p. 118)

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 expression or the entire `--args` string must be enclosed in single quotes ('). Example: `--srcPattern,.*daemons.*-hadoop-.*`	No

Option	Description	Required
`--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. 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
`--appendToLastFile`	Specifies the behavior of S3DistCp when copying to files 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

Option	Description	Required
`--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
`--startingIndex,INDEX`	Used with `--numberFiles` to specify the first number in the sequence. Example: `--startingIndex,1`	No

Option	Description	Required
`--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. 114).

  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","--srcPat
```

```
tern .*[a-zA-Z,]+"],
        "ActionOnFailure":"CONTINUE",
        "Type":"CUSTOM_JAR",
        "Jar":"command-runner.jar"
    }
]
```

```
aws emr add-steps --cluster-id j-3GYXXXXXX9IOK --steps --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://mybuck
et/logs/j-3GYXXXXXX9IOJ/node/","--dest hdfs:///output","--srcPattern .*daemons.*-
hadoop-.*"],
        "ActionOnFailure":"CONTINUE",
        "Type":"CUSTOM_JAR",
        "Jar":"command-runner.jar"
    }
]
```

Document History

The following table describes the important changes to the documentation since the last release of Amazon Elastic MapReduce (Amazon EMR).

API version: 2009-03-31

Latest documentation update: July 23, 2015

Change	Description	Release Date
Amazon EMR Release 4.0.0	Initial release of this guide for the emr-4.0.0 release.	July 23, 2015