

Cloudera Runtime 7.1.7

Configuring Apache Spark

Date published: 2021-02-29

Date modified: 2022-01-19

CLOUdera

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Configuring dynamic resource allocation.....	4
Customize dynamic resource allocation settings.....	4
Configure a Spark job for dynamic resource allocation.....	4
Dynamic resource allocation properties.....	5
 Spark security.....	 6
Enabling Spark authentication.....	6
Enabling Spark Encryption.....	6
Running Spark applications on secure clusters.....	7
Configuring HSTS for Spark.....	7
 Accessing compressed files in Spark.....	 8
 Sample script to connect Spark to Ozone.....	 8

Configuring dynamic resource allocation

This section describes how to configure dynamic resource allocation for Apache Spark.

When the dynamic resource allocation feature is enabled, an application's use of executors is dynamically adjusted based on workload. This means that an application can relinquish resources when the resources are no longer needed, and request them later when there is more demand. This feature is particularly useful if multiple applications share resources in your Spark cluster.



Important: Dynamic resource allocation does not work with Spark Streaming.

You can configure dynamic resource allocation at either the cluster or the job level:

- Cluster level: Dynamic resource allocation is enabled by default. The associated shuffle service starts automatically.
- Job level: You can customize dynamic resource allocation settings on a per-job basis. Job settings override cluster configuration settings.

Cluster configuration is the default, unless overridden by job configuration.

The following subsections describe each configuration approach, followed by a list of dynamic resource allocation properties.

Customize dynamic resource allocation settings

Use the following steps to review and customize dynamic resource allocation settings.

About this task

Dynamic resource allocation requires an external shuffle service that runs on each worker node as an auxiliary service of NodeManager. This service is started automatically; no further steps are needed.

Dynamic resource allocation is enabled by default. To modify dynamic allocation settings, use the following procedure.

Procedure

1. From the Cloudera Data Platform (CDP) interface, click through to Cloudera Manager for the cluster running Spark.
2. Go to the Spark service page (ClustersSpark service).
3. Click on the Configuration tab.
4. In the Search box, enter dynamicAllocation.
5. After making the changes you want, enter the reason for the change in the Reason for change... box, and then click Save Changes.
6. Restart the Spark service (Spark serviceActionsRestart).

Configure a Spark job for dynamic resource allocation

Use the following steps to configure dynamic resource allocation for a specific job.

There are two ways to customize dynamic resource allocation properties for a specific job:

- Include property values in the spark-submit command, using the -conf option.

This approach loads the default spark-defaults.conf file first, and then applies property values specified in your spark-submit command.

Example:

```
spark-submit --conf "property_name=property_value"
```

- Create a job-specific spark-defaults.conf file and pass it to the spark-submit command.

This approach uses the specified properties file, without reading the default property file.

Example:

```
spark-submit --properties-file <property_file>
```

Dynamic resource allocation properties

The following tables provide more information about dynamic resource allocation properties. These properties can be accessed by clicking through to Cloudera Manager from the Cloudera Data Platform interface for the cluster running Spark.

Table 1: Dynamic Resource Allocation Properties

Property Name	Default Value	Description
spark.dynamicAllocation.enabled	false for Spark jobs	Enable dynamic allocation of executors in Spark applications.
spark.shuffle.service.enabled	true	Enables the external shuffle service. The external shuffle service preserves shuffle files written by executors so that the executors can be deallocated without losing work. Must be enabled if dynamic allocation is enabled.
spark.dynamicAllocation.initialExecutors	Same value as spark.dynamicAllocation.minExecutors	When dynamic allocation is enabled, number of executors to allocate when the application starts. Must be greater than or equal to the minExecutors value, and less than or equal to the maxExecutors value.
spark.dynamicAllocation.maxExecutors	infinity	When dynamic allocation is enabled, maximum number of executors to allocate. By default, Spark relies on YARN to control the maximum number of executors for the application.
spark.dynamicAllocation.minExecutors	0	When dynamic allocation is enabled, minimum number of executors to keep alive while the application is running.
spark.dynamicAllocation.executorIdleTimeout	60 seconds (60s)	When dynamic allocation is enabled, time after which idle executors will be stopped.
spark.dynamicAllocation.cachedExecutorIdleTimeout	infinity	When dynamic allocation is enabled, time after which idle executors with cached RDD blocks will be stopped.
spark.dynamicAllocation.schedulerBacklogTimeout	1 second (1s)	When dynamic allocation is enabled, timeout before requesting new executors when there are backlogged tasks.
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	Same value as schedulerBacklogTimeout	When dynamic allocation is enabled, timeout before requesting new executors after the initial backlog timeout has already expired. By default this is the same value as the initial backlog timeout.

Related Information

[Apache Dynamic Resource Allocation](#)

Spark security

When you create an environment in the Cloudera Data Platform (CDP) Management Console, it automatically creates a Kerberos- and TLS-enabled data lake cluster. Data hub clusters are linked to environments, and therefore also have Kerberos enabled by default. You do not need to do anything to enable Kerberos or TLS for Apache Spark in CDP. Disabling security is not supported.

To submit Spark jobs to the cluster, users must authenticate with their Kerberos credentials. For more information, see the [Environments](#) documentation.

Enabling Spark authentication

Spark authentication here refers to an internal authentication mechanism, and not to Kerberos authentication, which is enabled automatically for all Cloudera Data Platform deployments.

Before you begin

Minimum Required Role: Security Administrator (also provided by Full Administrator)

About this task

Spark has an internal mechanism that authenticates executors with the driver controlling a given application. This mechanism is enabled using the Cloudera Manager Admin Console, as detailed below. Cluster administrators can enable the spark.authenticate mechanism to authenticate the various processes that support a Spark application.

To enable this feature on the cluster:

Procedure

1. In the Cloudera Data Platform (CDP) Management Console, go to Data Hub Clusters.
2. Find and select the cluster you want to configure.
3. Click the link for the Cloudera Manager URL.
4. Go to Clusters <Cluster Name>Spark serviceConfiguration .
5. Scroll down to the Spark Authentication setting, or search for spark.authenticate to find it.
6. In the Spark Authentication setting, click the checkbox next to the Spark (Service-Wide) property to activate the setting.
7. Enter the reason for the change at the bottom of the screen, and then click Save Changes.
8. Restart YARN:
 - a) Select Clusters YARN .
 - b) Select Restart from the Actions drop-down selector.
9. Re-deploy the client configurations:
 - a) Select Clusters *Cluster_name*
 - b) Select Deploy Client Configurations from the Actions drop-down selector.
10. Restart stale services.

Enabling Spark Encryption

Before you begin

Before enabling encryption, you must first enable [Spark authentication](#).

Procedure

1. In the Cloudera Data Platform (CDP) Management Console, go to Data Hub Clusters.
2. Find and select the cluster you want to configure.
3. Click the link for the Cloudera Manager URL.
4. Go to Clusters <Cluster Name>Spark serviceConfiguration .
5. Search for the Enable Network Encryption property. Use the checkbox to enable encrypted communication between Spark processes belonging to the same application.
6. Search for the Enable I/O Encryption property. Use the checkbox to enable encryption for temporary shuffle and cache files stored by Spark on local disks.
7. Enter the reason for the change at the bottom of the screen, and then click Save Changes.
8. Re-deploy the client configurations:
 - a) Select Clusters *Cluster_name*
 - b) Select Deploy Client Configurations from the Actions drop-down selector.
9. Restart stale services.

Running Spark applications on secure clusters

All CDP clusters are secure by default. Disabling security on CDP clusters is not supported. To run a Spark application on a secure cluster, you must first authenticate using Kerberos.

Users running Spark applications must first authenticate to Kerberos, using kinit, as follows:

```
kinit username@EXAMPLE.COM
```

After authenticating to Kerberos, users can submit their applications using spark-submit as usual, as shown below. This command submits one of the default Spark sample jobs using an environment variable as part of the path, so modify as needed for your own use:

```
$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn \
--deploy-mode cluster $SPARK_HOME/lib/spark-examples.jar 10
```

For information on creating user accounts in CDP, see [Onboarding Users](#).

Configuring HSTS for Spark

You can configure Apache Spark to include HTTP headers to prevent Cross-Site Scripting (XSS), Cross-Frame Scripting (XFS), MIME-Sniffing, and also enforce HTTP Strict Transport Security (HSTS).

Procedure

1. Go to the Spark service.
2. Click the Configuration tab.
3. Select History Server under Scope.
4. Select Advanced under Category.
5. Set the following HSTS credentials in History Server Advanced Configuration Snippet (Safety Valve) for spark-conf/spark-history-server.conf.

```
spark.ui.strictTransportSecurity=max-age=31536000;includeSubDomains
```

6. Restart the Spark service.

Accessing compressed files in Spark

You can read compressed files using one of the following methods:

- `textFile(path)`
- `hadoopFile(path,outputFormatClass)`

You can save compressed files using one of the following methods:

- `saveAsTextFile(path, compressionCodecClass="codec_class")`
- `saveAsHadoopFile(path,outputFormatClass, compressionCodecClass="codec_class")`

where `codec_class` is one of these classes:

- `gzip` - `org.apache.hadoop.io.compress.GzipCodec`
- `bzip2` - `org.apache.hadoop.io.compress.BZip2Codec`
- `LZO` - `com.hadoop.compression.lzo.LzopCodec`
- `Snappy` - `org.apache.hadoop.io.compress.SnappyCodec`
- `Deflate` - `org.apache.hadoop.io.compress.DeflateCodec`

For examples of accessing Avro and Parquet files, see [Spark with Avro and Parquet](#).

Sample script to connect Spark to Ozone

Learn how you can connect Spark to the Ozone object store with the help of a sample script.

This script, in Scala, counts the number of word occurrences in a text file. The key point in this example is to use the following string to refer to the text file: `o3fs://hivetest.s3v.o3service1/spark/jedi_wisdom.txt`

Word counting example in Scala

```
import sys.process._

// Put the input file into Ozone
// "hdfs dfs -put data/jedi_wisdom.txt o3fs://hivetest.s3v.o3service1/spark
" !
// Set the following spark setting in the file "spark-defaults.conf" on the
CML session using terminal
// spark.yarn.access.hadoopFileSystems=o3fs://hivetest.s3v.o3service1.neptune
e01.olympus.cloudera.com:9862

//count lower bound
val threshold = 2
// this file must already exist in hdfs, add a
// local version by dropping into the terminal.
val tokenized = sc.textFile("o3fs://hivetest.s3v.o3service1/spark/jedi_wisdo
m.txt").flatMap(_.split(" "))
// count the occurrence of each word
val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
// filter out words with fewer than threshold occurrences
val filtered = wordCounts.filter(_._2 >= threshold)
System.out.println(filtered.collect().mkString(", "))
```