Cloudera Runtime 7.3.3

Upgrading Apache Spark

Date published: 2020-07-28
Date modified: 2026-12-13



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. 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

Upgrading Spark 2 to Spark 3 for Cloudera on cloud 7.3.2 4
Upgrading from 7.2.18 5
Upgrading Apache Spark 2.4.8 on 7.2.18 SP2 to Spark 3 on 7.3.1 5
Upgrades with Spark 3.4.1 (bundled) 7
Upgrading Apache Spark 2.4.8 (with 3.4.1 bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1 7
Upgrading Apache Spark 3.4.1 (bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1 9
Upgrading from 7.2.17 10
Upgrading Apache Spark 2.4.8 on 7.2.17 to Spark 3 on 7.3.1 10
Upgrading Apache Spark 2.4.8 on 7.2.17 to Spark 3 on 7.3.1 12
Upgrading Apache Spark 2.4.8 (with 3.3.2 bundled) on 7.2.17 to Spark 3 on 7.3.1 13
Upgrading Apache Spark 2.4.8 (with 3.3.2 bundled) on 7.2.17 to Spark 3 on 7.3.1 15
Upgrading Apache Spark 3.3.2 (bundled) on 7.2.17 to Spark 3 on 7.3.1 15
Upgrading Apache Spark 3.5.4 to Spark 3.5.8 16

Migrating Spark applications 17
Java versions 18
Scala versions 18
Python versions 18
Spark commands 19
Spark connectors 19
Logging 20
Third-party libraries 20
Spark behavior changes 20
Apache Spark Migration guides 20
Spark 2 to Spark 3 workload refactoring 21
Unsupported features 24
Post-migration checklist 25
Benchmark testing 25
Troubleshooting 25

Upgrading Spark 2 to Spark 3 for Cloudera on cloud 7.3.2

Upgrading Apache Spark from version 2 to version 3 in Cloudera Cloudera on cloud is a process that involves:

1. Intermediate in-place cluster upgrade tasks, due to different support for Spark 3 versions of connectors in Cloudera versions.
2. Intermediate Spark application migration tasks, due to minor or maintenance Spark version changes (including maintenance rebases such as Spark 3.5.4 to 3.5.8).
3. Sidecar migration tasks for Data Hub clusters, because adjusting existing Data Hub clusters is not possible, but in-place 7.3.2 upgrade can only happen on clusters where Spark 2 is no longer present.
4. Application migration from Spark 2 to Spark 3, due to major Spark version changes.
5. Post-application migration tasks.
6. In-place cluster upgrade tasks.

   Important: Cluster upgrade to version 7.3.2 is only allowed if all Spark 2 applications have been migrated to Spark 3 and tested in the **source cluster**.

7. Spark application migration tasks, due to minor or maintenance Spark version changes (recompile and retest applications; see Migrating Spark applications).

Warning:

Upgrading from Spark 2 to Spark 3 in Cloudera Runtime 7.3.2 is only supported for the following service packs:

- 7.2.17.200
- 7.2.17.300
- 7.2.17.400
- 7.2.17.500
- 7.2.18.0
- 7.2.18.100
- 7.2.18.200

If your Cloudera Runtime version is not in the list of supported versions for a direct upgrade, it is possible that you can still upgrade to one of the above versions first, and then upgrade to Cloudera Runtime 7.3.2.

For more information on upgrading Cloudera Runtime 7.2.17.x and the supported upgrade paths, see Upgrading to Cloudera Runtime 7.3.2

Note: The application migration tasks (see *Migrating Spark Applications*) are done on your application codebase.

Note: Depending on your environment and Spark applications, some of these steps are not necessary.

This table below provides links to the upgrade guides based on the version of Cloudera on cloud you're using, and the source versions of Spark in your environment.

Important:

If you're using Spark 2 with any of the following connectors, follow the upgrade steps in the relevant *Upgrade guide (with connectors)* pages:

- Oozie
- Solr
- Phoenix
- Hive Warehouse Connector
- Spark Schema Registry

Each upgrade guide contain all steps needed to upgrade Spark 2 to Spark 3 and perform the upgrade to Cloudera on cloud version 7.3.2.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Data Hub template	
7.2.18 SP2	2.4.8	None	Data Hub was created with a custom template.	Upgrade guide
	2.4.8	3.4.1 (bundled)		Upgrade guide
	None	3.4.1 (bundled)	Data Hub was created with the *7.2.18 - Data Engineering: Apache Spark3, Apache Hive, Apache Oozie* or a custom template.	Upgrade guide
7.2.17	2.4.8	None	Data Hub was created with the *7.2.17 - Data Engineering: Apache Spark, Apache Hive, Apache Oozie* or a custom template.	Upgrade guide Upgrade (with connectors) guide
	2.4.8	3.3.2 (bundled)	Data Hub was created with a custom template.	Upgrade guide Upgrade guide (with connectors)
	None	3.3.2 (bundled)	Data Hub was created with the *7.2.17 - Data Engineering: Apache Spark3, Apache Hive, Apache Oozie* or a custom template.	Upgrade guide

Spark 3 maintenance upgrade to Apache Spark 3.5.8

Apache Spark **3.5.8** is included in Cloudera Runtime **7.3.2.10000 SP1** and higher on the 7.3.2 line. Use this guide when you already run Spark 3.5.4 on 7.3.2.0 and apply the service pack that rebases Spark to 3.5.8.

Source Spark 3 version	Target Cloudera Runtime (Spark 3.5.8)	Documentation
3.5.4 (bundled)	7.3.2.10000 SP1 and higher	Upgrade guide

Related Information

Migrating Spark applications

Upgrading from 7.2.18

Upgrading Apache Spark 2.4.8 on 7.2.18 SP2 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 on Cloudera on cloud 7.2.18 SP2 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[1]
7.2.18 SP2	2.4.8	none	**7.3.1**	**3.4.1**	no

Sidecar migration of Data Hub clusters

Procedure

Sidecar migration tasks for Data Hub clusters

The new 7.2.18 Data Hub cluster needs to use Spark 3 and Livy 3 instead of Spark 2 and Livy 2.

Depending on the template you used for your existing Data Hub clusters, a new custom template might be needed that contains Spark 3 instead of Spark 2. Alternatively, the built-in *7.2.18 - Data Engineering: Apache Spark3, Apache Hive, Apache Oozie* template can be used, as it contains Spark 3 only.

1. Check the current services in your template, and add the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template.
2. If the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template doesn't work, you can create a custom template. Replace all Spark 2 and Livy 2 references with Spark 3 and Livy 3, respectively.
3. Add a new Spark 3-based 7.2.18 cluster to the environment, using your custom template or the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template.
4. Migrate all non-spark workloads from the old cluster to the new cluster.

Application migration tasks (Spark 2 to 3)

Procedure

1. Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 2.4.8 to 3.4.1
   a) Check the supported Java versions.
   b) Check the supported Scala version.
   c) Check the supported Python versions.
   d) Account for changed or versioned Spark commands in your code. (spark-submit, pyspark, etc.)
   e) Check supported versions for Spark connectors.
   f) Check the logging library used in your code.
   g) Check the compatibility of 3rd-party libraries used in your code.
   h) Check Spark behavior changes and refactor your code.
2. Migrate all Spark 2 applications in the old cluster to Spark 3 applications in the new cluster.

Post-application migration tasks

Procedure

1. Move Spark 2 event logs to the Spark 3 event logs directory.
2. Drop the old cluster.

In-place cluster upgrade

Before you begin

Important: Cluster upgrade to version 7.3.1 is only allowed if all Spark 2 applications have been migrated to Spark 3 and tested in the **source cluster**.

Procedure

1. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.
   c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
   d) Click the Target Version drop-down menu to see any available upgrades.
   e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.
   f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
   g) Click Upgrade to initiate the upgrade.
   h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

   For more information, see Data Lake upgrade.
2. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrades with Spark 3.4.1 (bundled)

Guides for cluster versions that ship Apache Spark 3.4.1 (bundled) while you move toward Spark 3 on Cloudera Runtime 7.3.x.

Use the guide that matches how Spark 2 and Spark 3 are deployed in your environment. These paths apply whether you run Cloudera on premises or Cloudera on cloud; follow the documentation link for your deployment when the guide references a specific source cluster version.

Upgrading Apache Spark 2.4.8 (with 3.4.1 bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 (with 3.4.1 bundled) on Cloudera on cloud 7.2.18 SP2 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[2]
7.2.18 SP2	2.4.8	3.4.1 (bundled)	**7.3.1**	**3.4.1**	no

Sidecar migration of Data Hub clusters

Procedure

Sidecar migration tasks for Data Hub clusters

The new 7.2.18 Data Hub cluster needs to use Spark 3 and Livy 3 instead of Spark 2 and Livy 2.

Depending on the template you used for your existing Data Hub clusters, a new custom template might be needed that contains Spark 3 instead of Spark 2. Alternatively, the built-in *7.2.18 - Data Engineering: Apache Spark3, Apache Hive, Apache Oozie* template can be used, as it contains Spark 3 only.

1. Check the current services in your template, and add the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template.
2. If the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template doesn't work, you can create a custom template. Replace all Spark 2 and Livy 2 references with Spark 3 and Livy 3, respectively.
3. Add a new Spark 3-based 7.2.18 cluster to the environment, using your custom template or the built-in *7.2.18 - : Apache Spark3, Apache Hive, Apache Oozie* template.
4. Migrate all non-spark workloads from the old cluster to the new cluster.

Application migration tasks (Spark 2 to 3)

Procedure

1. Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 2.4.8 to 3.4.1
   a) Check the supported Java versions.
   b) Check the supported Scala version.
   c) Check the supported Python versions.
   d) Account for changed or versioned Spark commands in your code. (spark-submit, pyspark, etc.)
   e) Check supported versions for Spark connectors.
   f) Check the logging library used in your code.
   g) Check the compatibility of 3rd-party libraries used in your code.
   h) Check Spark behavior changes and refactor your code.
2. Migrate all Spark 2 applications in the old cluster to Spark 3 applications in the new cluster.

Post-application migration tasks

Procedure

1. Move Spark 2 event logs to the Spark 3 event logs directory.
2. Drop the old cluster.

In-place cluster upgrade

Before you begin



Important: Cluster upgrade to version 7.3.1 is only allowed if all Spark 2 applications have been migrated to Spark 3 and tested in the **source cluster**.

Procedure

1. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.
   c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
   d) Click the Target Version drop-down menu to see any available upgrades.
   e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.
   f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
   g) Click Upgrade to initiate the upgrade.
   h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

   For more information, see Data Lake upgrade.
2. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrading Apache Spark 3.4.1 (bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 (with 3.4.1 bundled) on Cloudera on premises 7.2.18 SP2 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[3]
7.2.18 SP2	none	3.4.1 (bundled)	**7.3.1**	**3.4.1**	no

In-place cluster upgrade

Procedure

1. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.

c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
d) Click the Target Version drop-down menu to see any available upgrades.
e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.
f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
g) Click Upgrade to initiate the upgrade.
h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

For more information, see Data Lake upgrade.

2. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrading from 7.2.17

Upgrading Apache Spark 2.4.8 on 7.2.17 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 on Cloudera on cloud 7.2.17 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[4]
7.2.17	2.4.8	none	**7.3.1**	**3.4.1**	no

Intermediate in-place cluster upgrade

Procedure

Upgrade the cluster OS from Centos 7 to RedHat 8.

Sidecar migration of Data Hub clusters

Procedure

Sidecar migration tasks for Data Hub clusters

The new 7.2.17 Data Hub cluster needs to use Spark 3 and Livy 3 instead of Spark 2 and Livy 2.

Depending on the template you used for your existing Data Hub clusters, a new custom template might be needed that contains Spark 3 instead of Spark 2. Alternatively, the built-in *7.2.17 - Data Engineering: Apache Spark3* template can be used, as it contains Spark 3 only.

1. Check the current services in your template, and add the built-in *7.2.17 - : Apache Spark3* template.
2. If the built-in *7.2.17 - : Apache Spark3* template doesn't work, you can create a custom template. Replace Spark 2 and Livy 2 references with Spark 3 and Livy 3, respectively..
3. Add a new Spark 3-based 7.2.17 cluster to the environment, using your custom template or the built-in *7.2.17 - : Apache Spark3* template.
4. Migrate all non-spark workloads from the old cluster to the new cluster.

Application migration (Spark 2 to 3)

Procedure

1. Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 2.4.8 to 3.3.2.
   a) Check the supported Java versions.
   b) Check the supported Scala version.
   c) Check the supported Python versions.
   d) Account for changed or versioned Spark commands in your code. (spark-submit, pyspark, etc.)
   e) Check supported versions for Spark connectors.
   f) Check the logging library used in your code.
   g) Check the compatibility of 3rd-party libraries used in your code.
   h) Check Spark behavior changes and refactor your code.
2. Migrate all Spark 2 applications in the old cluster to Spark 3 applications in the new cluster.

Post-application migration tasks

Procedure

1. Move Spark 2 event logs to the Spark 3 event logs directory.
2. Drop the old cluster.

In-place cluster upgrade

Before you begin

Important: Cluster upgrade to version 7.3.1 is only allowed if all Spark 2 applications have been migrated to Spark 3 and tested in the **source cluster**.

Procedure

1. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.
   c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
   d) Click the Target Version drop-down menu to see any available upgrades.
   e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.

f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
g) Click Upgrade to initiate the upgrade.
h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

For more information, see Data Lake upgrade.

2. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Application migration tasks (Spark 3.x to 3.4.1)

Procedure

Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 3.3.2 to 3.4.1

a) Refactor your Spark application code.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrading Apache Spark 2.4.8 on 7.2.17 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 on Cloudera on cloud 7.2.17 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[5]
7.2.17	2.4.8	none	**7.3.1**	**3.4.1**	yes

Intermediate in-place cluster upgrade

Procedure

1. Upgrade the cluster OS from Centos 7 to RedHat 8.
2. Upgrade your cluster to 7.2.18 SP2.
   a) Identify cluster version details.
   b) Identify your upgrade path.
   c) Review the prerequisites

d) High-level upgrade steps.

For more information on upgrading your cluster to 7.2.18 SP2, see:

- Upgrading to 7.2.18

3. Once complete, continue the upgrade with the steps described in Upgrading Apache Spark 2.4.8 (with 3.4.1 bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1.

Upgrading Apache Spark 2.4.8 (with 3.3.2 bundled) on 7.2.17 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 (with 3.3.2 bundled) on Cloudera Public Cloud 7.2.17 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[6]
7.2.17	2.4.8	3.3.2 (bundled)	**7.3.1**	**3.4.1**	no

Intermediate in-place cluster upgrade

Procedure

Upgrade the cluster OS from Centos 7 to RedHat 8.

Intermediate application migration tasks

Procedure

Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 3.3.2 to 3.4.1

a) Refactor your Spark application code.

Sidecar migration of Data Hub clusters

Procedure

The new 7.2.17 Data Hub cluster needs to use Spark 3 and Livy 3 instead of Spark 2 and Livy 2.

Depending on the template you used for your existing Data Hub clusters, a new custom template might be needed that contains Spark 3 instead of Spark 2. Alternatively, the built-in *7.2.17 - Data Engineering: Apache Spark3* template can be used, as it contains Spark 3 only.

1. Check the current services in your template, and add the built-in *7.2.17 - : Apache Spark3* template.
2. If the built-in *7.2.17 - : Apache Spark3* template doesn't work, you can create a custom template. Replace Spark 2 and Livy 2 references with Spark 3 and Livy 3, respectively..
3. Add a new Spark 3-based 7.2.17 cluster to the environment, using your custom template or the built-in *7.2.17 - : Apache Spark3* template.
4. Migrate all non-spark workloads from the old cluster to the new cluster.

Application migration tasks (Spark 2 to 3)

Procedure

1. Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 2.4.8 to 3.3.2.
   a) Check the supported Java versions.
   b) Check the supported Scala version.

c) Check the supported Python versions.
d) Account for changed or versioned Spark commands in your code. (spark-submit, pyspark, etc.)
e) Check supported versions for Spark connectors.
f) Check the logging library used in your code.
g) Check the compatibility of 3rd-party libraries used in your code.
h) Check Spark behavior changes and refactor your code.

2. Migrate all Spark 2 applications in the old cluster to Spark 3 applications in the new cluster.

Post-application migration tasks

Procedure

1. Move Spark 2 event logs to the Spark 3 event logs directory.
2. Drop the old cluster.

In-place cluster upgrade

Before you begin



Important: Cluster upgrade to version 7.3.1 is only allowed if all Spark 2 applications have been migrated to Spark 3 and tested in the **source cluster**.

Procedure

1. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.
   c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
   d) Click the Target Version drop-down menu to see any available upgrades.
   e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.
   f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
   g) Click Upgrade to initiate the upgrade.
   h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

   For more information, see Data Lake upgrade.
2. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Application migration tasks (Spark 3.x to 3.4.1)

Procedure

Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 3.3.2 to 3.4.1

a) Refactor your Spark application code.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrading Apache Spark 2.4.8 (with 3.3.2 bundled) on 7.2.17 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 2.4.8 (with 3.3.2 bundled) on Cloudera on cloud 7.2.17 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[7]
7.2.17	2.4.8	3.3.2 (bundled)	**7.3.1**	**3.4.1**	yes

Intermediate in-place cluster upgrade

Procedure

1. Upgrade the cluster OS from Centos 7 to RedHat 8.
2. Upgrade your cluster to 7.2.18 SP2.
   a) Identify cluster version details.
   b) Identify your upgrade path.
   c) Review the prerequisites
   d) High-level upgrade steps.

   For more information on upgrading your cluster to 7.2.18 SP2, see:

   - Upgrading to 7.2.18
3. Once complete, continue the upgrade with the steps described in Upgrading Apache Spark 2.4.8 (with 3.4.1 bundled) on 7.2.18 SP2 to Spark 3 on 7.3.1.

Upgrading Apache Spark 3.3.2 (bundled) on 7.2.17 to Spark 3 on 7.3.1

The following steps will help you upgrading from Apache Spark 3.3.2 (bundled) on Cloudera on cloud 7.2.17 to Spark 3.4.1 on 7.3.1.

Source cluster version	Source cluster Spark 2 version	Source cluster Spark 3 version	Target cluster version	Target cluster Spark 3 version	Spark 2 used with connectors[8]
7.2.17	none	3.3.2 (bundled)	**7.3.1**	**3.4.1**	no

In-place cluster upgrade

Procedure

1. Upgrade the cluster OS from Centos 7 to RedHat 8.

2. Upgrade the Data Lake cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Stop all s attached to the environment.
   c) From the , click Data LakesEnvironment Name, scroll to the bottom of the **Data Lake details** page, and click the Upgrade tab.
   d) Click the Target Version drop-down menu to see any available upgrades.
   e) If you want to skip the automatic backup that is taken before the upgrade, uncheck the Automatic backup box.
   f) Click Validate and Prepare to check for any configuration issues and begin the parcel download and distribution.
   g) Click Upgrade to initiate the upgrade.
   h) Click the Event History tab to monitor the upgrade process and verify that it completes successfully.

   For more information, see Data Lake upgrade.
3. Upgrade the new cluster to 7.3.1
   a) Check the support matrix for upgrades.
   b) Start the cluster.
   c) Check the current version of .
   d) If your cluster uses Streams Replication Manager, export or migrate aggregated metrics.
   e) If you use autoscaling, disable autoscaling on the cluster.
   f) Upgrade the cluster.
   g) Monitor the upgrade progress using the Event History tab.
   h) When the upgrade is complete, verify the new version.
   i) If you disabled autoscaling on the cluster, you can re-enable it after upgrade.

   For more information, see Upgrading s.

Application migration tasks (Spark 3.x to 3.4.1)

Procedure

Follow the Spark application migration documentation to migrate your Apache Spark Applications from version 3.3.2 to 3.4.1

a) Refactor your Spark application code.

Final steps

Procedure

After the upgrade and application migration are complete:

1. Check the status of your Data Lakes, s, and clusters.
2. Perform benchmark testing on your applications. See Spark Application Migration.

Upgrading Apache Spark 3.5.4 to Spark 3.5.8

Cluster and application considerations when Cloudera Runtime rebases bundled Spark from 3.5.4 to 3.5.8.

Apache Spark **3.5.8** ships with Cloudera Runtime **7.3.2.10000 SP1** and higher service packs on the 7.3.2 line (previously 3.5.4 in Cloudera Runtime 7.3.2.0). This is a maintenance rebase: cluster upgrade steps follow your normal Runtime service pack process; application owners should still validate workloads.

Important:

When Cloudera rebases the Apache Spark version shipped with or CDS, minor and maintenance releases can still change SQL behavior, configuration defaults, and serialization. Recompile Java and Scala applications built against Spark, and retest PySpark and Spark SQL workloads in a non-production environment before you upgrade production clusters.

Review *Upgrading Spark* and *Migrating Spark applications* for steps that match your source and target Spark versions.

Read the *Apache Spark migration guides* for the **exact** Apache Spark version on your cluster (for example 3.5.8 or 3.5.4).

For upstream changes between Spark 3.5.4 and 3.5.8, see the Apache Spark 3.5.8 documentation and the migration guides linked from Migrating Spark applications (use the 3.5.8 doc paths where they differ from latest).

For Cloudera application migration tasks and behavior notes, see *Migrating Spark applications* and the topics under *Spark behavior changes* in that guide.

Important: Plan and validate Spark application behavior on Spark 3.5.8 in a non-production environment before you upgrade production clusters.

Related Information

Upgrading Spark

Migrating Spark applications

Migrating Spark applications

How to refactor Spark 2 workloads to Spark 3 during the upgrade/migration process due to the removal of Spark 2 in Cloudera on cloud.

Introduction

The purpose of this document is to gather all information required to carry out a Spark application migration between different versions:

- Spark 2 to Spark 3 (upgrading from Cloudera on cloud 7.2.18).
- Spark 3.4 to Spark 3.5.4 (upgrading to Cloudera on cloud 7.3.1.100 or 7.3.2 or higher).
- Spark 3.5.4 to Spark 3.5.8 (upgrading to Cloudera Runtime 7.3.2.10000 SP1 or higher on the 7.3.2 line); see Upgrading Apache Spark 3.5.4 to Spark 3.5.8.

The necessary set of steps largely depends on the source and target Spark versions. Major version changes require considerable effort. Minor and maintenance Spark rebases in Cloudera Runtime can still change SQL semantics, defaults, or serialization; plan to recompile Java and Scala artifacts and retest applications even when cluster upgrade steps are limited to a service pack.

Important:

When Cloudera rebases the Apache Spark version shipped with or CDS, minor and maintenance releases can still change SQL behavior, configuration defaults, and serialization. Recompile Java and Scala applications built against Spark, and retest PySpark and Spark SQL workloads in a non-production environment before you upgrade production clusters.

Review *Upgrading Spark* and *Migrating Spark applications* for steps that match your source and target Spark versions.

Read the *Apache Spark migration guides* for the **exact** Apache Spark version on your cluster (for example 3.5.8 or 3.5.4).

This guide covers major upgrades (from Spark 2 to Spark 3), Spark 3 minor-line upgrades (for example toward Apache Spark 3.5.4), and maintenance rebases (for example 3.5.4 to 3.5.8). For version-specific behavior notes, see the topics under *Spark behavior changes* in this guide and the Apache Spark migration guides section.

Major version migration

Migration between major versions requires considerable effort and taking into account many factors.

This documentation focuses on migrating applications from Spark 2 to Spark 3, the two major versions currently supported by Cloudera on different versions of Cloudera on cloud.

Related Information

Upgrading Spark
Migrating Spark applications

Java versions

Cloudera currently supports the following JDK versions in general:

- 17

Refer to *Support Matrix* for a list of supported versions of Java.

Important:

JDK17 introduced changes in garbage collection,moving from ParallelGC to G1GC as the default method, which is also the recommended method for Apache Spark.

Cloudera recommends increasing XX:G1HeapRegionSize to prevent too many regions, depending on the workload and heap size assigned to the executor.

Related Information

Cloudera Support Matrix

Scala versions

As Cloudera only supports only Spark 2 applications compiled with Scala **2.11**, and Spark 3 applications with Scala **2.12**, a major version change always require:

1. Spark Scala applications to be recompiled with Scala 2.12,
2. adjusting the dependencies to use Spark 3 version binaries provided by Cloudera in the *public maven repository* and the Scala 2.12 version of third-party libraries.

Scala version changes can also require source code changes, for which see the *Scala documentation*.

Related Information

Cloudera public maven repositories
Cloudera on cloud | Using the Cloudera Runtime Maven repository
Scala documentation | Scala

Python versions

The supported versions of Python can change between Spark versions. Refer to the table below for details on the supported Python versions for each Spark version, and follow the *Python documentation* to adjust your application.

Spark version	Minimum supported Python version	Maximum supported Python version
3.5.4	3.8	3.11
3.5.0	3.8	3.11
3.4.0	3.7	3.11
3.3.2	3.7	3.10
3.3.0	3.7	3.10
3.2.3	3.6	3.9
2.4.8	2.7/3.4	3.7

Related Information

Python documentation | Python

Spark commands

Cloudera supports multiple versions of Spark, depending on the version of Cloudera on cloudCloudera Data Hubs. The general (unversioned) Apache Spark commands (spark-submit, pyspark, etc.) can point to different versions based on the cluster version.

1. The original commands always point to the earliest available version of Spark in the distribution.

   For example, the spark-submit command points to **Spark 2 in version 7.2.18**, but points to **Spark 3 in Cloudera Runtime 7.3.1.100 and higher or 7.3.2.0** (including Spark 3.5.4).
2. Other available Spark 3 versions can be used via versioned commands.

   For example, the spark3-submit command points to Spark 3 in all versions.

Spark connectors

Spark 3 supports certain Spark connectors from certain versions.

If Spark 2 connectors are used, please take the connectors into account when choosing the minimum Cloudera on cloud version you need to upgrade to when migrating a Spark application to a higher version.

- *Hive Warehouse Connector* for Spark 3 is supported from:
  - Cloudera on cloud version 7.2.16
- *HBase connector* for Spark 3 is supported from:
  - Cloudera on cloud version 7.2.12
- *Phoenix* connector for Spark 3 is supported from:
  - Cloudera on cloud version 7.2.15
- *Oozie* for Spark 3 is supported from:
  - Cloudera on cloud version 7.2.18
- *Solr* for Spark 3 is supported from:
  - Cloudera on cloud version 7.2.18
- *Spark Schema Registry* is supported from:

- Cloudera on cloud version 7.2.18 SP2

Logging

Since Apache Spark 3 has transitioned from log4j to log4j2, you need to adjust the logging library and/or logging configuration used in your application.



Important: The log4j and reload4j runtime libraries are no longer included in the classpath by default. They have been replaced with the analogous log4j2 libraries.

Third-party libraries

When migrating between versions, ensure that your 3rd-party runtime dependencies align with the Spark versions.

Spark behavior changes

As Apache Spark evolves, its behavior can change between major and minor versions, but many times legacy configurations are added to support the old behavior.

As configurations can be defined at multiple levels, restoring the old behavior might require changing the application itself, the application starting commands/scripts, and/or the default Spark configurations defined on the cluster.

Cloudera Runtime 7.3.1.200 Service Pack 1

Summary:

The dataframe.show() function displays NULL instead of null.

Previous behavior:

The dataframe.show() function displayed null when the default value is ommitted in the decode function.

New behavior:

The dataframe.show() function now displays NULL instead of the expected null for consistent behavior.

Apache Jira: SPARK-43063: df.show handle null should print NULL instead of null

Cloudera Runtime 7.3.1

For an overview of behavioral changes in Spark, see *Spark 2 to Spark 3 workload refactoring*.

If you read from or write to external databases through the Spark JDBC DataFrame data source, pass the JDBC driver JAR explicitly (for example with --jars) and set the driver option to your JDBC driver class.

Apache Spark Migration guides

The comprehensive guide of behavior changes between versions are available in the Apache Spark Migration Guides.

Always refer to the following documents to ensure that your Spark application remains compatible with newer Spark versions:

- Migration Guide: Spark Core
- Migration Guide: SQL, Datasets and DataFrame
- Migration Guide: Structured Streaming
- Machine Learning Library (MLlib) Guide
- Upgrading PySpark

Spark 2 to Spark 3 workload refactoring

The following list summarizes the most important behavior changes from Spark 2 to Spark 3, and gives examples on how to refactor the Spark 2 application to become Spark 3 compatible.

The list is not exhaustive, refer to the *Apache Spark Migration guides* for the complete list.

Spark Core

Spark Core language/syntactic-level changes

Spark 2	Spark 3	Refactor action
TaskContext.isRunningLocally	Deprecated method, removed.	Remove TaskContext.isRunningLocally if used in code.
ShuffleBytesWritten and shuffleRecordsWritten (ShuffleWriteMetrics class)	bytesWritten and recordsWritten (org.apache.spark.status.api.v1.OutputMetrics class)	Use bytesWritten and recordsWritten, available in class org.apache.spark.status.api.v1.OutputMetrics.
org.apache.spark Class Accumulator	org.apache.spark.util.AccumulatorV2	Replace org.apache.spark.Accumulator with org.apache.spark.util.AccumulatorV2.
For non-struct types, (e.g. int, string, array, Dataset.groupByKey) results in a grouped dataset with key attribute is wrongly named as value.	For non-struct types (e.g. int, string, array, Dataset.groupByKey) results to a grouped dataset with key attribute is named as key.	Refactor the value attribute used in logic to key. To preserve the old behavior, set spark.sql.legacy.dataset.nameNonStructGroupingKeyAsValue to true.

Spark SQL

Spark SQL language/syntactic-level changes

Spark 2	Spark 3	Refactor action
Path option is overwritten if one path parameter is passed to DataFrameReader.load(), DataFrameWriter.save(), DataStreamReader.load(), or DataStreamWriter.start().	Path option cannot coexist when the following methods are called with path parameter(s): DataFrameReader.load(), DataFrameWriter.save(), DataStreamReader.load(), or DataStreamWriter.start().	Remove the path option if it's the same as the path parameter, or add it to the load() parameter if you do want to read multiple paths. To ignore this check, set spark.sql.legacy.pathOptionBehavior.enabled to true.
count(tblName.*) works.	An exception is thrown if count(tblName.*) is used for getting the number of records in the table.	Refactor the code to use count(*), or expand the columns manually. (Example: count(col1, col2).) To restore the old behavior, set spark.sql.legacy.allowStarWithSingleTableIdentifierInCount to true.

Spark SQL configuration-level changes

Spark 2	Spark 3	Refactor action
SET command works for SparkConf entries.	AnalysisException error is thrown if SET command is used to modify the SparkConf entries.	Remove SET commands for SparkConf entries from your code. You can enter SparkConf values at the cluster level by entering them in the cluster's Spark configuration and restarting the cluster. To disable the check, set spark.sql.legacy.setCommandRejectsSparkCoreConfs to false.
The second argument of date_add function (num_days) can be a fraction, as it gets casted to Int internally.	The second argument of date_add function (num_days). If an integer is not provided, an AnalysisException is thrown.	Make sure that in code always integer is passed as the second argument to date_add and date_subtract function.
Fractional and string types are allowed in percentile_approx third argument i.e. accuracy, as it gets casted to Int internally.	percentile_approx third argument accuracy can only be integer. If an integer is not provided, an AnalysisException is thrown.	Make sure that in code always integer is passed as the third argument to percentile_approx function.
Hash expressions can be applied on MapType elements.	Hash expressions are prohibited on MapType elements.	If hash expression is applied on map type, refactor the code to remove it, OR set spark.sql.legacy.allowHashOnMapType to true.
a map can be created with duplicate keys via built-in functions like CreateMap, StringToMap, map_from_arrays etc.	Spark throws RuntimeException when duplicated keys are found in Map. Users may still read map values with duplicate keys from data sources which do not enforce it (for example, Parquet).	If duplicate keys are passed into built in functions to create a map then try to remove duplicate keys OR set spark.sql.mapKeyDedupPolicy to LAST_WIN, the map keys are deduplicated.
the resulting date is adjusted in add_months, when the original date is a last day of months. For example, adding a month to 2019-02-28 results in 2019-03-31.	the add_months function does not adjust the resulting date to a last day of month if the original date is a last day of months. For example, adding a month to 2019-02-28 results in 2019-03-28.	Adjust the code according to logic if required.
multiple from-to units is allowed in Interval literal.	multiple from-to units Interval literal is not allowed.	Remove multiple from-to units is allowed in Interval literal. Adjust the code according to logic if required.
Dataset query success if it contains ambiguous column reference that is caused by self join.	Dataset query fails if it contains ambiguous column reference that is caused by self join. This is because Spark cannot resolve Dataset column references that point to tables being self joined, and df1("a") is exactly the same as df2("a") in Spark.	Use aliases. For example: df2.as("purchases").join(df1.as("devices"), col("devices.key1") === col("purchases.key2")).show()
invalid time zone ids are silently ignored and replaced by GMT timezone.	invalid time zone ids are rejected, and Spark throws java.time.DateTimeException.	rectify to correct Zone ID.
for Parsing and formatting of timestamp and date strings, java.text.SimpleDateFormat is used for timestamp/date string conversions, and the supported patterns are described in SimpleDateFormat.	DateTimeFormatter under the hood or Parsing and formatting of timestamp and date strings. Strict checking of Input is performed.	Refactor the code to correct pattern matching for Input OR set spark.sql.legacy.timeParserPolicy to LEGACY to restore the behavior, OR set it to CORRECTED and treat it as an invalid datetime string.
datetime pattern letter F is aligned to week of month that represents the concept of the count of weeks within the month where weeks start on a fixed day-of-week.	datetime pattern letter F is aligned to day of week in month that represents the concept of the count of days within the period of a week where the weeks are aligned to the start of the month.	Refactor the code to accommodate new behavior of pattern F.
SparkContext can be created in executors.	an exception will be thrown when creating SparkContext in executor.	Refactor the code to remove the creation of Spark context OR allow it by setting the configuration spark.executor.allowSparkContext when creating SparkContext in executors.
TRANSFORM operator can support alias in inputs.	TRANSFORM operator can't support alias in inputs.	Refactor the code to remove aliases from Inputs.
Loading and saving of timestamps from and to Parquet files does not fail if the timestamps are before 1900-01-01 00:00:00Z.	Loading and saving of timestamps from and to Parquet files fails if the timestamps are before 1900-01-01 00:00:00Z.	Ensure that Input reads do not contain timestamps before 1900-01-01 00:00:00Z. Alternatively, set spark.sql.parquet.int96RebaseModeInWrite to CORRECTED to write the datetime values as it is.

Spark 2	Spark 3	Refactor action
The Char(n) type handled inconsistently, depending on whether the table is partitioned or not.	In upstream Spark 3 the spark.sql.legacy.charVarcharAsString config was introduced, but does not solve all incompatibilities.	A new configuration spark.cloudera.legacy.charVarcharLegacyPadding is introduced in Cloudera to keep the full compatibility. **Note:** Although this workaround is available, using CHAR is discouraged.
The Row field names are sorted alphabetically when constructing with named arguments for Python versions 3.6 and above.	The Row field names are no longer sorted alphabetically.	To enable sorted fields by default as in Spark 2.4, set the environment variable PYSPARK_ROW_FIELD_SORTING_ENABLED to true for both the executors and the driver.

Spark SQL property-level changes

Spark 2	Spark 3	Refactor action
When there is nested CTE with a conflicting name, outer CTE definitions take precedence.	When there is nested CTE with conflicting name, Spark throws an AnalysisException by default, and forces users to choose the specific substitution order they wanted. If the value of spark.sql.legacy.ctePrecedencePolicy is set to CORRECTED (recommended), inner CTE definitions take precedence over outer definitions . If the value of spark.sql.legacy.ctePrecedencePolicy is set to LEGACY, outer CTE definitions take precedence over inner definitions.	Set spark.sql.legacy.ctePrecedencePolicy to CORRECTED.
Type conversions during table insertion are allowed as long as they are valid Cast.	The type coercion is performed as per the ANSI SQL standard.	Ensure the type coercion is performed as per the ANSI SQL standard. Alternatively, set spark.sql.storeAssignmentPolicy to Legacy to restore previous behavior.

Spark storage location configuration changes

To execute workloads in Cloudera on cloud, modify the local data storage locations to cloud storage (for example, from HDFS to S3 bucket).

The following example shows a sample workload, with the modified data location **highlighted in bold**.

Spark 3.2 (HDFS)	Spark 3.2 (S3)
spark-shell	spark-shell
`scala> spark.sql("CREATE TABLE IF NOT EXISTS default.sales_spark_2(Region string, Country string,Item_Type string,Sales_Channel string,Order_Priority string,Order_Date date,Order_ID int,Ship_Date date,Units_sold string,Unit_Price string,Unit_cost string,Total_revenue string,Total_Cost string,Total_Profit string) row format delimited fields terminated by ','")` `scala> spark.sql("load data local inpath '/tmp/sales.csv' into table default.sales_spark_3")`	`scala> spark.sql("CREATE TABLE IF NOT EXISTS default.sales_spark_2(Region string, Country string,Item_Type string,Sales_Channel string,Order_Priority string,Order_Date date,Order_ID int,Ship_Date date,Units_sold string,Unit_Price string,Unit_cost string,Total_revenue string,Total_Cost string,Total_Profit string) row format delimited fields terminated by ','")` `scala> spark.sql("load data inpath 's3://[*** BUCKET ***]/sales.csv' into table default.sales_spark_3")`
`scala> spark.sql("select count(*) from default.sales_spark_3").show()`	`scala> spark.sql("select count(*) from default.sales_spark_3").show()`

Unsupported features

Unsupported Spark 3 features in Cloudera on cloud.

ZSTD compression in ORC data source

spark.sql.orc.compression.codec config doesn't accept zstd value.

Apache Jira:SPARK-33978

spark.hadoopRDD.ignoreEmptySplits

Causes issues in HBase TableInputFormat.

Apache Jira:SPARK-34809

LDAP authentication for livy-server

Open CVEs in Apache Directory Server dependency, LDAP based authentication is not supported in Livy Server.

Apache Jira:LIVY-356

Thrift ldap authentication, based on ldapurl, basedn, domain

Open CVEs in Apache Directory Server dependency, LDAP based authentication is not supported in Livy Thrift Server.

Apache Jira:LIVY-678

For more information, see *Unsupported Apache Spark Features*.

Related Information

Unsupported Apache Spark Features

Post-migration checklist

Benchmark testing

After all post-migration configurations are performed, perform benchmark testing on the new Apache Spark version.

Troubleshooting

Troubleshoot failed or slow performing workloads by analyzing the application event and driver logs, and fine-tune the workloads for better performance.

JDBC No suitable driver errors after an upgrade

Spark jobs can fail with java.sql.SQLException: No suitable driver when you read from or write to an external database through the Spark JDBC data source, often after you upgrade to Spark 3.x (including in YARN cluster mode).

To resolve the error, complete the following steps:

1. Pass the JDBC driver JAR to the application so that executors receive it (for example spark-submit --jars /path/to/driver.jar ... with a path on shared storage that all cluster nodes can read).
2. Set the JDBC driver class in Spark options (for example .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") on your format("jdbc") read or write).
3. If the driver is still not loaded, set spark.driver.extraClassPath and spark.executor.extraClassPath for your deployment.

Executor or driver OOM on joins after upgrading to Spark 3

After you upgrade from Spark 2.4 to Spark 3.x, join-heavy ETL pipelines can fail with driver or executor java.lang.OutOfMemoryError messages even though the same workloads were stable on Spark 2.4.

Spark 3 enables Adaptive Query Execution (AQE) by default. Unlike the static query plans in Spark 2.4, AQE can re-optimize join strategies at runtime and dynamically choose broadcast joins based on updated statistics. That can broadcast larger datasets than you expect and increase memory pressure on the driver or executors.

Mitigate the issue using Spark SQL configuration:

- Disable broadcast joins by setting spark.sql.autoBroadcastJoinThreshold to -1.
- Or reduce the broadcast threshold by setting spark.sql.autoBroadcastJoinThreshold to a lower value that is safe for your workload.
- If dynamic join re-planning remains problematic, set spark.sql.adaptive.enabled to false.

For Cloudera Manager steps to apply these settings on the Spark client, see Known Issues in Apache Spark (INSIGHT-9609).