Cloudera Runtime 7.2.18

Using Apache Zeppelin

Date published: 2020-07-28
Date modified: 2024-03-11

CLOUDERA

Legal Notice

Contents

Introduction

An Apache Zeppelin notebook is a browser-based GUI you can use for interactive data exploration, modeling, and visualization.

⚠️ **Important:** Zeppelin will be deprecated in Cloudera Runtime 7.2.18. For more information, see *Deprecation Notices in Cloudera Runtime 7.2.18.*

As an Apache Zeppelin notebook author or collaborator, you write code in a browser window. When you run the code from the browser, Zeppelin sends the code to backend processors such as Spark. The processor or service and returns results; you can then use Zeppelin to review and visualize results in the browser.

Apache Zeppelin is supported on the following browsers:

- Internet Explorer, latest supported releases. (Zeppelin is not supported on versions 8 or 9, due to lack of built-in support for WebSockets.)
- Google Chrome, latest stable release.
- Mozilla Firefox, latest stable release.
- Apple Safari, latest stable release. Note that when SSL is enabled for Zeppelin, Safari requires a Certificate Authority-signed certificate to access the Zeppelin UI.

Launch Zeppelin

Use the following steps to launch Apache Zeppelin.

To launch Zeppelin in your browser:

1. In the Cloudera Data Platform (CDP) Management Console, go to Data Hub Clusters.
2. Find and select the cluster you want to use.
3. Click the Zeppelin link.

When you first connect to Zeppelin, you will see the home page:
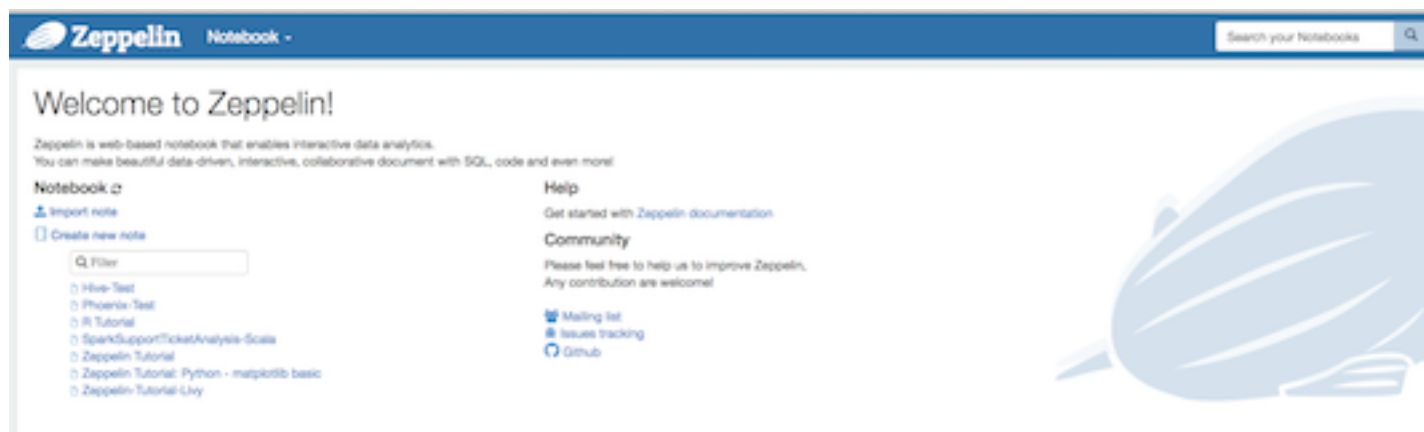
The following menus are available in the top banner of all Zeppelin pages:

- Notebook

  Open a note, filter the list of notes by name, or create a note:
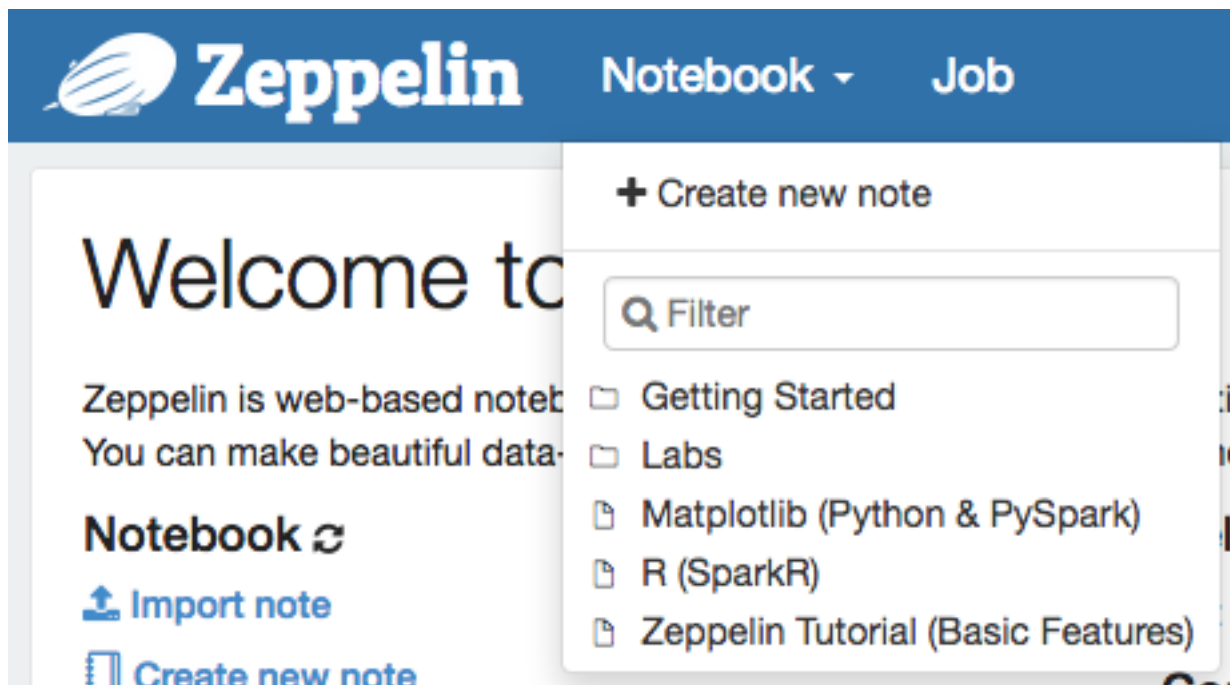
- User settings
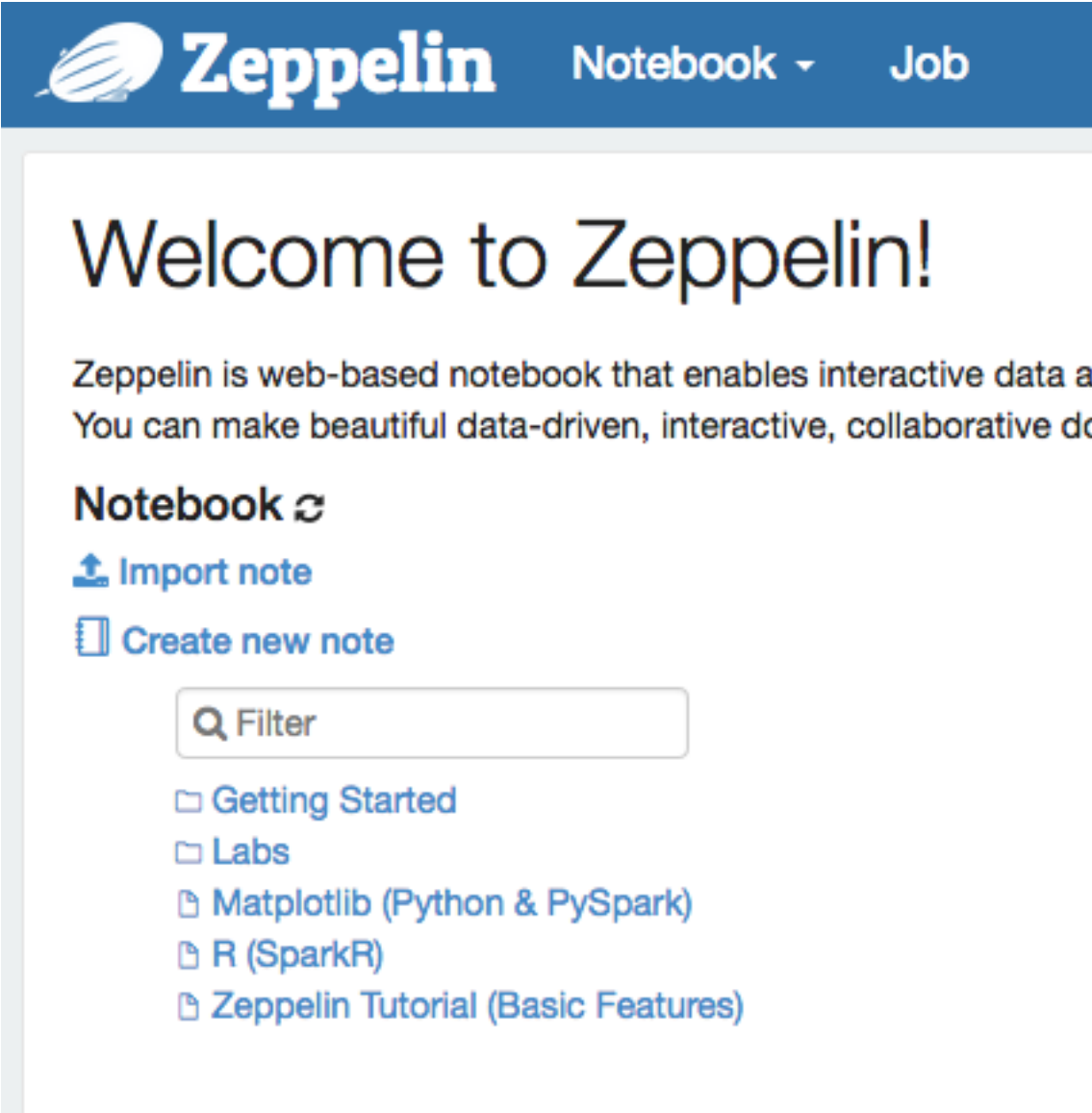
  Displays your username, or "anonymous" if security is not configured for Zeppelin.
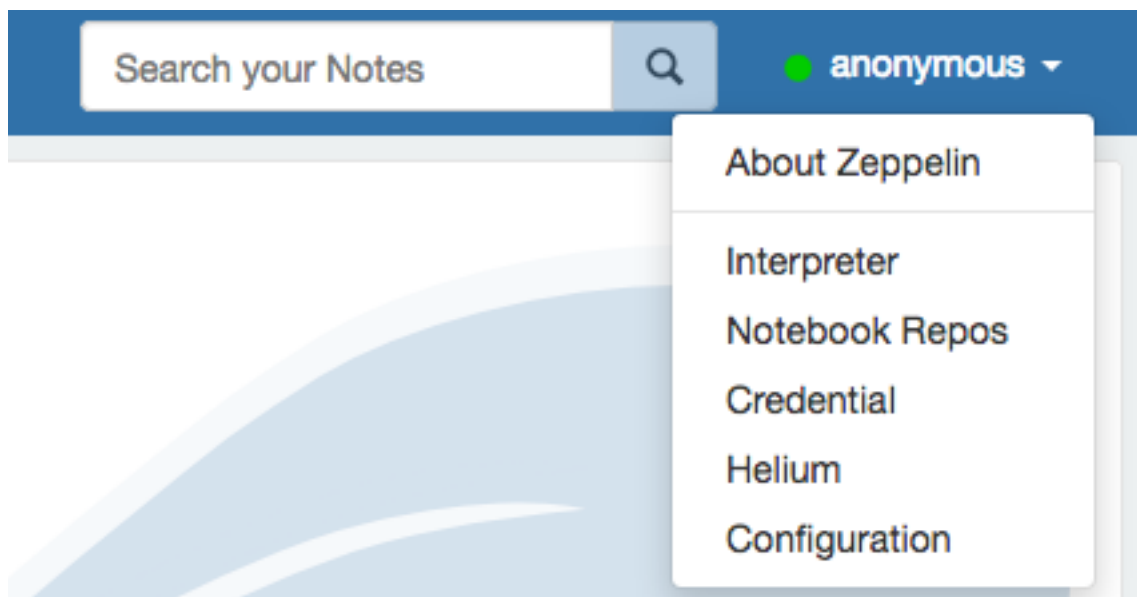
- List version information about Zeppelin
- Review interpreter settings and configure, add, or remove interpreter instances
- Save credentials for data sources
- Display configuration settings

Each instance of Zeppelin contains "notes", which are the fundamental elements of a Zeppelin notebook.

Zeppelin lists available notes on the left side of the welcome screen and in the "Notebook" menu. Zeppelin ships with several sample notes, including tutorials:

Working with Zeppelin Notes

This section provides an introduction to Apache Zeppelin notes.

An Apache Zeppelin note consists of one or more paragraphs of code, which you can use to define and run snippets of code in a flexible manner.

A paragraph contains code to access services, run jobs, and display results. A paragraph consists of two main sections: an interactive box for code, and a box that displays results. To the right is a set of paragraph commands. The following graphic shows paragraph layout.

Zeppelin ships with several sample notes, including tutorials that demonstrate how to run Spark scala code, Spark SQL code, and create visualizations.

To run a tutorial:

1. Navigate to the tutorial: click one of the Zeppelin tutorial links on the left side of the welcome page, or use the Notebook pull-down menu.
2. Zeppelin presents the tutorial, a sequence of paragraphs prepopulated with code and text.
3. Starting with the first paragraph, click the triangle button at the upper right of the paragraph. The status changes to PENDING, RUNNING, and then FINISHED when done.
4. When the first cell finishes execution, results appear in the box underneath your code. Review the results.
5. Step through each cell, running the code and reviewing results.

Create and Run a Note

Use the following steps to create and run an Apache Zeppelin note.

To create a note:

1. Click "Create new note" on the welcome page, or click the "Notebook" menu and choose "+ Create new note."

2. Type your commands into the blank paragraph in the new note.

When you create a note, it appears in the list of notes on the left side of the home page and in the Notebook menu. By default, Zeppelin stores notes in the $ZEPPELIN_HOME/notebook folder.

To run your code:

1. Click the triangle button in the cell that contains your code:

▷

2. Zeppelin displays status near the triangle button: PENDING, RUNNING, ERROR, or FINISHED.
3. When finished, results appear in the result section below your code.

The settings icon (outlined in red) offers several additional commands:

FINISHED ▷ ⤢ 📖 ⚙

These commands allow you to perform several note operations, such as showing and hiding line numbers, clearing the results section, and deleting the paragraph.

FINISHED ▷ ⤢ 📖 ⚙

20150210-015259_1403135953

↔ Width   12 ⇕

⊕ Move Up

⊕ Move Down

⊕ Insert New

A Hide title

☰ Show line numbers

▷ Disable run

↗ Link this paragraph

✎ Clear output

✖ Remove

Import a Note

Use the following steps to import an Apache Zeppelin note.

About this task
To import a note from a URL or from a JSON file in your local file system:

Procedure

1. Click "Import note" on the Zeppelin home page:

Notebook ↻

⬆ Import note

2. Zeppelin displays an import dialog box:
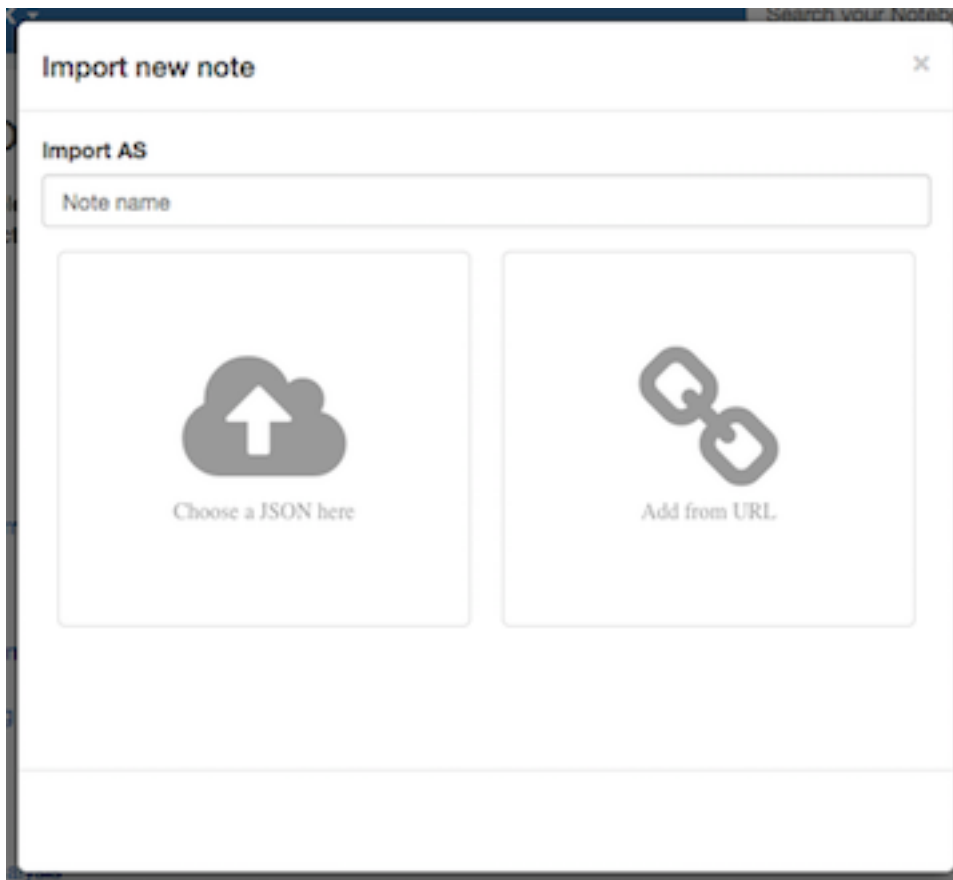
3. To upload the file or specify the URL, click the associated box.

By default, the name of the imported note is the same as the original note. You can rename it by providing a new name in the "Import AS" field.

Export a Note

Use the following steps to export an Apache Zeppelin note.

To export a note to a local JSON file, use the export note icon in the note toolbar:
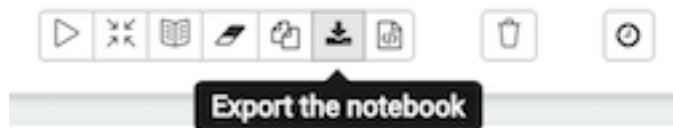
Zeppelin downloads the note to the local file system.

Note: Zeppelin exports code and results sections in all paragraphs. If you have a lot of data in your results sections, consider trimming results before exporting them.

Using the Note Toolbar

This section describes how to use the Apache Zeppelin Note toolbar.

At the top of each note there is a toolbar with buttons for running code in paragraphs and for setting configuration, security, and display options:

There are several buttons in the middle of the toolbar:

These buttons perform the following operations:

• Run all paragraphs in the note sequentially, in the order in which they are displayed in the note.
• Hide or show the code section of all paragraphs.
• Hide or show the result sections in all paragraphs.
• Clear the result section in all paragraphs.
• Clone the current note.
• Export the current note to a JSON file.

    Note that the code and result sections in all paragraphs are exported. If you have extra data in some of your result sections, trim the data before exporting it.
• Commit the current note content.
• Delete the note.
• Schedule the execution of all paragraphs using CRON syntax. This feature is not currently operational. If you need to schedule Spark jobs, consider using Oozie Spark action.

There are additional buttons on the right side of the toolbar:
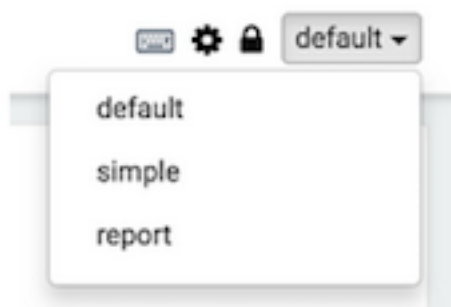
These buttons perform the following operations (from left to right):

• Display all keyboard shortcuts.
• Configure interpreters that are bound to the current note.
• Configure note permissions.

- Switch display mode:

  - Default: the notebook can be shared with (and edited by) anyone who has access to the notebook.
  - Simple: similar to default, with available options shown only when your cursor is over the cell.
  - Report: only your results are visible, and are read-only (no editing).

  Note: Zeppelin on HDP does not support sharing a note by sharing its URL, due to lack of proper access control over who and how a note can be shared.

Import External Packages

Use the following steps to import external packages into Apache Zeppelin.

To use an external package within a Zeppelin note, you can use one of the following approaches:

- Specify the dependency for the associated interpreter on the Interpreter page.

  For more information, see the Dependency Management for Interpreter documentation at zeppelin.apache.org.
- For Spark jobs, you can pass a jar, package, or list of files to spark-submit using SPARK_SUBMIT_OPTIONS; for example:

  - SPARK_SUBMIT_OPTIONS in conf/zeppelin-env.sh

    ```
    export SPARKSUBMITOPTIONS="--packages com.databricks:spark-csv_2.10:1.2.
    0 --jars /path/mylib1.jar,/path/mylib2.jar --files /path/mylib1.py,/path
    /mylib2.zip,/path/mylib3.egg"
    ```

  - In SPARK_HOME/conf/spark-defaults.conf

    ```
    spark.jars /path/mylib1.jar,/path/mylib2.jar spark.jars.packages com.dat
    abricks:spark-csv_2.10:1.2.0 spark.files /path/mylib1.py,/path/mylib2.eg
    g,/path/mylib3.zip
    ```

If you want to import a library for a note that uses the Livy interpreter, see "Using the %livy Interpreter to Access Spark" in the HDP Apache Spark guide.

Configuring and Using Zeppelin Interpreters

An Apache Zeppelin interpreter is a plugin that enables you to access processing engines and data sources from the Zeppelin UI.

For example, if you want to use Python code in your Zeppelin notebook, you need a Python interpreter. Each interpreter runs in its own JVM on the same node as the Zeppelin server. The Zeppelin server communicates with interpreters through the use of Thrift.

Apache Zeppelin on Cloudera Data Platform supports the following interpreters:

- JDBC (supports Hive).

  However, do note that the JDBC interpreter for Apache Hive is deprecated.
- Markdown
- Livy (supports Spark, Spark SQL, PySpark, PySpark3, and SparkR)

  Note: If you are upgrading from HDP to , ensure that your existing notebooks are manualy migrated to the Livy interpreter. Support for the Spark interpreter is no longer available.

- AngularJS

Note: PySpark and associated libraries require Python version 2.7 or later, or Python version 3.4 or later, installed on all nodes.

Modify interpreter settings

Use the following steps to modify Apache Zeppelin interpreter settings.

Before using an interpreter, you might want to modify default settings such as the home directory for the Spark interpreter, the name of the Hive JDBC driver for the JDBC interpreter, or the keytab and principal name for a secure installation.

To set custom configuration values for an interpreter:

1. Click the user settings menu and navigate to the Interpreter page.
2. Scroll down to the Properties section for the interpreter of interest, and click "edit":

3. Make your changes.
4. Scroll to the end of the list of settings and click "Save".
5. Some types of changes require restarting the interpreter; use the button next to the edit button.

Note: The Interpreter page is subject to access control settings. If the page does not list a set of interpreters, check with your system administrator.

Using Zeppelin Interpreters

This section describes how to use Apache Zeppelin interpreters.

Before using an interpreter, ensure that the interpreter is available for use in your note:

1. Navigate to your note.
2. Click on "interpreter binding":

3. Under "Settings", make sure that the interpreter you want to use is selected (in blue text). Unselected interpreters appear in white text:

4. To select an interpreter, click on the interpreter name to select the interpreter. Each click operates as a toggle.
5. You should unselect interpreters that will not be used. This makes your choices clearer. For example, if you plan to use %livy to access Spark, unselect the %spark interpreter.

Whenever one or more interpreters could be used to access the same underlying service, you can specify the precedence of interpreters within a note:

- Drag and drop interpreters into the desired positions in the list.
- When finished, click "Save".

Use an interpreter in a paragraph

To use an interpreter, specify the interpreter directive at the beginning of a paragraph, using the format %[INTERP RETER_NAME]. The directive must appear before any code that uses the interpreter.

The following paragraph uses the %sh interpreter to access the system shell and list the current working directory:

```
%sh
```

```
pwd
home/zeppelin
```

Some interpreters support more than one form of the directive. For example, the %livy interpreter supports directives for PySpark, PySpark3, SparkR, Spark SQL.

To view interpreter directives and settings, navigate to the Interpreter page and scroll through the list of interpreters or search for the interpreter name. Directives are listed immediately after the name of the interpreter, followed by options and property settings. For example, the JDBC interpreter supports the %jdbc directive:

Note: The Interpreter page is subject to access control settings. If the Interpreters page does not list settings, check with your system administrator for more information.

Use interpreter groups

Each interpreter belongs to an interpreter group. Interpreters in the same group can reference each other. For example, if the Spark SQL interpreter and the Spark interpreter are in the same group, the Spark SQL interpreter can reference the Spark interpreter to access its SparkContext.

Customize interpreter settings in a note

This section describes how to customize Apache Zeppelin interpreter settings on a per-note basis.

About this task

You can use the Zeppelin conf interpreter to customize interpreter configuration settings on a per-note basis. The conf interpreter is a generic interpreter that can be used to customize any Zeppelin interpreter on a per-note basis.

In the following example, zeppelin_custom_note_conf.png to customize the Spark interpreter in a Note.

First paragraph:

```
%spark.conf
spark.app.namehelloworld
master  yarn-client
spark.jars.packages com.databricks:spark-csv_2.11:1.2.0
```

Second paragraph:

```
%spark

import com.databricks.spark.csv._
```

In the first paragraph, the conf interpreter is used to create a custom Spark interpreter configuration (set app name, yarn-client mode, and add spark-csv dependencies). After running the first paragraph, the second paragraph can be run to use spark-csv in the note.

In order for the conf interpreter to run successfully, it must be configured on an isolated per-note basis. Also, the paragraph with the conf interpreter customization settings must be run first, before subsequent applicable interpreter processes are launched.

Use the JDBC interpreter to access Hive

This section describes how to use the Apache Zeppelin JDBC interpreter to access Apache Hive.

The %jdbc interpreter supports access to Apache Hive data. The interpreter connects to Hive via Thrift.

If you want Hive queries to run under the user ID originating the query, see "Configuring User Impersonation for Access to Hive" in this guide.

To use the JDBC interpreter to access Hive:

1. Add the following directive at the start of a paragraph:

    %jdbc(hive)
2. Next, add the query that accesses Hive.

Here is a sample paragraph:

```
%jdbc(hive)
SELECT * FROM db_name;
```

If you receive an error, you might need to complete the following additional steps:

1. Copy Hive jar files to /opt/cloudera/parcels/<version>/lib/zeppelin/interpreter/<name_of_interpreter> (or create a soft link).

    For example, /opt/cloudera/parcels/<version>/lib/zeppelin/interpreter/livy/
2. In the Zeppelin UI, navigate to the %jdbc section of the Interpreter page.
3. Click edit, then add a hive.proxy.user.property property and set its value tohive.server2.proxy.user.
4. Click Save, then click restart to restart the JDBC interpreter.

Use the Livy interpreter to access Spark

This section describes how to use the Livy interpreter to access Apache Spark.

The Livy interpreter offers several advantages over the default Spark interpreter (%spark):

•   Sharing of Spark context across multiple Zeppelin instances.
•   Reduced resource use, by recycling resources after 60 minutes of activity (by default). The default Spark interpreter runs jobs--and retains job resources--indefinitely.
•   User impersonation. When the Zeppelin server runs with authentication enabled, the Livy interpreter propagates user identity to the Spark job so that the job runs as the originating user. This is especially useful when multiple users are expected to connect to the same set of data repositories within an enterprise. (The default Spark interpreter runs jobs as the default Zeppelin user.)
•   The ability to run Spark in yarn-cluster mode.

Prerequisites:

•   Before using SparkR through Livy, R must be installed on all nodes of your cluster. For more information, see "SparkR Prerequisites" in the HDP Apache Spark guide.
•   Before using Livy in a note, check the Interpreter page to ensure that the Livy interpreter is configured properly for your cluster.

Note: The Interpreter page is subject to access control settings. If the Interpreters page does not list access settings, check with your system administrator for more information.

To access PySpark using Livy, specify the corresponding interpreter directive before the code that accesses Spark; for example:

```
%livy.pyspark
print "1"
```

```
1
```

Similarly, to access SparkR using Livy, specify the corresponding interpreter directive:

```
%livy.sparkr
hello <- function( name ) {
    sprintf( "Hello, %s", name );
}

hello("livy")
```

> **Important:**
>
> To use SQLContext with Livy, do not create SQLContext explicitly. Zeppelin creates SQLContext by default.
>
> If necessary, remove the following lines from the SparkSQL declaration area of your note:
>
> ```
> //val sqlContext = new org.apache.spark.sql.SQLContext(sc)
> //import sqlContext.implicits._
> ```

Livy sessions are recycled after a specified period of session inactivity. The default is one hour.

For more information about using Livy with Spark, see "Submitting Spark Applications Through Livy" in the HDP Apache Spark guide.

Importing External Packages

To import an external package for use in a note that runs with Livy:

1. Navigate to the interpreter settings.
2. If you are running the Livy interepreter in yarn-cluster mode, edit the Livy configuration on the Interpreters page as follows:

    a. Add a new key, livy.spark.jars.packages.
    b. Set its value to <group>:<id>:<version>.

        Here is an example for the spray-json library, which implements JSON in Scala:

        io.spray:spray-json_2.10:1.3.1

Using Spark Hive Warehouse and HBase Connector Client .jar files with Livy

This section describes how to use Spark Hive Warehouse Connector (HWC) and HBase-Spark connector client .jar files with Livy. These steps are required to ensure token acquisition and avoid authentication errors.

Use the following steps to use Spark HWC and HBase-Spark client .jar files with Livy:

1. Copy the applicable HWC or HBase-Spark .jar files to the Livy server node and add these folders to the livy.file.lo cal-dir-whitelist property in the livy.conf file.

**2.** Add the required Hive and HBase configurations in the Spark client configuration folder:

- Hive:

  For Spark 2 — /etc/spark/conf/hive-site.xml

  For Spark 3 — /etc/spark3/conf/hive-site.xml
- HBase:

  For Spark 2 — /etc/spark/conf/hbase-site.xml

  For Spark 3 — /etc/spark3/conf/hbase-site.xml

Or add the required configurations using the conf field in the session creation request. This is equivalent to using "--conf" in spark-submit.

3. Reference these local .jar files in the session creation request using the file:/// URI format.

HWC Example

1. In Cloudera Manager, go to Clusters Livy .
2. Click the Configuration tab and search for the "Livy Server Advanced Configuration Snippet (Safety Valve) for livy-conf/livy.conf" property.
3. Add the /opt/cloudera/parcels/CDH-<version>/jars folder to the livy.file.local-dir-whitelist property in the livy .conf file.
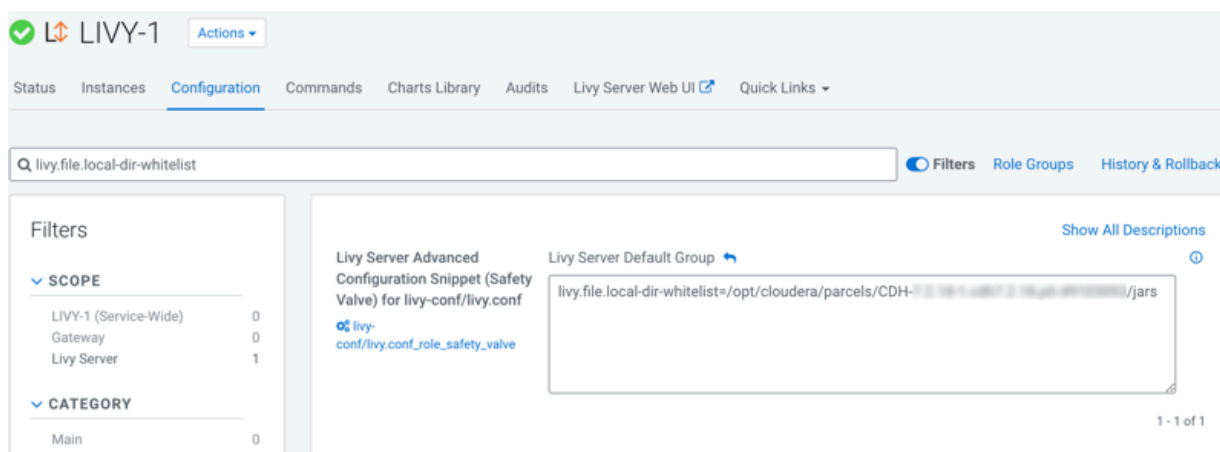
4. Save the changes and restart the Livy service.
5. Log in to the Zeppelin Server Web UI and restart the Livy interpreter.
6. When running the Zeppelin Livy interpreter, reference the HWC .jar file as shown below.

- For Spark 2:

```
%livy2.conf
livy.spark.jars file:///opt/cloudera/parcels/CDH-<version>/jars/hive-war
ehouse-connector-assembly-<version>.jar
```

- For Spark 3:

```
%livy2.conf
livy.spark.jars file:///opt/cloudera/parcels/CDH-<version>/jars/hive-war
ehouse-connector-spark3-assembly-<version>.jar
```

HBase-Spark Connector Example

1. When running the Zeppelin Livy interpreter, reference the following HBase .jar files as shown below. Note that some of these .jar files have 644/root permissions, and therefore may throw an exception. If this happens, you may need to change the permissions of the applicable .jar files on the Livy node.

```
%livy.conf
livy.spark.jars file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-spa
rk3-1.0.0.7.2.18.0-566.jar
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-shaded-protobuf-4.1
.4.7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-shaded-miscellane
ous-4.1.4.7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-protocol-shaded-2.
4.17.7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-shaded-netty-4.1.4.
7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-shaded-client-2.4.17
.7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-shaded-mapreduce-2.
4.17.7.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-common-2.4.17.7.2.1
8.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-server-2.4.17.7.2
.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-client-2.4.17.7.2.1
8.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-protocol-2.4.17.7
.2.18.0-566.jar,
file:///opt/cloudera/parcels/CDH-<version>/jars/hbase-mapreduce-2.4.17.
7.2.18.0-566.jar,
ile:///opt/cloudera/parcels/CDH-<version>/jars/guava-32.1.3-jre.jar
```

Note: The references to HBase-Spark connector jar file are included because HBase-Spark connector was used in this example. They are not required for token acquisition.