

ab: Zeppelin Setup

About This Lab

Objective:	Setup Zeppelin for this class
File locations:	N/A
Successful outcome:	You will: Run Zeppelin notebooks using the Python and shell interpreters on your machines
Before you begin	N/A
Related lesson:	<i>None</i>

Start Zeppelin

1. Connect to the external IP of your Ambari node on port 9995
2. Log in as admin/admin

Login

User Name

User Name

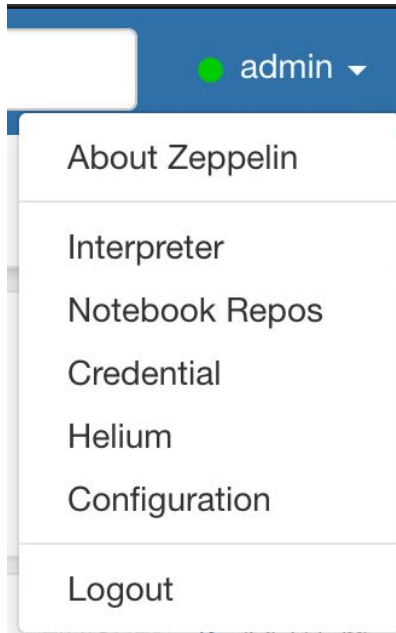
Password

Password

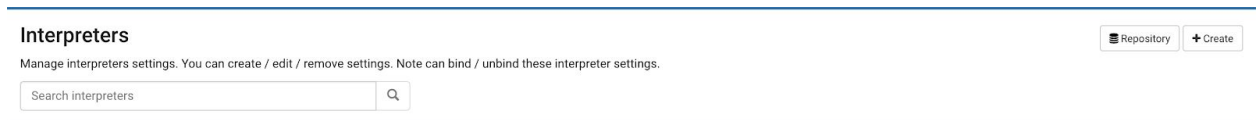
Login

Install the shell interpreter

3. Select the interpreter option of the admin menu



4. Click on the 'Create' button on the right hand side



5. Select the 'sh' interpreter group,
name the interpreter 'sh',
boost the shell.command.timeout.millisecs property to 6000000 and
check the 'shell.working.directory.user.home'.

sh %sh

Option

The interpreter will be instantiated Globally in shared process

☐ Connect to existing process

☐ Set permission

Properties

name	value	action
shell.command.timeout.millisecs	6000000	<input type="button" value="x"/>
shell.working.directory.user.home	<input checked="" type="checkbox"/>	<input type="button" value="x"/>
zeppelin.shell.auth.type	<input type="text"/>	<input type="button" value="x"/>
zeppelin.shell.interpolation	<input type="checkbox"/>	<input type="button" value="x"/>
zeppelin.shell.keytab.location	<input type="text"/>	<input type="button" value="x"/>
zeppelin.shell.principal	<input type="text"/>	<input type="button" value="x"/>
<input type="text"/>	<input type="text"/>	<input type="button" value="x"/> <input type="button" value="S"/> <input type="button" value="+"/> <input type="button" value="textarea"/>

6. Click on the 'Save' button.

Install the python interpreter

1. Open a terminal and connect to your Ambari node using the private keypair provided to you:
`ssh -i .ssh/training-keypair.pem centos@<AmbariNode>`
2. Assume zeppelin service user
`sudo su - zeppelin`
3. Navigate to your Zeppelin home directory
`cd /usr/hdp/current/zeppelin-server/`
4. Add the python interpreter
`./bin/install-interpreter.sh --name python`
5. Restart the Zeppelin server
`./bin/zeppelin-daemon.sh restart`
6. Log in as admin/admin again

Login

User Name

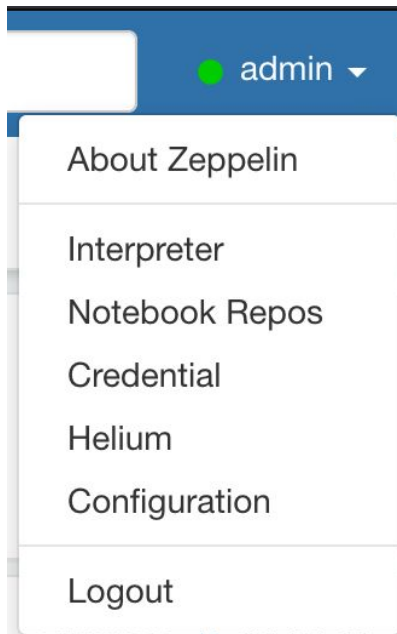
User Name

Password

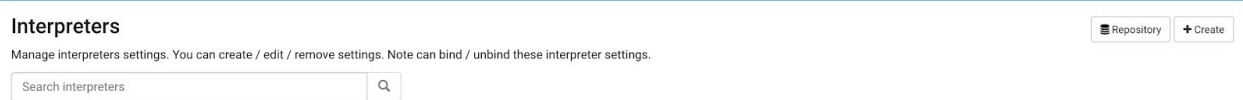
Password

Login

7. In the Zeppelin UI select the interpreter option of the admin menu



8. Click on the 'Create' button on the right hand side



9. Select the 'python' interpreter group, name the interpreter 'python' and set the zeppelin.python property to 'python3.5'.

Create new interpreter

Interpreter Name
python

Interpreter group
python

Option
The interpreter will be instantiated Globally in shared process

☐ Connect to existing process

☐ Set permission

Properties

name	value	action	description
zeppelin.ipython.grpc.message_size	33554432	✕	grpc message size, default is 32M
zeppelin.ipython.launch.timeout	30000	✕	time out for ipython launch
zeppelin.python	python3.5	✕	Python directory. It is set to python by default.(assume python is in your \$PATH)
zeppelin.python.maxResult	1000	✕	Max number of dataframe rows to display.
zeppelin.python.useIPython	<input checked="" type="checkbox"/>	✕	whether use IPython when it is available

10. Click on the 'Save' button.

Configure the SPARK_HOME environment variable

1. Open a terminal and connect to your Ambari node using the private keypair provided to you:
`ssh -i .ssh/training-keypair.pem centos@<AmbariNode>`
2. Assume root super user
`sudo su -`
3. Type the following command
`export SPARK_HOME=/usr/hdp/current/spark2-client`

Configure the spark.executor.memory in Zeppelin

1. Connect to the external IP of your Ambari node on port 9995
2. Login as admin/admin

Login

User Name

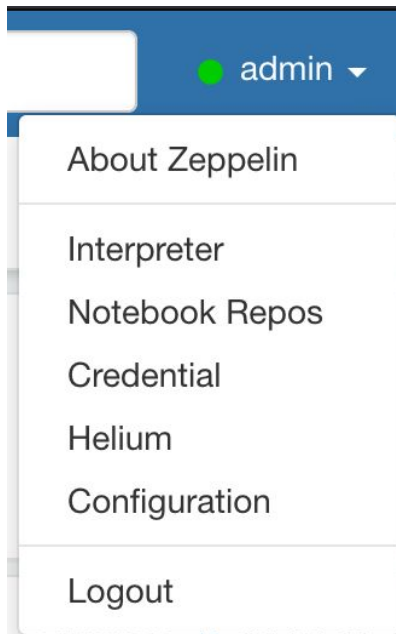
User Name

Password

Password

Login

3. In the Zeppelin UI select the interpreter option of the admin menu



4. Select the spark interpreter by using the search bar in the top

Interpreters

Manage interpreters settings. You can create / edit / remove settings. Note can bind / unbind these interpreter settings.

spark2 %spark2, %sql, %dep, %pyspark, %ipyspark, %r ●

Option

The interpreter will be instantiated in process ⓘ

☐ Connect to existing process

☐ Set permission

Properties

name	value
SPARK_HOME	/usr/hdp/current/spark2-client/
args	
master	yarn-client
spark.app.name	Zeppelin

5. Make the interpreter editable by clicking the edit button

spark2 %spark2, %sql, %dep, %pyspark, %ipyspark, %r ●

6. Find the property spark.executor memory and enter '6300m' in the adjacent text box.

spark.app.name	<input type="text" value="Zeppelin"/>	<input type="button" value="x"/>
spark.cores.max	<input type="text"/>	<input type="button" value="x"/>
spark.executor.memory	<input type="text" value="6300m"/>	<input type="button" value="x"/>

7. Click on the 'Save' button

Result

You have now:

Installed all the packages and data to run the Zeppelin notebooks for this class.

Introduction to Spark REPLs and Zeppelin

About This Lab

Objective:

Access and browse Spark REPLs and Zeppelin

File Locations:

N/A

Successful Outcome:

Use Spark REPLs and browse Zeppelin

Before You Begin:

Complete the Pre-Lab and confirm cluster operation

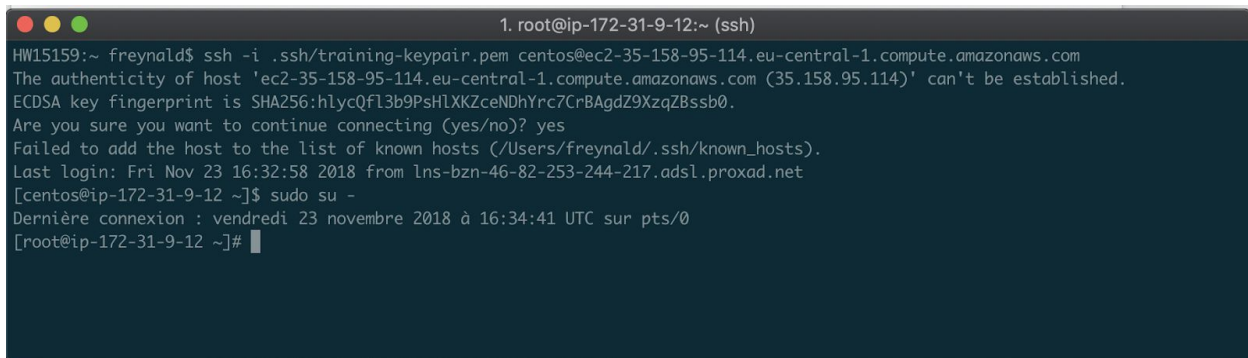
Lab Steps

Perform the following steps:

1. Access the Spark REPLs.

- a. Open a Terminal and ssh to the Zeppelin node on your cluster.

```
# ssh -i ~/.ssh/training-keypair.pem centos@<zeppelinNodeExternalIP>
```



```
1. root@ip-172-31-9-12:~ (ssh)
HW15159:~ freynald$ ssh -i .ssh/training-keypair.pem centos@ec2-35-158-95-114.eu-central-1.compute.amazonaws.com
The authenticity of host 'ec2-35-158-95-114.eu-central-1.compute.amazonaws.com (35.158.95.114)' can't be established.
ECDSA key fingerprint is SHA256:hlycQf13b9PsHlXKZceNDhYrc7CrBAgdZ9XzqZBssb0.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/Users/freynald/.ssh/known_hosts).
Last login: Fri Nov 23 16:32:58 2018 from lns-bzn-46-82-253-244-217.adsl.proxad.net
[centos@ip-172-31-9-12 ~]$ sudo su -
Dernière connexion : vendredi 23 novembre 2018 à 16:34:41 UTC sur pts/0
[root@ip-172-31-9-12 ~]#
```

- b. Switch to the Zeppelin user

```
# su - zeppelin
```

- c. Run the Spark REPL for Scala.

```
# spark-shell
```



```
1. root@ip-172-31-9-12:~ (ssh)
Spark session available as 'spark'.
Welcome to

 _   _ 
/_ \_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\ version 2.3.1.3.0.1.0-187
/_ \_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_/_
/_ \_/_/_/_/_/_/_/_/_/_/_/_/_/__\_/_

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala> |
```

d. View the values for the `SparkContext`, `appName`, and `version`.

```
scala> sc
scala> sc.appName
scala> sc.version
```

- e. Exit the Spark Scala REPL.

```
scala> :quit
```

```
[root@ip-172-31-9-12 ~]# spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
18/11/06 16:47:31 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Spark context Web UI available at http://ip-172-31-9-12.eu-central-1.compute.internal:4041
Spark context available as 'sc' (master = yarn, app id = application_1541510488826_0004).
Spark session available as 'spark'.
Welcome to

      ____
     /__/\_  _  ____/_/\_
    _\  \_  \_  _  _/\_  '\_
   /\_/\_ ._\^\. ,/_/\_/\^\. version 2.3.1.3.0.1.0-187
      /\

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@470f0637

scala> sc.appName
res1: String = Spark shell

scala> sc.version
res2: String = 2.3.1.3.0.1.0-187

scala> :quit
[root@ip-172-31-9-12 ~]#
```

- f. Run the Spark REPL for Python.

```
# pyspark
```

g. View the values for the `SparkContext`, `appName`, and `version`.

>>> SC

```
>>> sc.appName
```

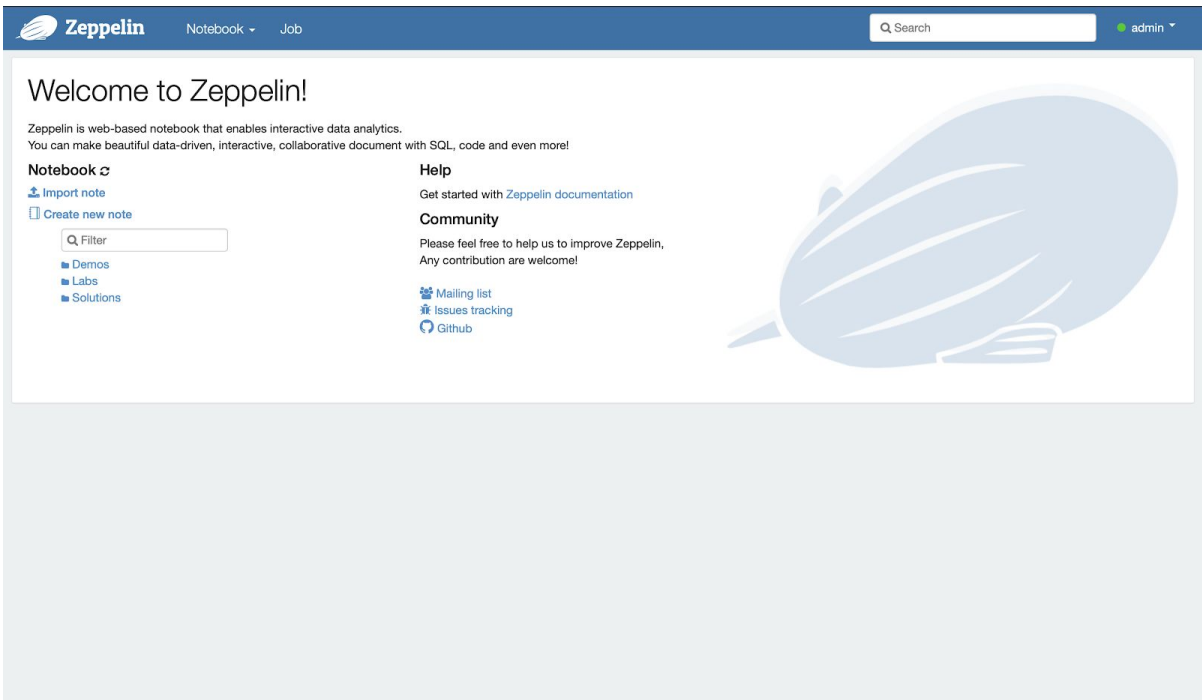
```
>>> sc.version
```

h. Exit the Spark Python REPL.

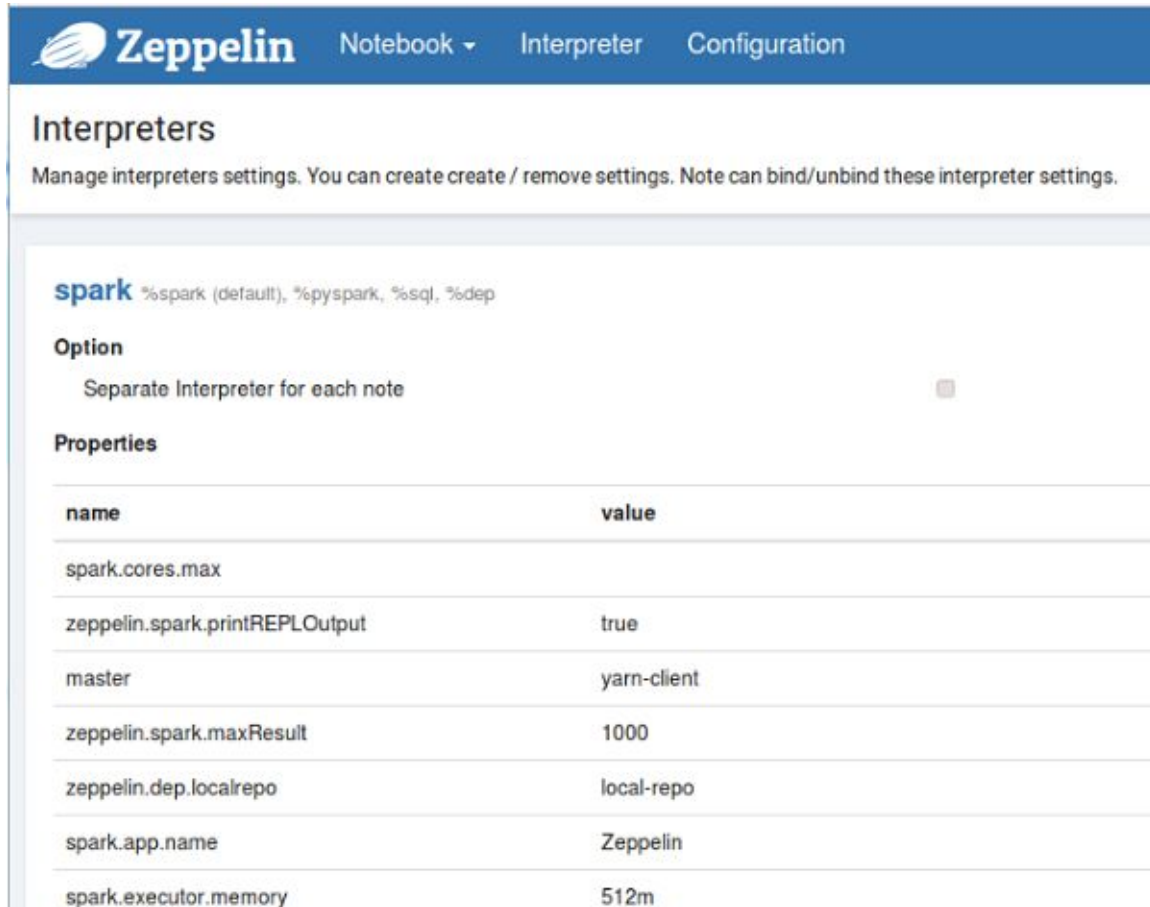
```
>>> exit()
```

2. Access and browse Zeppelin.

- a. Open the Firefox browser and enter the URL to view the Zeppelin UI:
<http://ZeppelinNodeExternalIP:9995/>



- b. Click Interpreter in the top menu and note that Zeppelin's default interpreter is set to Spark and has a number of default settings configured.



Zeppelin Notebook Interpreter Configuration

Interpreters

Manage interpreters settings. You can create create / remove settings. Note can bind/unbind these interpreter settings.

spark %spark (default), %pyspark, %sql, %dep

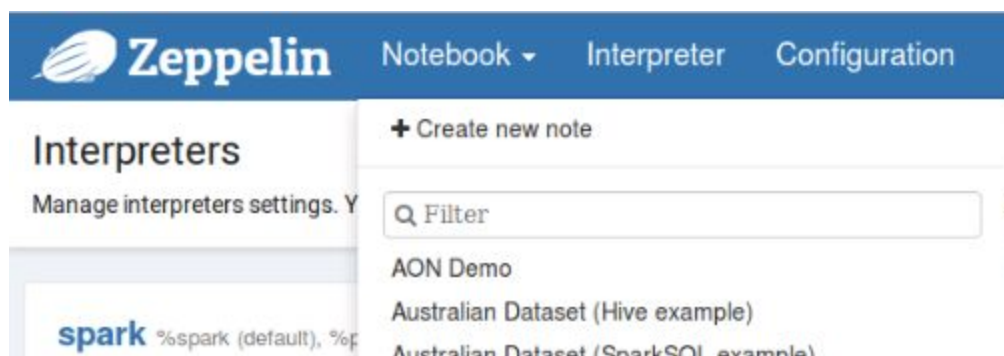
Option

Separate Interpreter for each note ☐

Properties

name	value
spark.cores.max	
zeppelin.spark.printREPLOutput	true
master	yarn-client
zeppelin.spark.maxResult	1000
zeppelin.dep.localrepo	local-repo
spark.app.name	Zeppelin
spark.executor.memory	512m

- c. Click on Notebook in the top menu and select Create new note from the resulting drop down options.



Zeppelin Notebook Interpreter Configuration

Interpreters

Manage interpreters settings. Y

spark %spark (default), %p

Notebook Interpreter Configuration

+ Create new note

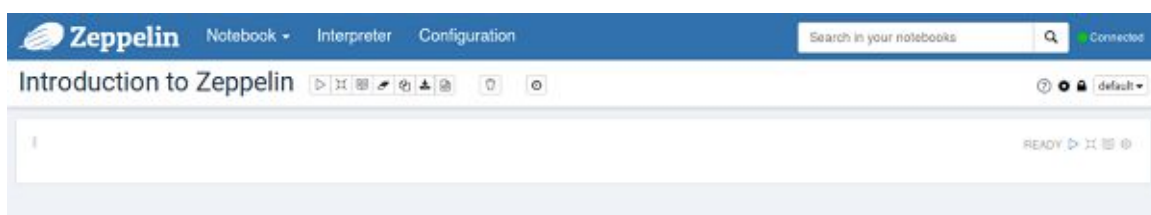
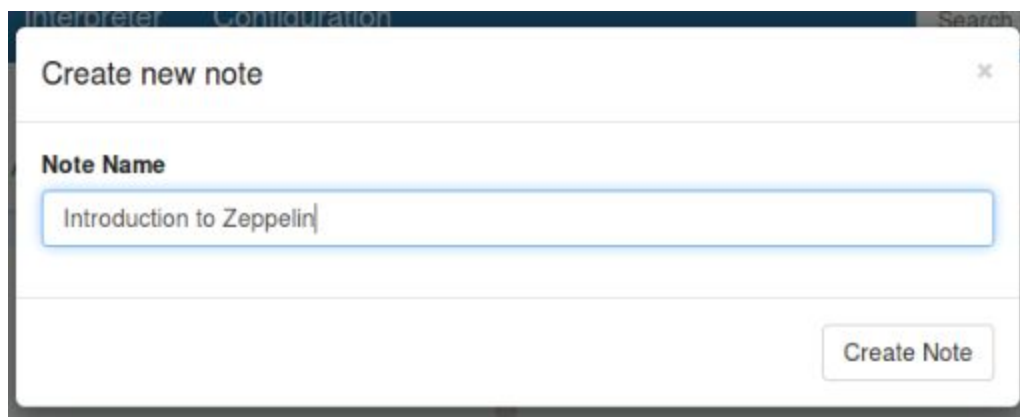
Q Filter

AON Demo

Australian Dataset (Hive example)

Australian Dataset (SparkSQL example)

- d. Name this note Introduction to Zeppelin and click Create Note.



- e. Find the values for Spark version and the Spark home directory. When you type the commands, run them either by pressing the Shift + Enter keys, or by clicking on the Play icon to the right of the word Ready.

Note: The first time this is run, it may take a few minutes to complete. Future commands will run much faster, including this one if repeated.

```
sc.version
```



While processing, Zeppelin will display a status of RUNNING. It will also display a Pause icon should it become necessary.

The output may vary slightly from the screenshot below, but should look something like this when processing is completed:



- f. Zeppelin can be instructed to use multiple languages in an interactive fashion within the same notebook. Simply specify the desired language prior to the command.

Run the following commands to demonstrate this flexibility using Shell, Python, Scala, Markdown, and Spark SQL. Execute each command by clicking on the Play icon or pressing Shift + Enter when you are finished typing.

Shell:

```
%sh echo "Introduction to Zeppelin"
```

```
%sh echo "Introduction to Zeppelin"
Introduction to Zeppelin
```

FINISHED ▶ ⌵ ⌶ ⌵

Python:

```
%pyspark
print('Introduction to Zeppelin')
```

```
%pyspark
print('Introduction to Zeppelin')
Introduction to Zeppelin
```

FINISHED ▶ ⌵ ⌶ ⌵

Scala (default, so no need to specify prior to running command):

```
val s = "Introduction to Zeppelin"
```

```
val s = "Introduction to Zeppelin"
s: String = Introduction to Zeppelin
```

FINISHED ▶ ⌵ ⌶ ⌵

Markdown:

```
%md Introduction to Zeppelin
```

```
Introduction to Zeppelin
```

FINISHED ▶ ⌵ ⌶ ⌵

Spark SQL:

```
%sql
show tables
```

```
%sql
show tables
```



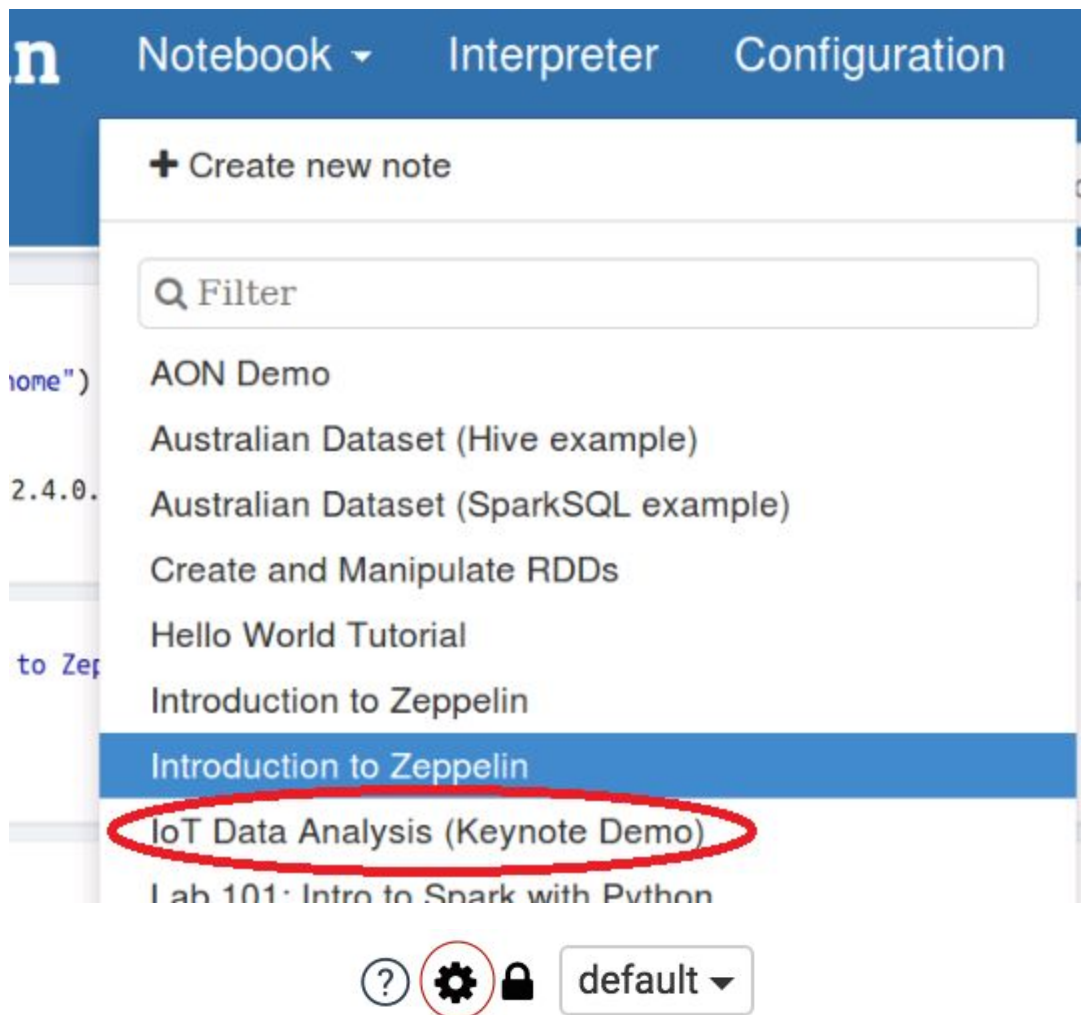
tableName

isTemporary

3. Use a preconfigured notebook to browse Zeppelin's capabilities.

- a. Zeppelin has four major functions: data ingestion, data discovery, data analytics, and data visualization. One of the easiest ways to explore these functions is with a preconfigured notebook, many of which are available by default.

Click on Notebook at the top of the browser window and find and select the notebook labeled IoT Data Analysis (Keynote Demo) in the resulting drop-down menu.



- b. For the purposes of this lab, all necessary code has already been entered for you in the saved notebook. All you have to do is scroll to the appropriate section and click the Play icon or press Shift + Enter.

IoT Data Analysis (Keynote Demo)

Using Zeppelin for Data Science Tasks: Data Ingestion, Data Formatting, Exploratory Analysis and Model Building.

Data Science involves a typical sequence of tasks: acquiring data, cleaning it, analyzing it for relationships, and then building a model. Zeppelin allows you to do all these from one unified interface.

Task 2 seconds

First, let's load the data into HDFS and make sure we can access it.

Task 3 seconds

```
%sh
whoami

curl -sSL -O "https://www.dropbox.com/s/ggjitrobwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip

hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/

hadoop fs -ls /user/zeppelin/iotdemo/

zeppelin
Archive: iotdemo-notebook-data.zip
Found 2 items
```

- c. The first major block of code ingests data from an online source into HDFS and then displays those files using the shell scripting interpreter. Find and run that code.

Note that the label to the left of the Play icon says FINISHED, but this will not prohibit you from running the code again on this machine.

Also note: this notebook uses a deprecated command, `hadoop fs`, rather than the more updated `hdfs dfs` command we used in the previous lab. This should not affect the functionality of the demo.

```
%sh
whoami

curl -sSL -O "https://www.dropbox.com/s/ggjitrobwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip

hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/

hadoop fs -ls /user/zeppelin/iotdemo/

zeppelin
Archive: iotdemo-notebook-data.zip
Found 2 items
```

When the code has finished, the output at the bottom should look like this:

```
hadoop fs -ls /user/zeppelin/iotdemo/

zeppelin
Archive: iotdemo-notebook-data.zip
Found 2 items
-rw-r--r--  3 zeppelin zeppelin    63570 2016-05-27 16:50 /user/zeppelin/iotdemo/enrichedEvents
-rw-r--r--  3 zeppelin zeppelin    33084 2016-05-27 16:50 /user/zeppelin/iotdemo/trainingData
```

- d. The next section of the notebook once again uses the shell scripting interpreter to view some of the raw data in one of the downloaded files. Scroll down and run this code, then view its output.


```
%sh
hadoop fs -cat /user/zeppelin/iotdemo/enrichedEvents | tail -n 10

Overspeed,"Y","hours",45,2773,-90.07,35.68,0,1,1
Lane Departure,"Y","hours",45,2773,-90.04,35.19,1,1,0
Normal,"Y","hours",45,2773,-90.68,35.12,1,0,0
Normal,"Y","hours",45,2773,-91.14,34.96,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
Normal,"Y","hours",45,2773,-92.31,34.78,0,1,0
Normal,"Y","hours",45,2773,-92.09,34.8,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
Normal,"Y","hours",45,2773,-90.68,35.12,0,0,0
```

- e. The next section of the notebook performs actions necessary to import and use this data with Spark SQL. You may note that the status to the left of the Play icon is shown as ERROR. This is due to the fact that the file being manipulated did not exist at the time the notebook was opened on this system. Run this code and view the output.

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val eventsFile = sc.textFile("hdfs:///user/zeppelin/iotdemo/enrichedEvents")

case class Event(eventType: String,
                 isCertified: String,
                 paymentScheme: String,
                 hoursDriven: Int,
                 milesDriven: Int,
                 lat: Float,
                 long: Float,
                 isFoggy: Int,
                 isRainy: Int)
```

The output should look like this:

```
eventsRDD.toDF().registerTempTable("enrichedEvents")

sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@312d2a12
eventsFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at <console>:29
defined class Event
eventsRDD: org.apache.spark.rdd.RDD[Event] = MapPartitionsRDD[5] at map at <console>:35
res4: Long = 1359
```

- f. The next block of code utilizes Spark SQL to view this data. Run this code and examine the output.

```
%sql
select * from enrichedEvents order by hoursDriven desc limit 10
```

FINISHED

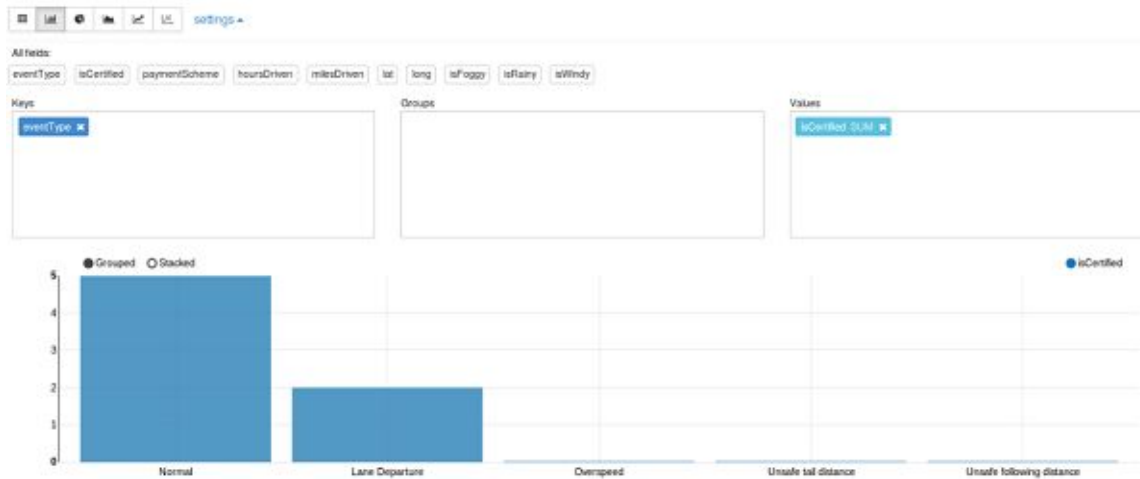
Visualizations: Table, Bar, Pie, Line, Area, Map

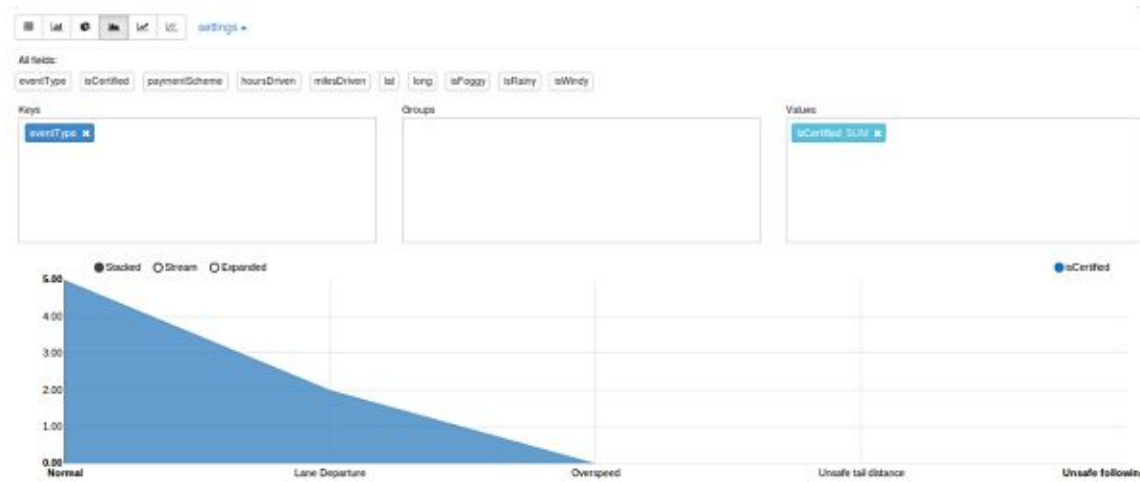
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy
Normal	N	miles	90	4,300	-90.29	40.96	0	0
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1

- g. Note that at the top of the results there are six buttons that allow you to display the results using six different visualizations. Click on each one to view the differences between them.

Visualizations: Table, Bar, Pie, Line, Area, Map

eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy	isWindy
Normal	N	miles	90	4,300	-90.29	40.96	0	0	1
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1	1
Normal	N	miles	90	4,300	-89.91	40.86	0	0	0
Overspeed	N	miles	90	4,300	-93.04	41.71	1	0	0
Unsafe tail distance	N	miles	90	4,300	-87.67	41.87	1	1	1
Normal	N	miles	90	4,300	-89.52	40.7	0	0	0
Normal	N	miles	90	4,300	-91.05	41.72	0	0	1
Normal	N	miles	90	4,300	-91.47	41.74	0	0	0
Lane Departure	N	miles	90	4,300	-91.59	41.7	1	0	0







TIP: In this lab you ran each section of code, known as a paragraph, individually. The entire notebook could have been played at once, however, by clicking the Play icon labeled Run all paragraphs directly to the right of the notebook title at the top of the browser.



Result

You have accessed the Spark REPLs for both Scala and Python, created a Zeppelin notebook and demonstrated Zeppelin's ability to interpret multiple languages, and used a pre-built Zeppelin notebook to briefly explore Zeppelin's ability to ingest, view, analyze, and visualize data.