Developer Training for Spark and Hadoop I: Hands-On Exercises

General Notes

Cloudera's training courses use a Virtual Machine running the CentOS Linux distribution. This VM has CDH (Cloudera's Distribution, including Apache Hadoop) installed in Pseudo-Distributed mode. Pseudo-Distributed mode is a method of running Hadoop whereby all Hadoop daemons run on the same machine. It is, essentially, a cluster consisting of a single machine. It works just like a larger Hadoop cluster, the only key difference (apart from speed, of course!) being that the block replication factor is set to 1, since there is only a single DataNode available.

Getting Started

1. Before starting the exercises, run the course setup script in a terminal window:

```
$ $DEV1/scripts/training_setup_dev1.sh
```

   This script will enable services and set up any data required for the course. You must run this script before starting the Hands-On Exercises.

Working with the Virtual Machine

1. The VM is set to automatically log in as the user `training`. Should you log out at any time, you can log back in as the user `training` with the password `training`.

2. Should you need it, the root password is `training`. You may be prompted for this if, for example, you want to change the keyboard layout. In general, you should not need this password since the `training` user has unlimited sudo privileges.

3. In some command-line steps in the exercises, you will see lines like this:

```
$ hdfs dfs -put shakespeare \
/user/training/shakespeare
```

The dollar sign (**$**) at the beginning of each line indicates the Linux shell prompt. The actual prompt will include additional information (e.g., `[training@localhost workspace]$`) but this is omitted from these instructions for brevity.

The backslash (`\`) at the end of the first line signifies that the command is not completed, and continues on the next line. You can enter the code exactly as shown (on two lines), or you can enter it on a single line. If you do the latter, you should *not* type in the backslash.

Points to note during the exercises

1. The main directory for the exercises for this course is `$DEV1/exercises` (`~/training_materials/dev1/exercises`). Each directory under that one corresponds to an exercise or set of exercises – this is referred to in the instructions as "the exercise directory". Any scripts or files required for the exercise (other than data) are in the exercise directory.

2. Within each exercise directory you may find the following subdirectories:

    a. `solution` – Solution code for each exercise.

    b. `stubs` – A few of the exercises depend on provided starter files containing skeleton code.

    c. Maven project directories – For exercises for which you must write Scala classes, you have been provided with preconfigured Maven project directories. Within these projects are two packages: `stubs`, where you will do your work using starter skeleton classes; and `solution`, containing the solution class.

3. Data files used in the exercises are in `$DEV1DATA` (`~/training_materials/data`). Usually you will upload the files to HDFS before working with them.

4. As the exercises progress, and you gain more familiarity with Hadoop and Spark, we provide fewer step-by-step instructions; as in the real world, we merely give you a requirement and it's up to you to solve the problem! You should feel free to refer to the solutions provided, ask your instructor for assistance, or consult with your fellow students!

5. There are additional bonus exercises for some of the exercises. If you finish the main exercise, please attempt the additional steps.

Catching Up

Most of the exercises in this course build on prior exercises. If you are unable to complete an exercise (even using the provided solution files), or if you need to catch up to the current exercise, run the provided catch up script:

```
$ $DEV1/scripts/catchup.sh
```

The script will prompt for which exercise you are starting; it will set up all the data required as if you had completed all the exercises.

Warning: If you run the catch up script, you will lose all your work! (e.g. All data will be deleted from HDFS, tables deleted from Hive, etc.)

Hands-On Exercise: Create and Populate Tables in Impala or Hive

Files and Directories Used in this Exercise

Exercise directory: `$DEV1/exercises/impala`

MySQL Database: `loudacre`

MySQL Tables: `device`

Data files (HDFS): `/loudacre/webpage`

In these exercises you will define Impala/Hive tables to model and view data in HDFS.

Note: The accounts data will not be used in this exercise but will in a subsequent exercise.

You may perform this and subsequent exercises in either Impala or Hive. Most of the instructions are the same whichever tool you choose; where the instructions differ, the difference is noted. Following whichever set of instructions you prefer; if you have no preference, we suggest Impala because it is faster.

Create and Query a Table in Impala or Hive

In this section you will use Impala or Hive to query the webpage data you imported in the previous exercise.

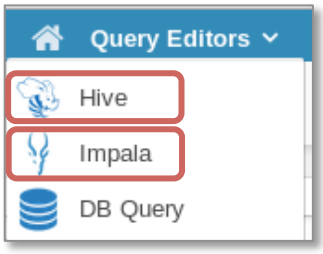
There are a number of ways to interact with Impala and Hive. In this exercise you will use the Impala or Hive Query Editor in Hue.

1. If you plan to use Hive rather than Impala for this or subsequent exercises, start the Hive server, which is not started by default, by entering the following two commands in a terminal window:

```
$ sudo service zookeeper-server start
$ sudo service hive-server2 start
```

2. Visit the Hue page in Firefox, as described earlier in the "Using HDFS" exercise.

3. Open either the Impala query editor or Hive query editor, by selecting the editor of your choice from the **Query Editors** menu.



4. In the query editor pane on the right, enter an SQL command to create an table for the webpage data imported in previous exercises:

```
CREATE EXTERNAL TABLE webpage
   (page_id SMALLINT,
    name STRING,
    assoc_files STRING)
   ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      LOCATION '/loudacre/webpage'
```

5. Click the **Execute** button to execute the command.

6. To see the table you just created, refresh the table list on the left.

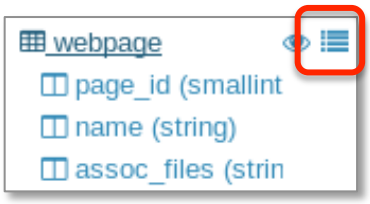
7. Click on the **webpage** table to see the column definitions.

8. Click the **New Query** button, then enter and execute a test query such as:

```
SELECT * FROM webpage WHERE name LIKE "ifruit%"
```

The resulting data is shown in the **Results** tab of the pane below the query.

9. Click on the Preview Sample Data icon.



Use Sqoop to Import Directly into Hive and Impala

In this section you will use Sqoop to import data from MySQL into HDFS and also automatically create the corresponding table in the Hive Metastore.

10. In a terminal window, import the `device` table directly into the Hive Metastore.

```
$ sqoop import \
  --connect jdbc:mysql://localhost/loudacre \
  --username training --password training \
  --fields-terminated-by '\t' \
  --table device \
  --hive-import
```

Use `--hive-import` for either Impala or Hive; this adds metadata to the Metastore, which both tools use.

Note: You may get a warning message that `chgrp` is unable to change the ownership of the generated files; you can disregard the warning, it does not affect the import.

11. Using Hue or the HDFS command line, review the imported data files. The Hive import copies the data to the default Hive warehouse location: `/user/hive/warehouse/device`.

12. If you are using Impala, refresh the Impala metadata cache by entering the command in the Hue Impala Query Editor:

```
INVALIDATE METADATA
```

13. As in the previous exercise, view the columns and execute a test query:

```
SELECT * FROM device LIMIT 10;
```

This is the end of the Exercise

Hands-On Exercise: Select a Format for a Data File

Files and Data Used in this Exercise

Exercise directory: `$DEV1/exercises/data-format/`

MySQL Database: `loudacre`

MySQL Table: `accounts`

In this exercise, you will use import data in Avro format and create an Impala/Hive table to access it.

1. Change directories to the exercise directory (`$DEV1/exercises/data-format/`).

2. Import the `accounts` table to an Avro data format.

```
$ sqoop import \
  --connect jdbc:mysql://localhost/loudacre \
  --username training --password training \
  --table accounts \
  --target-dir /loudacre/accounts_avro \
  --null-non-string '\\N' \
  --as-avrodatafile
```

3. View the files imported by Sqoop into HDFS. What do you see when you try to view the content of the data files?

4. Sqoop will generate a schema in the current directory: `sqoop_import_accounts.avsc`. Review this file and copy it to HDFS `/loudacre/`.

5. In Impala or Hive, create a table using this schema:

```
CREATE EXTERNAL TABLE accounts_avro
STORED AS AVRO
LOCATION '/loudacre/accounts_avro'
TBLPROPERTIES ('avro.schema.url'=
'hdfs:/loudacre/sqoop_import_accounts.avsc');
```

6. Confirm correct creation of the table by issuing a query such as

```
SELECT * FROM accounts_avro LIMIT 10
```

7. Optional: Use the `DESCRIBE` or `DESCRIBE FORMATTED` command to list the columns and data types of the `accounts_avro` table created from the Avro schema.

Bonus Exercise

8. Create a new table based on the existing `accounts_avro` table data, using Parquet for the storage format.

This is the end of the Exercise

Hands-On Exercise: Partition Data in Impala or Hive

Files and Data Used in this Exercise

Exercise directory: `$DEV1/exercises/data-partition`

Data files (HDFS): `/loudacre/accounts_avro`

In this exercise you will create and load an Impala/Hive table with account data, partitioned by area code.

In the previous exercise you imported data from the accounts table using Sqoop, into a table called `accounts_avro`. In this exercise, you will create a new table with some of the account data, partitioned by area code (the first three digits of the phone number).

1. Create a new, empty table in Impala or Hive:

```
CREATE EXTERNAL TABLE accounts_by_areacode (
    acct_num INT,
    first_name STRING,
    last_name STRING,
    phone_number STRING)
  PARTITIONED BY (areacode STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/loudacre/accounts_by_areacode';
```

2. In order to populate the new table, you will need to extract the area code from the phone number. Try executing the following query to demonstrate:

```
SELECT acct_num, first_name, last_name,
   phone_number, SUBSTR(phone_number,1,3) AS areacode
  FROM accounts_avro
```

3. Use the `SELECT` statement above in an `INSERT INTO TABLE` command to copy the specified columns to the new table, dynamically partitioning by area code.

4. Execute a simple query to confirm that the table was populated correctly, such as

```
SELECT * FROM accounts_by_areacode LIMIT 10
```

5. Using Hue or the `hdfs` command line interface, confirm that the directory structure of the `accounts_by_areacode` table includes partition directories. Review the data in the directories to verify that the partitioning is correct.

This is the end of the Exercise

Appendix A: Enabling iPython Notebook

iPython Notebook is installed on the VM for this course. To use it instead of the command-line version of iPython, follow these steps:

1. Open the following file for editing: `/home/training/.bashrc`

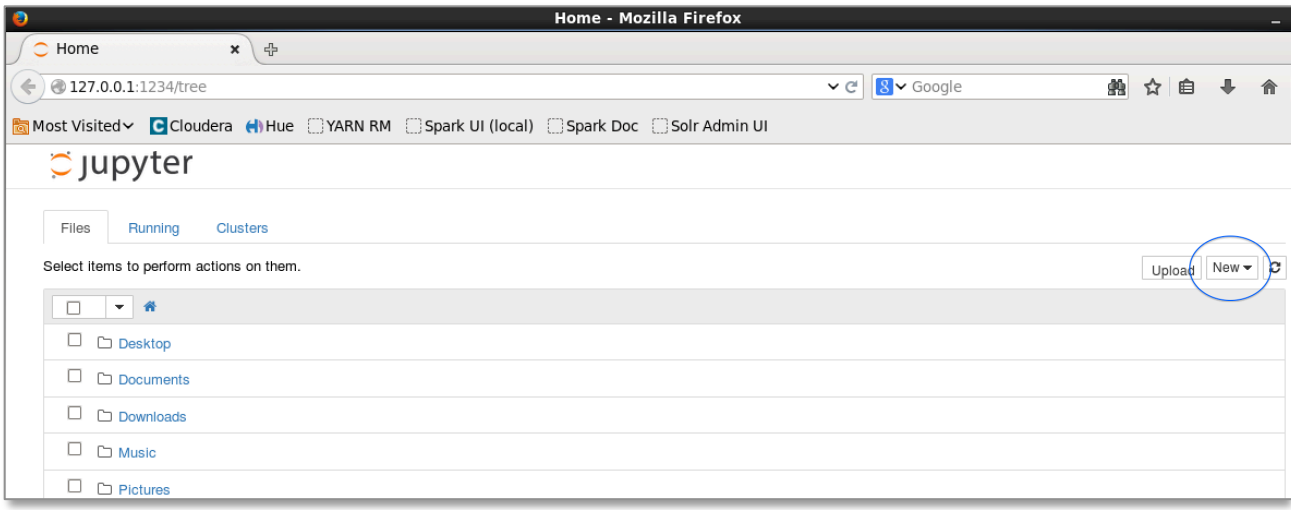
2. Uncomment out the following line (remove the leading #).

```
# export PYSPARK_DRIVER_PYTHON_OPTS='notebook ……..jax'
```

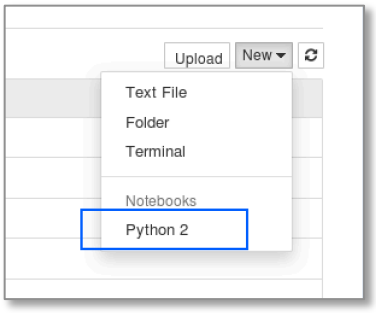
3. Save the file.

4. Open a new terminal window. (Must be a new terminal so it loads your edited `.bashrc` file).

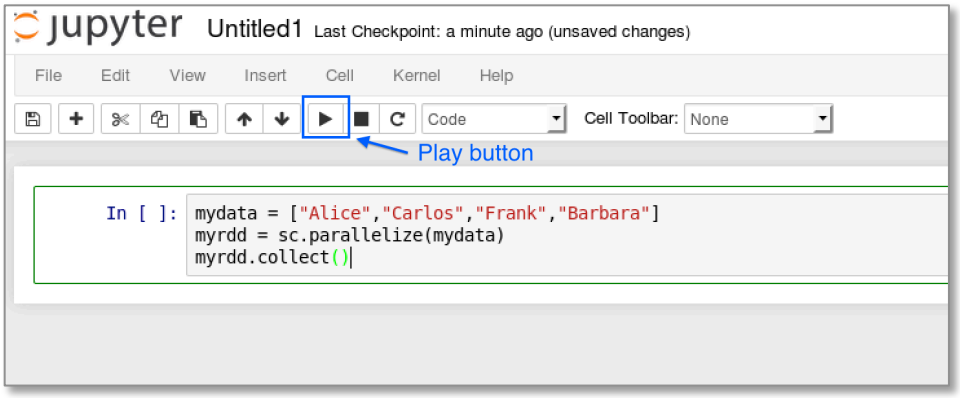
5. Enter `pyspark` in the terminal. This will cause a browser window to open, and you should see the following web page:



6. On the right hand side of the page select **Python 2** from the **New** menu



7. Enter some spark code such as the following and use the play button to execute your spark code.



8. Notice the output displayed.