Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

Files and Data Used in This Exercise	
Exercise directory	$DEVSH/exercises/sqoop
MySQL database	loudacre
MySQL table	basestations
HDFS paths	/loudacre/basestations_import /loudacre/basestations_import_parquet

In this exercise, you will import tables from MySQL into HDFS using Sqoop.

Important: This exercise depends on a previous exercise: "Accessing HDFS with Command Line and Hue." If you did not complete that exercise, run the course catch-up script and advance to the current exercise:

```
$ $DEVSH/scripts/catchup.sh
```

Import a Table from MySQL to HDFS

You can use Sqoop to look at the table layout in MySQL. With Sqoop, you can also import the table from MySQL to HDFS.

1. If you do not already have one started, open a gateway terminal window.
2. Run the sqoop help command to familiarize yourself with the options in Sqoop:

```
$ sqoop help
```

3. List the tables in the loudacre database:

```
$ sqoop list-tables \
--connect jdbc:mysql://localhost/loudacre \
--username training --password training
```

4. Run the sqoop help import command to see its options:

```
$ sqoop help import
```

5. Use Sqoop to import the basestations table in the loudacre database and save it in HDFS under /loudacre:

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

The --null-non-string option tells Sqoop to represent null values as \N, which makes the imported data compatible with Hive and Impala. Single quotes must be used around the \\N character. Using double quotes will result in an "illegal escape character" error from Sqoop.

6. Optional: While the Sqoop job is running, try viewing it in the Hue Job Browser or YARN Web UI.

View the Imported Data

Sqoop imports the contents of the specified tables to HDFS. You can use the Linux command line or the Hue File Browser to view the files and their contents.

7. List the contents of the basestations_import directory:

```
$ hdfs dfs -ls /loudacre/basestations_import
```

- **Note: Output of Hadoop jobs is saved as one or more "partition" files.**

8. Use either the Hue File Browser or the -tail option to the hdfs command to view the last part of the file for each of the MapReduce partition files, for example:

```
$ hdfs dfs -tail /loudacre/basestations_import/part-m-00000
$ hdfs dfs -tail /loudacre/basestations_import/part-m-00001
$ hdfs dfs -tail /loudacre/basestations_import/part-m-00002
$ hdfs dfs -tail /loudacre/basestations_import/part-m-00003
```

Import a Table Using an Alternate File Format

9. Import the basestations table to a Parquet data format rather than the default file form (text file).

```
$ sqoop import \
--connect jdbc:mysql://localhost/loudacre \
--username training --password training \
--table basestations \
--target-dir /loudacre/basestations_import_parquet \
--as-parquetfile
```

10. View the results of the import command by listing the contents of the basestations_import_parquet directory in HDFS, using either Hue or the hdfs command. Note that the Parquet files are each given unique names, such as e8f3424e-230d-4101-abba-66b521bae8ef.parquet.
11. You can't directly view the contents of the Parquet files because they are binary files rather than text. Use the parquet-tools head command to view the first few records in the set of data files imported by Sqoop.

```
$ parquet-tools head \
  hdfs://localhost/loudacre/basestations_import_parquet/
```

This is the end of the exercise.

GITHUB:

Create a new repository in your github and name it data_ingest. Create a folder and name it Sqooq. Provide a markup with answers to the following.

Please provide screenshots of the following.

1. From the accounts table, import only the primary key, along with the first name, last name to HDFS directory /loudacre/accounts/user_info. Please save the file in text format with tab delimiters.

   Hint: You will have to figure out what the name of the table columns are in order to complete this exercise.

2. This time save the same in parquet format with snappy compression. Save it in /loudacre/accounts/user_compressed. Provide.a screenshot of HUE with the new directory created.

3. Finally save in /loudacre/accounts/CA only clients whose state is from California. Save the file in avro format and compressed using snappy. From the terminal, display some of the records that you just imported. Take a screenshot and save it as CA_only.

Hands-On Exercise: Collecting Web Server Logs with Apache Flume

Files and Data Used in This Exercise	
Exercise directory	$DEVSH/exercises/flume
Data files (local)	$DEVDATA/weblogs

In this exercise, you will run a Flume agent to ingest web log data from a local directory to HDFS.

Apache web server logs are generally stored in files on the local machines running the server. In this exercise, you will simulate an Apache server by placing provided web log files into a local spool directory, and then use Flume to collect the data.

Both the local and HDFS directories must exist before using the spooling directory source.

Create an HDFS Directory for Flume-Ingested Data

1. In a gateway session on your Get2Cluster remote desktop, create a directory in HDFS called /loudacre/weblogs_flume to hold the data files that Flume ingests:

```
$ hdfs dfs -mkdir -p /loudacre/weblogs_flume
```

Create a Local Directory for Web Server Log Output

2. Create the spool directory into which the web log simulator will store data files for Flume to ingest. On the local Linux filesystem on the gateway node, create the directory /flume/weblogs_spooldir:

```
$ sudo mkdir -p /flume/weblogs_spooldir
```

3. Give all users permission to write to the /flume/weblogs_spooldir directory:

```
$ sudo chmod a+w -R /flume
```

Configure Flume

A Flume agent configuration file has been provided for you: $DEVSH/exercises/flume/spooldir.conf.

Review the configuration file. *You do not need to edit this file.* Take note in particular of the following:

• The *source* is a spooling directory source that pulls from the local /flume/weblogs_spooldir directory.

- The *sink* is an HDFS sink that writes files to the HDFS /loudacre/weblogs_flume directory.
- The *channel* is a memory channel.

Run the Flume Agent

Next, start the Flume agent on the gateway node and copy the files to the spooling directory.

4. Start the Flume agent using the configuration you just reviewed:

```
$ flume-ng agent \
--conf /etc/flume-ng/conf \
--conf-file $DEVSH/exercises/flume/spooldir.conf \
--name agent1 -Dflume.root.logger=INFO,console
```

5. Wait a few moments for the Flume agent to start up. You will see a message like: Component type: SOURCE, name: webserver-log-source started

Simulate Apache Web Server Output

6. Open a separate gateway terminal window. Run the script to place the web log files in the /flume/weblogs_spooldir directory:

```
$ $DEVSH/exercises/flume/copy-move-weblogs.sh \
/flume/weblogs_spooldir
```

This script will create a temporary copy of the web log files and move them to the spooldir directory.

7. Return to the terminal that is running the Flume agent and watch the logging output. The output will give information about the files Flume is putting into HDFS.
8. Once the Flume agent has finished, enter Ctrl+C to terminate the process.
9. Using the command line or Hue File Browser, list the files that were added by the

   Flume agent in the HDFS directory /loudacre/weblogs_flume.

   Note that the files that were imported are tagged with a Unix timestamp corresponding to the time the file was imported, such as FlumeData.1427214989392.

This is the end of the exercise.

CONTINUE TO GITHUB ON NEXT PAGE

GITHUB:

Create a new folder and name it Flume in your data_ingest repository in github.

1. Create a new flume configuration file with the following:

Source	
Type	Netcat
Bind	localhost
Port	44444
Channel	
Type	Memory
Capacity	1000
transactionCapacity	100
Sink	
Type	logger

2. Start the agent.

3. From another terminal start telnet and connect to port 4444. Start typing and you should see the results from the other terminal. Provide a screenshot of your results.

```
$ telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
Hello world! <ENTER>
OK
```

Hands-On Exercise: Producing and Consuming Apache Kafka Messages

In this exercise, you will use Kafka's command line tool to create a Kafka topic. You will also use the command line producer and consumer clients to publish and read messages.

Files and Data Used in This Exercise	
Exercise directory	$DEVSH/exercises/kafka

Creating a Kafka Topic

1. Open a new gateway terminal window on your Get2Cluster remote desktop and create a Kafka topic named weblogs that will contain messages representing lines in Loudacre's web server logs.

```
$ kafka-topics --create \
--zookeeper localhost:2181 \
--replication-factor 1 \
--partitions 1 \
--topic weblogs
```

   You will see the message: Created topic "weblogs".
   Note: If you previously worked on an exercise that used Kafka, you may get an error here indicating that this topic already exists. You may disregard the error.

2. Display all Kafka topics to confirm that the new topic you just created is listed:

```
$ kafka-topics --list \
--zookeeper localhost:2181
```

3. Review the details of the weblogs topic.

```
$ kafka-topics --describe weblogs \
--zookeeper localhost:2181
```

Producing and Consuming Messages

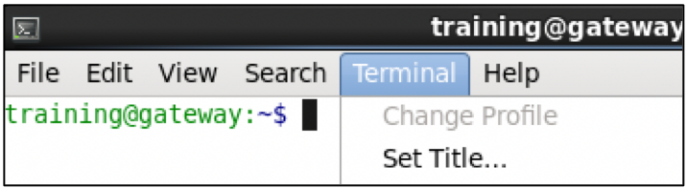
You will now use Kafka command line utilities to start producers and consumers for the topic created earlier.

4. Start a Kafka producer for the weblogs topic:

```
$ kafka-console-producer \
--broker-list localhost:9092 \
--topic weblogs
```

You will see a few SLF4J messages, at which point the producer is ready to accept messages on the command line.

Tip: This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each one by selecting Set Title... on the Terminal menu:



Set the title for this window to "Kafka Producer."

5. Publish a test message to the weblogs topic by typing the message text and then pressing Enter. For example:

```
test weblog entry 1
```

6. Open a new terminal window and adjust it to fit on the window beneath the producer window. Set the title for this window to "Kafka Consumer."
7. In the new gateway terminal window, start a Kafka consumer that will read from the beginning of the weblogs topic:

```
$ kafka-console-consumer \
--zookeeper localhost:2181 \
--topic weblogs \
--from-beginning
```

After a few SLF4J messages, you should see the status message you sent using the producer displayed on the consumer's console, such as:

```
test weblog entry 1
```

8. Press Ctrl+C to stop the weblogs consumer, and restart it, but this time omit the --from-beginning option to this command. You should see that no messages are displayed.
9. Switch back to the producer window and type another test message into the terminal, followed by the Enter key:

```
test weblog entry 2
```

10. Return to the consumer window and verify that it now displays the alert message you published from the producer in the previous step.

Cleaning Up

11. Press Ctrl+C in the consumer terminal window to end its process. 12.
12. Press Ctrl+C in the producer terminal window to end its process.

This is the end of the exercise.

Hands-On Exercise: Sending Messages from Flume to Kafka

Files and Data Used in This Exercise	
Exercise directory	$DEVSH/exercises/flafka
Data files (local)	$DEVDATA/weblogs

In this exercise, you will run a Flume agent on the gateway node that ingests web logs from a local spool directory and sends each line as a message to a Kafka topic.

The Flume agent is configured to send messages to the weblogs topic you created earlier.

Important: This exercise depends on two prior exercises: "Collect Web Server Logs with Flume" and "Produce and Consume Kafka Messages." If you did not complete both of these exercises, run the catch-up script and advance to the current exercise:

```
$ $DEVSH/scripts/catchup.sh
```

Configure a Flume Agent with a Kafka Sink

A Flume agent configuration file has been provided for you on the gateway node:

$DEVSH/exercises/flafka/spooldir_kafka.conf

1. Review the configuration file. You do not need to edit this file. Take note in particular of the following points:

- The *source* and *channel* configurations are identical to the ones in the "Collect Web Server Logs with Flume" exercise: a spooling directory source that pulls from the local /flume/weblogs_spooldir directory, and a memory channel.
- Instead of an HDFS sink, this configuration uses a Kafka sink that publishes messages to the weblogs topic.

Run the Flume Agent

2. Start the Flume agent on the gateway node using the configuration you just reviewed:

```
$ flume-ng agent --conf /etc/flume-ng/conf \
--conf-file $DEVSH/exercises/flafka/spooldir_kafka.conf \
--name agent1 -Dflume.root.logger=INFO,console
```

3. Wait a few moments for the Flume agent to start up. You will see a message like: Component type: SINK, name: kafka-sink started

   Tip: This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each window. Set the title of the current window to "Flume Agent."

Test the Flume Agent Kafka Sink

4. In a new gateway terminal window, start a Kafka consumer that will read from the weblogs topic:

```
kafka-console-consumer \
--zookeeper localhost:2181 \
--topic weblogs
```

   Tip: Set the title of this window to "Kafka Consumer."

5. In a separate new gateway terminal window, run the script to place the web log files in the /flume/weblogs_spooldir directory:

```
$ $DEVSH/scripts/copy-move-weblogs.sh \
/flume/weblogs_spooldir
```

   Note: If you completed an earlier Flume exercise or ran catchup.sh, the script will prompt you whether you want to clear out the spooldir directory. Be sure to enter y when prompted.

6. In the terminal that is running the Flume agent, watch the logging output. The output will give information about the files Flume is ingesting from the source directory.

7. In the terminal that is running the Kafka consumer, confirm that the consumer tool is displaying each message (that is, each line of the web log file Flume is ingesting).

8. Once the Flume agent has finished, enter Ctrl+C in both the Flume agent terminal and the Kafka consumer terminal to end their respective processes.

This is the end of the exercise.