# Lab 2: Hadoop Ecosystem

This lab explores 4 different data pipelines using tools in the Hadoop ecosystem.

## Exercise 1: Kafka + Spark streaming

**Summary**: In this exercise, you will write syslog messages to a Kafka broker. Then you will implement a Spark streaming service that subscribes to the syslog topic in Kafka, consumes those messages in sliding windows, and prints summaries to a file.

**Preparation**:
In advance of getting the final specifications for the submission of this exercise, you will need to perform the following steps:

1. Download, install, and configure Apache Kafka on your MapR sandbox. Note that the Kafka package is not part of the MapR distribution, so you'll need to install and configure it as per the instructions on http://kafka.apache.org. Note that you will need to change the server configuration and/or command-line utilities to use port 5181 (not the default 2181) for the Zookeeper service on MapR.
2. Install the MapR Spark software on your MapR sandbox using the instructions at http://doc.mapr.com. Note: you do not need to install the Spark master or history server for the purpose of this exercise. Launch your Spark jobs using local, yarn-client, or yarn-cluster deployment methods.
3. Create a topic, publish messages to that topic, and consume messages from that topic using the command-line utilities that ship with Kafka.
4. Implement a Spark streaming service that can act as a kafka consumer.

**Final Submission**:

1. Submit your Spark java source file along with a `rebuild.sh` and `rerun.sh` script that I can use to rebuild and rerun the Java code. Assume that I will be running these scripts from the `/user/user01/LAB2_SUBMISSION/E1` directory on my virtual machine.
2. Submit a `start-pipeline.sh` script which runs a Kafka producer for syslog messages. The script should make the following assumptions:
    a. My autograder script will run your `start-pipeline.sh` script as the `user01` user.
    b. Use the '`tail -f`' command on the `/var/log/messages` file to capture messages as they are written to syslog. My cluster will grant permissions to the `user01` user to read from the `/var/log/messages` file.
    c. Send the output of the '`tail -f`' command to your Kafka producer.
    d. Start a Kafka producer that reads syslog messages from `/var/log/messages` and writes them to a topic called "cs286".

e. Kafka will already be installed on my VM under this path. You can use this path or the one you have in your ZIP file for your kafka command.

**`/user/user01/LAB2/E1/KAFKA/kafka_2.10-0.8.2.2`**

3. Youre rebuild.sh script should build a jar called kafka.jar next to the rebuild.sh script.
4. Your `rerun.sh` script should make the following assumptions:
   a. My autograder script will run your `rerun.sh` script as the `user01` user.
   b. The MapR Spark package will already be installed on my cluster in `/opt/mapr/spark/spark-1.4.1`.
   c. Use the "local" deployment mode in your spark-submit command.
   d. Do **NOT** start a Kafka broker. My cluster will already have a Kafka broker running at localhost:9092.
   e. Do **NOT** start a separate zookeeper instance for Kafka. My cluster will already have a zookeeper instance running at localhost:5181.
5. If your solution requires a changed syslog configuration, provide your `/etc/rsyslog.conf` file in your submission.
6. The autograder will be looking at the contents of a file called "`/user/user01/syslog2spark.txt`" for syslog entries generated by the `logger` command (which I will run while your Spark code and Kafka producer are running). Do **NOT** modify the log message that your Spark code receives before writing it to the file. In order to get the Spark output from your program into this file, you simply need to redirect the output from your `spark-submit` command using the '>' character in your `rerun.sh` script.

# Exercise 2: Flume + MapR-FS

**Summary**: In this exercise, you will configure syslog to write its messages to a Flume agent. Then you will configure a Flume agent that writes these syslog messages to a file in MapR-FS.

**Preparation**:
In advance of getting the final specifications for the submission of this exercise, you will need to perform the following steps:

1. Install the MapR Apache Flume software on your MapR sandbox using the instructions at [http://doc.mapr.com](http://doc.mapr.com).
2. Configure a Flume agent to read log messages sent from rsyslog and write them to a file.
3. Configure rsylogd to write messages to a UDP or TCP port and use the logger facility to write messages to syslog.

**Final Submission**:

1. Submit your syslog configuration file (either `/etc/rsyslog.conf` or `/etc/rsyslog.d/50-default.conf` -- whichever approach you used will work for me).
2. Submit your flume agent configuration file called `syslog-flume.conf`.
3. I expect that log messages I generate using the `logger` utility will go to the `/user/user01/syslog2flume` directory. Adjust the sink definition in your `syslog-flume.conf` accordingly.
4. Do NOT modify the log message before writing it to disk.

# Exercise 3: Hive + Sqoop + MySQL

**Summary**: In this exercise, you will write a HiveQL script that calculates the mean value of the petal length of the first species (0.0) from the Iris data set. Then you will use sqoop to pull the data out of the Hive table and puts it into a MySQL table.

**Preparation**:
In advance of getting the final specifications for the submission of this exercise, you will need to perform the following steps:

1. Install and configure Hive, Hive Metastore, and HiveServer2 on your MapR sandbox following the instructions at http://doc.mapr.com.
2. Write a HiveQL script that creates a table, loads data into the table, and calculates the mean value for the petal length column of the first species.
3. Install and configure MySQL server (`yum install mysql-server`) on your MapR sandbox.
4. Install the MapR Sqoop software on your MapR sandbox following the instructions at http://doc.mapr.com.
5. Use Sqoop to pull data out of a Hive table and write into a MySQL table.

**Final Submission**:

1. Submit your HiveQL script called `hive-iris.ql`. This script should read the iris data set from a file called `/user/user01/iris-data.txt` and load the data into a table called `iris_table`. This script should calculate the mean of the petal length for the first species (0.0) and write the results to a directory called `/user/user01/iris_hive_out`.
2. Submit a bash script called `iris-sqoop-2-mysql.sh` which calls the `sqoop` command to read the data from the Hive `iris_table` table and write it to a MySQL table called `iris_table` in the `default` database. Here are the assumptions:
    a. Authenticate your mysql session with username 'user01' and password 'mapr'. I will have already created that user and granted appropriate privileges in my MySQL.
    b. Do **NOT** create the default database -- it will already be there in my MySQL.
    c. Create the `iris_table` table using `mysql` before populating it with `sqoop`.
3. I will run the `mysql` command as the user01 user to submit the following SQL query to check the contents of the table:

```
SELECT * from default.iris_table;
```

# Exercise 4: MapReduce + Oozie

**Summary**: In this exercise, you will write 2 MapReduce programs that run successively, the first of which calculates the mean value and the second of which calculates the standard deviation. Finally, you will construct an Oozie workflow that manages this data pipeline from start to finish.

**NOTE: I have removed the requirement to use a Pig script in this exercise. There is nothing you will need to do to the `iris-data.txt` file to prepare it for this workflow.**

**Preparation**:
In advance of getting the final specifications for the submission of this exercise, you will need to perform the following steps:

1. Install the MapR packages for oozie on your MapR sandbox following the instructions at [http://doc.mapr.com](http://doc.mapr.com). Make sure the run the '`/opt/mapr/server/configure.sh -R`' command as the root user just after running '`yum install`'.
2. Write a MapReduce program that calculates the mean for the petal length of each species in the Iris data set.
3. Write a MapReduce program that calculates the standard deviation for petal length of each species in the Iris data set.
4. Create an Oozie workflow that combines the 2 MapReduce programs into a single workflow.
5. Note that you may run into a bug (where the output of a submitted oozie job says "No FileSystem for scheme: maprfs". If you run into this bug, follow the steps provided at this URL:

http://answers.mapr.com/questions/163554/error-while-running-oozie-on-mapr-410.html

But instead of running step 3 in that write-up, just run:
`/opt/mapr/server/configure.sh -R`.

6. Use the example workflows that ship with the MapR Oozie package. Note that the `job.properties` files in the example ZIP file have not been "upgraded" to work with YARN, so you'll need to make the following change in your `job.properties` file:

from:
`jobTracker=maprfs:///`
to:
`jobTracker=YARN_RESOURCE_MANAGER:8032`

7. Some commands you might find useful:

```
oozie validate workflow.xml
oozie job -log <oozie-job-id>
```

8. You may also find it useful to install the Oozie Web interface, the instructions for which are located here. Note you will need to enable the port 11000 in your NAT port forwarding rules of your VirtualBox VM. Also, the GUI wants you to click the “circular arrow” button next to “All Jobs” to refresh the UI.

http://doc.mapr.com/display/MapR/Manage+Oozie+Services+and+Interface#ManageOozieServicesandInterface-EnablingtheOozieWebUI

**Final Submission**:

1. Submit your 2 MapReduce Java source files along with a `rebuild.sh` script to build your jar file called `Iris.jar`.
2. Submit your Oozie `workflow.xml` and `job.properties` files.
3. The input file `iris-data.txt` will be located in /user/user01
4. I will submit your Oozie workflow on my sandbox (after changing the `jobTracker` property to match my VM hostname) and look in the output directory of the second mapreduce job defined in the `workflow.xml` file to find the standard deviation of the petal length for each of the 3 iris flower species. Define the output directory as “`/user/user01/oozie_iris_stdev`”.

   The output file should look as below:

   ```
   0.0   petal-length-standard-deviation
   1.0   petal-length-standard-deviation
   2.0   petal-length-standard-deviation
   ```

# How to package all your artifacts for submission

This section describes how my autograder script expects to find all the artifacts for the 4 exercises of this lab.

1. Create a ZIP file called `E1.zip` for exercise 1 as follows:
   a. Create a directory called `E1`.
   b. Put your `start-pipeline.sh`, `rebuild.sh`, and `rerun.sh` scripts in `E1`.
   c. Put your Java source file(s) in `E1`.
   d. If you have a custom `rsyslog.conf` file, put it in `E1`.
   e. If you have any other files your solution requires, put them in `E1`.
   f. Create the zip file as follows:
      `zip -r E1.zip E1`

   My autograder script will put your `E1.zip` file in a directory called `/user/user01/LAB2_SUBMISSION`. I will extract your `E1.zip` file in the `/user/user01/LAB2_SUBMISSION` folder and run your scripts from there. If you have a custom `rsyslog.conf` file present in your submission, I will replace the one on my machine with the one you provide and restart the syslog service before running your submission artifacts.

2. Create a ZIP file called `E2.zip` for exercise 2 as follows:
   a. Create a directory called `E2`.
   b. Put your `rsyslog.conf` and `syslog-flume.conf` files in `E2`.
   c. Create the zip file as follows:
      `zip -r E2.zip E2`

3. Create a ZIP file called `E3.zip` for exercise 3 as follows:
   a. Create a directory called `E3`.
   b. Put your `hive-iris.ql` and `iris-sqoop-2-mysql.sh` scripts in `E3`.
   c. Create the zip file as follows:
      `zip -r E3.zip E3`

4. Create a ZIP file called `E4.zip` for exercise 4 as follows:
   a. Create a directory called E4.
   b. Put your `rebuild.sh`, Java source files, `job.properties`, and `workflow.xml` in `E4`.
      Note: your `rebuild.sh` script should build a JAR called `Iris.jar` in the `E4` directory (i.e. “next to” the `rebuild.sh` script).
      Note: your application path that is defined in the `job.properties` should be `/user/user01/LAB2_SUBMISSION/E4`
   c. Create the ZIP file as follows:

```
zip -r E4.zip E4
```

5. Create a ZIP file called `FIRSTNAME_LASTNAME_LAB2_SUBMISSION.zip` for your lab 2 submission as follows:
    a. Create a directory called `FIRSTNAME_LASTNAME_LAB2_SUBMISSION`
    b. Move your `E1.zip`, `E2.zip`, `E3.zip`, and `E4.zip` files into the `FIRSTNAME_LASTNAME_LAB2_SUBMISSION` directory.
    c. Create the submission ZIP file as follows:

```
zip -r FIRSTNAME_LASTNAME_LAB2_SUBMISSION.zip
FIRSTNAME_LASTNAME_LAB2_SUBMISSION
```