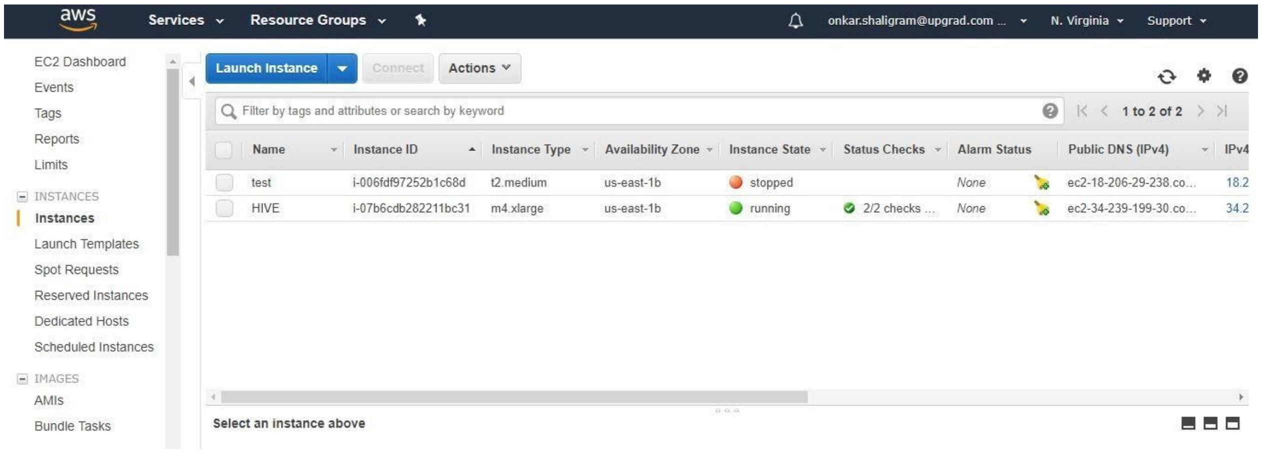
Session 3: MapReduce Programming on AWS EC2

Step 1: Start your AWS EC2 instance by logging in to your AWS Management Console.



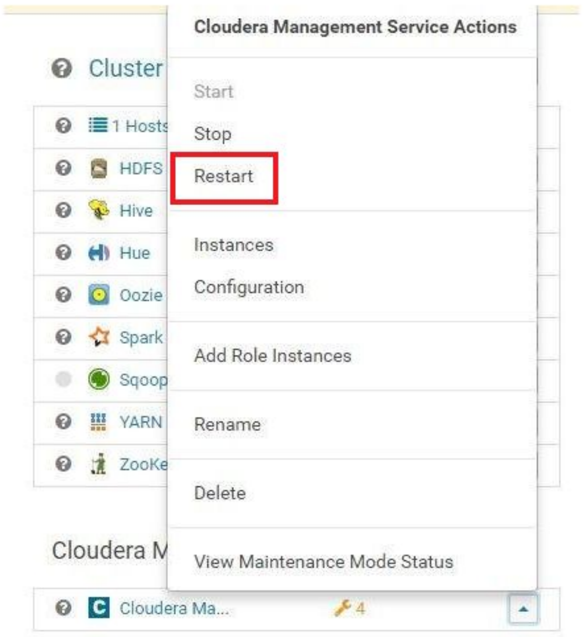
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4
test	i-006fdf97252b1c68d	t2.medium	us-east-1b	stopped		None	ec2-18-206-29-238.co...	18.2
HIVE	i-07b6cdb282211bc31	m4.xlarge	us-east-1b	running	2/2 checks ...	None	ec2-34-239-199-30.co...	34.2

Step 2: Make sure that your instance is up and running fully after Step 1. Next, log in to the Cloudera Manager. It is available at the following link:

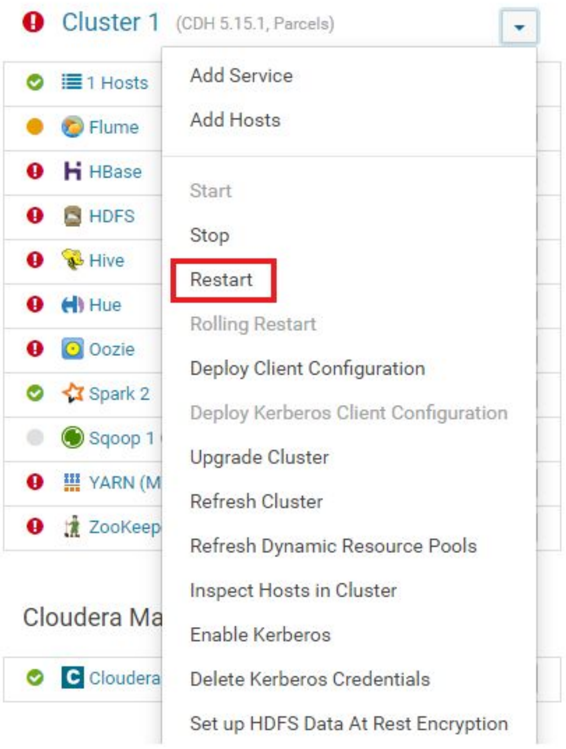
http://<public ip>:7180 [Here, <public ip> will be the ip of your machine such as 34.239.199.30.]

By default, both the username and the password for Cloudera Manager are ‘admin’. (If you have changed them manually, then use the new credentials.)

Step 3a: Restart the Cloudera Management Service



Step 3b: Restart Cluster 1.



Step 4: Connect to AWS EC2 (via PuTTY, etc.). You learnt how to connect to AWS EC2 in the previous modules.

Segment 3: Basic Implementation of MapReduce Using Python

Here, you will learn how to write basic MapReduce scripts for both Mapper and Reducer and simulate a basic MapReduce program using Python. You will be writing a simple MapReduce program to find the maximum age of the customers gender-wise, as discussed in the previous segment.

Step 1: First, you need to make a folder in your local EC2 instance where you will import your data set and store your scripts for ease of use.

```
[ec2-user@ip-10-0-0-113 ~]$ mkdir MapReduce_test
```

Step 2: Now, navigate to this folder.

```
[ec2-user@ip-10-0-0-113 ~]$ cd MapReduce_test
```

Now that you have made a directory to work on, you can start importing the data set to the newly created directory of your instance using WinSCP.

After importing the CustomerAge data set to your directory, you can start writing the Mapper script.

Step 3: Start by making a new file for your Mapper, say **'mapperage.py**', using vi, and then go into input mode by pressing '**I**' on your keyboard.

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ vi mapperage.py
```

Step 4: Now input the following code into the file:

```
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    if len(line) >=2:
        gender = line[1]
        age = line[2]
        print ('%s\t%s' % (gender, age))
```

Step 5: Save the file by going into command mode by pressing Esc, then type the following, and press Enter:

```
:wq!
```

Now, you will write the Reducer script.
Step 6: Make a new file for the Reducer script, say '**reduced_max.py'**, using vi, and go into input mode by pressing '**I**' on your keyboard.

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ vi reduced_max.py
```

Step 7: Now input the following code into the file:

```
#!/usr/bin/python
#Reducer.py
import sys
gender_age = {}

for line in sys.stdin:
    line = line.strip()
    gender, age = line.split('\t')

    if gender in gender_age:
        gender_age[gender].append(int(age))
    else:
        gender_age[gender] = []
        gender_age[gender].append(int(age))

#Reducer
for gender in gender_age.keys():
    max_age = max(gender_age[gender])
    print gender + "\t" + str(max_age)
```

Save the file again by typing '**:wq!**' after going into command mode. With this, you have written the scripts for both Mapper and Reducer for your MapReduce program.

Step 8: Before running the MapReduce program, you need to give execute permissions to your scripts. Run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ chmod 700 mapperage.py
reduced_max.py
```

Step 9: Now, to run the MapReduce job, run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ cat CustomerAge.txt | python
mapperage.py | python reduced_max.py
```

Segment 4: Hadoop Streaming

In this segment, you will learn how Hadoop Streaming can be used to run your MapReduce job on the Hadoop cluster.

Step 1: First, navigate to the test directory that you created in the previous segment, wherein your data set and scripts are stored.

```
[ec2-user@ip-10-0-0-113 ~]$ cd MapReduce_test
```

Step 2: Now, you need to push your data set into the HDFS cluster before you can run the MapReduce job on it. Run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop fs -put CustomerAge.txt
/user/ec2-user/
```

Step 3: Now that you have your data set in the Hadoop cluster, you can run the MapReduce job using Hadoop Streaming. Run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop jar
/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.
15.1.jar \
-file mapperage.py -mapper 'python mapperage.py' \
-file reduced_max.py -reducer 'python reduced_max.py' \
-input /user/ec2-user/CustomerAge.txt \
-output /user/ec2-user/output_1
```

Note: The Hadoop streaming jar that you will be using for your instance may have a different version. You can find out what jar you can use for using Hadoop Streaming by using the **'ls'** command to check out what jar files are present in the '/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/' directory of your instance.

Note: You need to always make sure that the output directory that you are specifying in your MapReduce job is not already present in the HDFS cluster; otherwise, the job will throw an error.

Note: You can also set the number of Reduce tasks for running your job by setting the argument '**-numReduceTasks**' to a number of your choice. By default, it is set to 1; this should be set to a low number, as a larger number will increase the number of partitions created in the HDFS.

Step 4: Now you can check the output of the MapReduce job. Run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop fs -ls
/user/ec2-user/output_1/
```

Step 5: To check the actual output, type the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop fs -cat
/user/ec2-user/output_1/part*
```

Segment 5: The Combiner

In this segment, you will be calculating the average of the ages of the customers corresponding to their gender with the help of a Combiner.

Step 1: Let's first create the Mapper script for your program. Follow the steps mentioned in the previous segment and create a script, say '**mapperage_avg.py**', and input the following code into the file:

```
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    if len(line) >=2:
        gender = line[1]
        age = line[2]

        print ('%s\t1\t%s' % (gender, age))
```

Step 2: Now you need to write the Combiner script. Create a file named '**combiner_avg.py**' and input the following code into the file:

```
import sys
gender_dict = {}
gender_count, gender_sum = 0, 0

for line in sys.stdin:
      line = line.strip()
      gender, count, age = line.split( '\t' )
      count, age = int(count), int(age)

      if gender in gender_dict:
            gender_dict[gender][1]+=age
            gender_dict[gender][0]+=count
      else :
            gender_dict[gender] = []
            gender_dict[gender].append(count)
            gender_dict[gender].append(age)

#Reducer
for gender in gender_dict.keys():

print(gender+"\t"+str(gender_dict[gender][0])+"\t"+str(gender_dict[gender][
1]))
```

Note: Please take care of proper indentation while typing the python code.

Step 3: Now you need to write the Reducer for this program. Create a new file, say '**reduced_avg.py'**, and input the following code into the file:

```
import sys
```

```
gender_dict = {}
gender_count, gender_sum = 0, 0

for line in sys.stdin:
        line = line.strip()
        gender, count, age = line.split('\t')
        count, age = int(count), int(age)

        if gender in gender_dict:
                gender_dict[gender][1]+=age
                gender_dict[gender][0]+=count
        else:
                gender_dict[gender] = []
                gender_dict[gender].append(count)
                gender_dict[gender].append(age)

#Reducer
for gender in gender_dict.keys():
        flag = float(gender_dict[gender][1])/float(gender_dict[gender][0])
        print(gender+"\t"+str(flag))
```

Note: Make sure that you give execute permissions to the code files created before running the MapReduce job using the 'chmod 700' command.

Note: Here you can try to run these files locally to check if the code is working properly before running it on Hadoop. For this you can use the command - 'cat CustomerAge.csv | python mapperage_avg.py | python combiner_avg.py | python reduced_avg.py'

Note: In Hadoop, even if a program fails, the output directory may be created and so always remember to check that the output directory has not already been created before running a MapReduce job.

Step 4: Now you can run the MapReduce job to find the average in question. Run the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop jar
/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.
15.1.jar \
-file mapperage_avg.py -mapper 'python mapperage_avg.py' \
-file reduced_avg.py -reducer 'python reduced_avg.py' \
-input /user/ec2-user/CustomerAge.txt \
-output /user/ec2-user/output_2 \
-file combiner_avg.py -combiner 'python combiner_avg.py'
```

Step 5: You can check the output of the program again by typing the following command:

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ hadoop fs -cat
/user/ec2-user/output_2/part*
```

Segment 6: The Partitioner

In this segment, you will use the airline data set and calculate the average flight delay corresponding to each year in the data set. Since most queries will be based on the year, you need to use the Partitioner so that all key-value pairs belonging to one year are processed by one Reduce task.

Step 1: You need to make another directory in your test directory to avoid any confusion between the previous MapReduce scripts and the ones that you will be writing now.

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ mkdir AirlineAverage/
```

```
[ec2-user@ip-10-0-0-113 MapReduce_test]$ cd AirlineAverage/
```

Step 2: You need to push the airline data set into the HDFS cluster after importing it into the EC2 instance by running the '**put**' command:

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -put AirLineData.csv
/user/ec2-user/demoAirLine/input/
```

Note: Here note that you need to first make a demoAirLine directory and its subdirectories before transferring the AirLineData.csv file to this directory in HDFS. You can do this by using the 'hadoop fs -mkdir /user/ec2-user/demoAirLine' and so on for the other subdirectories.

Step 3: Now you need to write the Mapper script. Create a file named '**mapper_airlineAvg.py**' and input the following code into the file:

```python
import sys
for line in sys.stdin:
    line = line.strip()
    line = line.split(",")
    if line[0] == "ID":
        continue
    year = line[1]
    arrdel = line[15]
    depdel = line[16]
    if arrdel == "NA" or depdel == "NA":
        continue
    arrdel = int(arrdel)
    depdel = int(depdel)
    if arrdel+depdel>0:
        total_del = arrdel+depdel
        print("{}\t1\t{}".format(year, total_del))
```

Step 4: Now, to write the Combiner script, create a file named ‘**combiner_airlineAvg.py**’ and input the following code into the file:

```python
import sys

flight = {}
flight_count, del_sum = 0, 0

for line in sys.stdin:
    line = line.strip()
    year, count, delay = line.split("\t")
    count, delay = int(count), int(delay)

    if year in flight:
        flight[year][1] += delay
        flight[year][0] += count
    else:
        flight[year] = []
        flight[year].append(count)
        flight[year].append(delay)

for year in flight.keys():
    print("{}\t{}\t{}".format(year, flight[year][0], flight[year][1]))
```

Step 4: After this, you need to write the Reducer script. For this, create a file named '**reducer_airlineAvg.py**' and input the following code into the file:

```
import sys

flight = {}
flight_count, del_sum = 0, 0

for line in sys.stdin:
    line = line.strip()
    year, count, delay = line.split("\t")
    count, delay = int(count), int(delay)

    if year in flight:
        flight[year][1] += delay
        flight[year][0] += count
    else:
        flight[year] = []
        flight[year].append(count)
        flight[year].append(delay)

for year in flight.keys():
    average = float(flight[year][1])/float(flight[year][0])
    print("{}\t{}".format(year, average))
```

Note: Make sure that you give execute permissions to the code files created before running the MapReduce job using the 'chmod 700' command.

Step 4: You can now run the MapReduce job to find the average in question. Run the following command:

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop jar
/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.
15.1.jar \
-files mapper_airlineAvg.py,reducer_airlineAvg.py,combiner_airlineAvg.py \
-mapper "python mapper_airlineAvg.py" \
-combiner "python combiner_airlineAvg.py" \
-reducer "python reducer_airlineAvg.py" \
-input /user/ec2-user/demoAirLine/input/AirLineData.csv \
-output /user/ec2-user/demoAirLine/output_1 \
-partitioner org.apache.hadoop.mapred.lib.HashPartitioner -numReduceTasks 5
```

Step 5: Now, let’s look at the output of one of the part files made after running the MapReduce job:

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -cat
/user/ec2-user/demoAirLine/output_1/part-00000
```

As you can see, this part file has the result only for one of the years..
Now, if you check the output part files individually for all the other files, you will see that each of them has the output of a single year, which means that each year had a different partition and, therefore, was assigned different Reduce tasks by the Partitioner.

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -cat
/user/ec2-user/demoAirLine/output_1/part-00001
```

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -cat
/user/ec2-user/demoAirLine/output_1/part-00002
```

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -cat
/user/ec2-user/demoAirLine/output_1/part-00003
```

```
[ec2-user@ip-10-0-0-113 AirlineAverage]$ hadoop fs -cat
/user/ec2-user/demoAirLine/output_1/part-00004
```