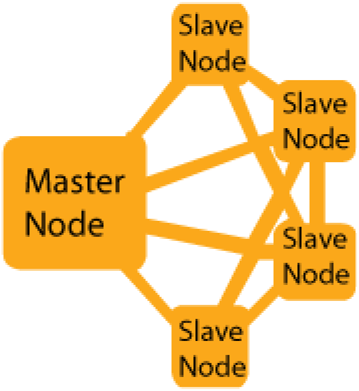
Tutorial 5: Hadoop MapReduce

1- What is Hadoop?

Hadoop uses a distributed processing architecture called MapReduce in which a task is mapped to a set of servers for processing. The results of the computation performed by those servers are then reduced to a single output set. One node, designated as the master node, controls the distribution of tasks. The following diagram shows a Hadoop cluster with the master node directing a group of slave nodes which process the data.



2-What is Amazon Elastic MapReduce?

With Amazon Elastic MapReduce (Amazon EMR) you can analyze and process vast amounts of data. It does this by distributing the computational work across a cluster of virtual servers running in the Amazon cloud. The cluster is managed using an open-source framework called Hadoop.

Objectives

- Learn how to write MapReduce code using java.
- Learn how to set up a single-note Hadoop deployment and test your MapReduce code.
- Learn how to run MapReduce code in cloud using AWS EMR.

1 Setup a single-node Hadoop Environment:

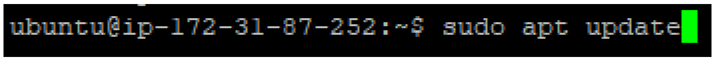
In this section, we will go over the steps to deploy a single-node Hadoop environment, and run MapReduce applications written in Java using Eclipse.

1. Go to AWS Management Console via AWS Academy portal and create an EC2 instance running Ubuntu Server 18 using the knowledge gathered during **Tutorial 03**.

   We will be using this EC2 instance to deploy Hadoop and run MapReduce applications in the subsequent steps.

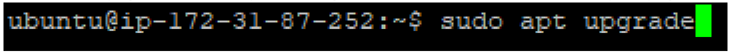
2. Log in to your EC2 instance via SSH either using the ssh command line utility on MacOS/Linux or Putty on Windows. We learnt how to do this in **Tutorial 03**.

3. In **Tutorial 03**, we installed Java 17 into the EC2 instances that were created. However, this time, we will install Java 1.8 as per the steps provided below. This is because, Hadoop still only supports Java 1.8 fully.
   a. First, check for updates on any package already installed via the following command.



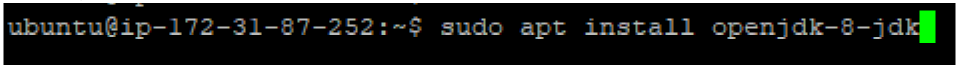
```
ubuntu@ip-172-31-87-252:~$ sudo apt update
```

   b. Then, upgrade all packages that are currently upgradable, as below.



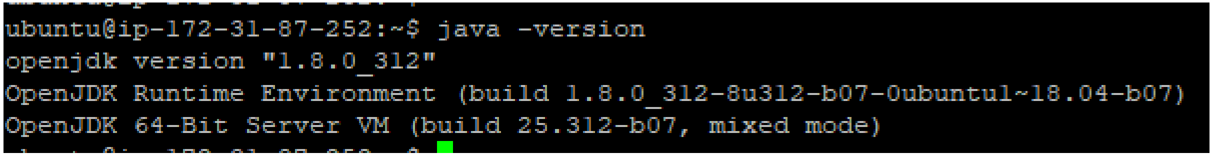
```
ubuntu@ip-172-31-87-252:~$ sudo apt upgrade
```

   c. Finally, we will install Java 1.8, as below.



```
ubuntu@ip-172-31-87-252:~$ sudo apt install openjdk-8-jdk
```

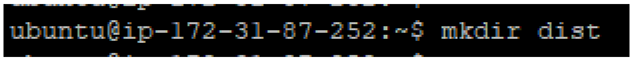
   d. Once done, verify your Java installation as below. You should see the message printed on the command line pointing to Java 1.8.



```
ubuntu@ip-172-31-87-252:~$ java -version
openjdk version "1.8.0_312"
OpenJDK Runtime Environment (build 1.8.0_312-8u312-b07-0ubuntu1~18.04-b07)
OpenJDK 64-Bit Server VM (build 25.312-b07, mixed mode)
```

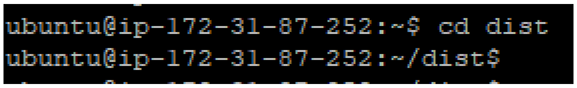
4. Once Java has been properly installed, download the latest released version of Hadoop version (e.g. 3.3.2) from https://archive.apache.org/dist/hadoop/core/current/ into your EC2 instance using the steps below.

   a. Create a directory called **dist** within the home directory (e.g. /home/ubuntu) of the currently logged in user (e.g. ubuntu)



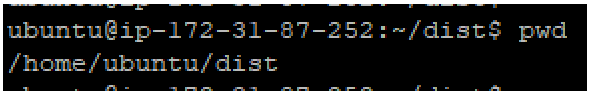
```
ubuntu@ip-172-31-87-252:~$ mkdir dist
```

   b. Change the current working directory to the one created above.



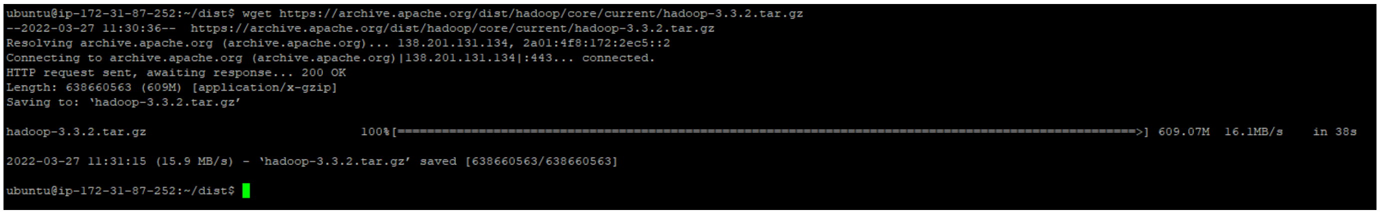
```
ubuntu@ip-172-31-87-252:~$ cd dist
ubuntu@ip-172-31-87-252:~/dist$
```

c. Verify that you are now in the **dist** directory by running the following command.



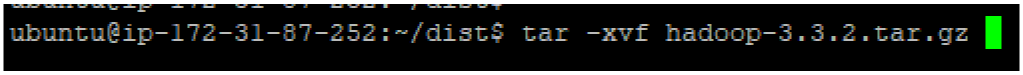
```
ubuntu@ip-172-31-87-252:~/dist$ pwd
/home/ubuntu/dist
```

d. Download the latest released version of Hadoop into your **dist** directory from https://archive.apache.org/dist/hadoop/core/current/, as below.



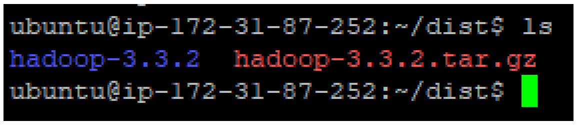
```
ubuntu@ip-172-31-87-252:~/dist$ wget https://archive.apache.org/dist/hadoop/core/current/hadoop-3.3.2.tar.gz
--2022-03-27 11:30:36--  https://archive.apache.org/dist/hadoop/core/current/hadoop-3.3.2.tar.gz
Resolving archive.apache.org (archive.apache.org)... 138.201.131.134, 2a01:4f8:172:2ec5::2
Connecting to archive.apache.org (archive.apache.org)|138.201.131.134|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 638660563 (609M) [application/x-gzip]
Saving to: ‘hadoop-3.3.2.tar.gz’

hadoop-3.3.2.tar.gz                   100%[=====================================================================================================>] 609.07M  16.1MB/s    in 38s

2022-03-27 11:31:15 (15.9 MB/s) - ‘hadoop-3.3.2.tar.gz’ saved [638660563/638660563]

ubuntu@ip-172-31-87-252:~/dist$
```

5. Extract the content of the .tar.gz file to a location of your choice by running the following command. In this example, we are extracting the Hadoop framework into the dist directory created above.



```
ubuntu@ip-172-31-87-252:~/dist$ tar -xvf hadoop-3.3.2.tar.gz
```

Once properly extracted, you will see the hadoop-3.3.2 directory in the file system, as shown below.

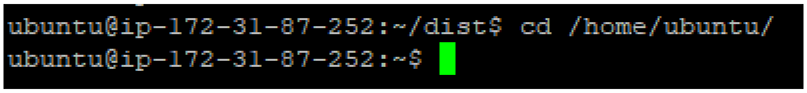


```
ubuntu@ip-172-31-87-252:~/dist$ ls
hadoop-3.3.2  hadoop-3.3.2.tar.gz
ubuntu@ip-172-31-87-252:~/dist$
```

6. In the next step, we will set some environment variables so that we can invoke the hadoop provided tools and scripts on the command line.

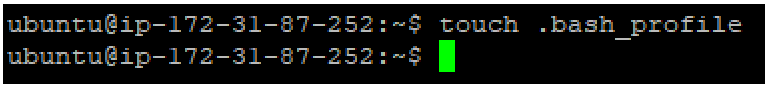
   For this we will be modifying a hidden file in your home folder of the currently logged in user (e.g. /home/ubuntu/) called .bash_profile, which allows you to set permanent environment variables.

a. Go to the home folder (e.g. /home/ubuntu) of the currently logged in user (e.g. ubuntu).



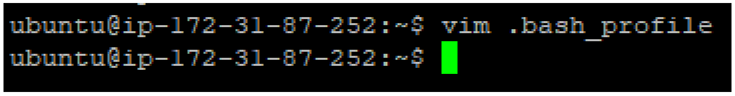
```
ubuntu@ip-172-31-87-252:~/dist$ cd /home/ubuntu/
ubuntu@ip-172-31-87-252:~$
```

b. If the .bash_profile file does not exist in the home folder, just create that file in your home folder using the touch command:



```
ubuntu@ip-172-31-87-252:~$ touch .bash_profile
ubuntu@ip-172-31-87-252:~$
```

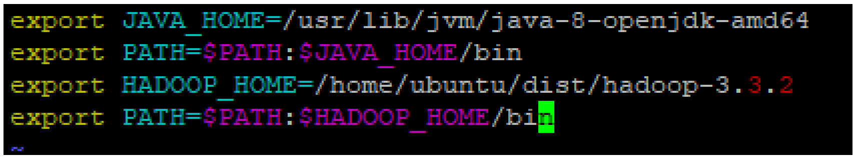
c. You can then open .bash-profile in your home folder using a text editor, for example



```
ubuntu@ip-172-31-87-252:~$ vim .bash_profile
ubuntu@ip-172-31-87-252:~$
```

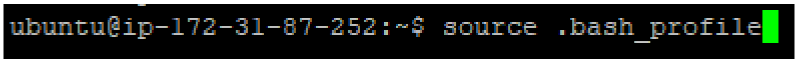
d. We will do the following tasks:

- Set the JAVA_HOME variable to the place where you installed the Java SDK, as well as configuring the PATH variable to point to its bin directory.
- Set the HADOOP_HOME variable to the place where you extract the Hadoop distribution, as well as configuring the PATH variable to point to its bin directory.



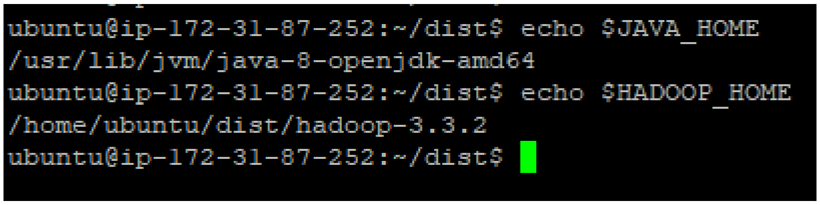
```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/home/ubuntu/dist/hadoop-3.3.2
export PATH=$PATH:$HADOOP_HOME/bin
~
```

Save the .bash_profile file, run the following command to reload your command line to be able to take those variables to take effect.



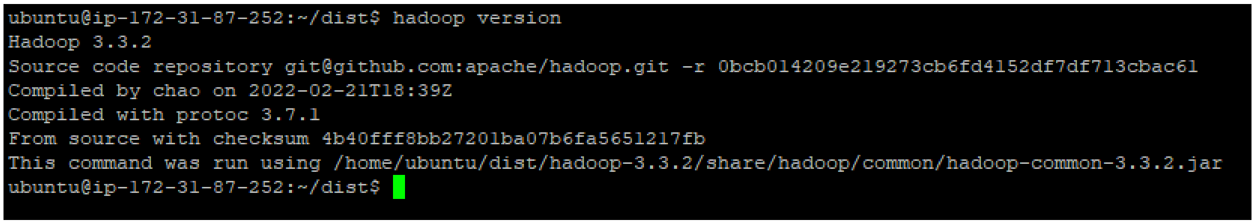
```
ubuntu@ip-172-31-87-252:~$ source .bash_profile
```

Now you can check and inspect all your environment variables via the command line, as below.



```
ubuntu@ip-172-31-87-252:~/dist$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64
ubuntu@ip-172-31-87-252:~/dist$ echo $HADOOP_HOME
/home/ubuntu/dist/hadoop-3.3.2
ubuntu@ip-172-31-87-252:~/dist$
```

7. Test your *hadoop version* command; you should be able to see something like:



```
ubuntu@ip-172-31-87-252:~/dist$ hadoop version
Hadoop 3.3.2
Source code repository git@github.com:apache/hadoop.git -r 0bcb014209e219273cb6fd4152df7df713cbac61
Compiled by chao on 2022-02-21T18:39Z
Compiled with protoc 3.7.1
From source with checksum 4b40fff8bb27201ba07b6fa5651217fb
This command was run using /home/ubuntu/dist/hadoop-3.3.2/share/hadoop/common/hadoop-common-3.3.2.jar
ubuntu@ip-172-31-87-252:~/dist$
```

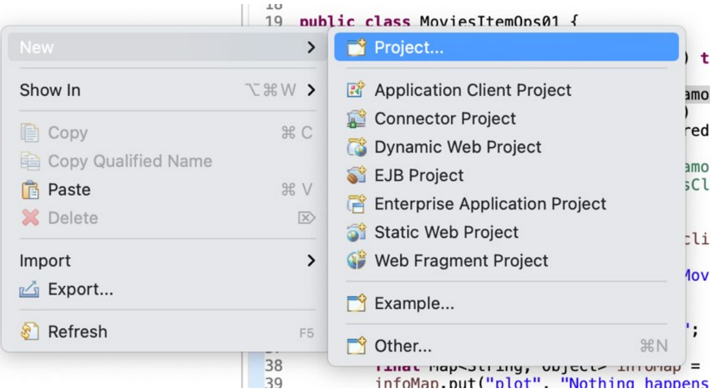
Now we are ready to run our MapReduce applications. In summary, we will write code and compile our MapReduce code on Eclipse. Then, we will export our MapReduce applications as a jar file and run it on top of Hadoop using the command line.

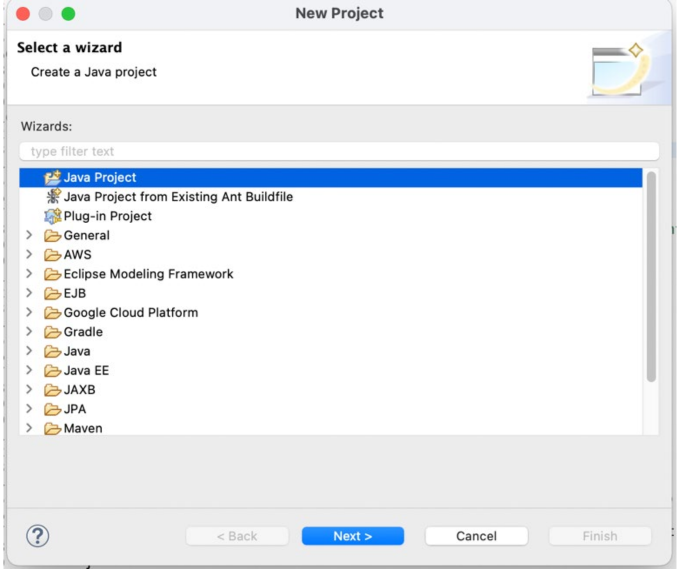
1.1 Set-up Eclipse to write MapReduce applications

1. Download the same Hadoop distribution (i.e. hadoop-3.3.2.tar.gz) from https://archive.apache.org/dist/hadoop/core/current/, now into **your local development environment**. Extract it to your preferred location in the file system.

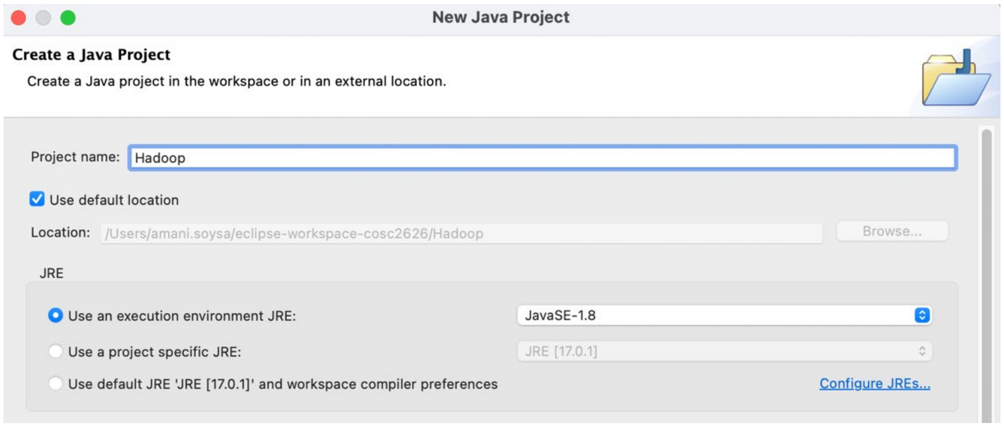
   This Hadoop distribution not only bundles the Hadoop distribution that runs MapReduce applications, but also the client libraries required to write the aforementioned applications. These client libraries could be located in hadoop-<version>/lib directory.

2. Create a Java project on Eclipse called **Hadoop**. To do this, either right click on the **Project Explorer** or select **File -> New -> Java Project**.





While creating your Java project, make sure you select **Java 1.8** as the JRE associated with the project, as shown below.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Hadoop

☑ Use default location

Location: /Users/amani.soysa/eclipse-workspace-cosc2626/Hadoop Browse...

JRE

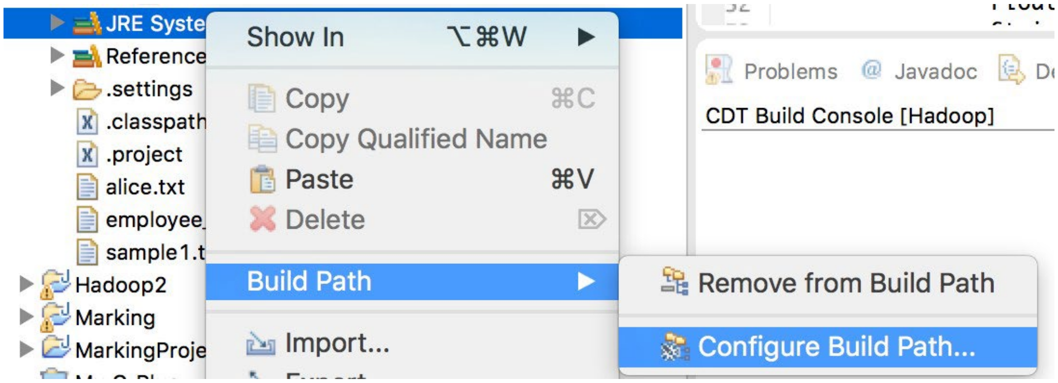
● Use an execution environment JRE: JavaSE-1.8

○ Use a project specific JRE: JRE [17.0.1]

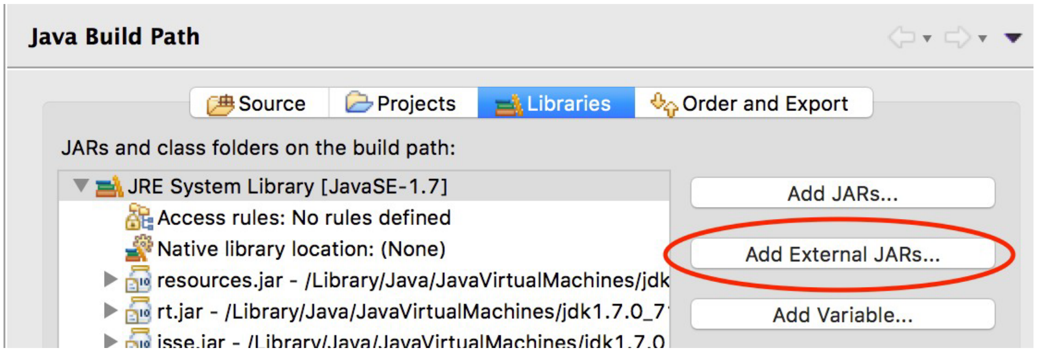
○ Use default JRE 'JRE [17.0.1]' and workspace compiler preferences Configure JREs...

3. Now, we need to add 2 more external jar files to the classpath of our newly created **Hadoop** project. These jar files correspond to the client libraries required to write MapReduce applications, which we previously downloaded as part of the Hadoop distribution.

   a. Right click on the JRE system library of your project, select Build Path, **Configure Build Path**, as below.



   b. Click on **Add External JARs**.



JARs and class folders on the build path:

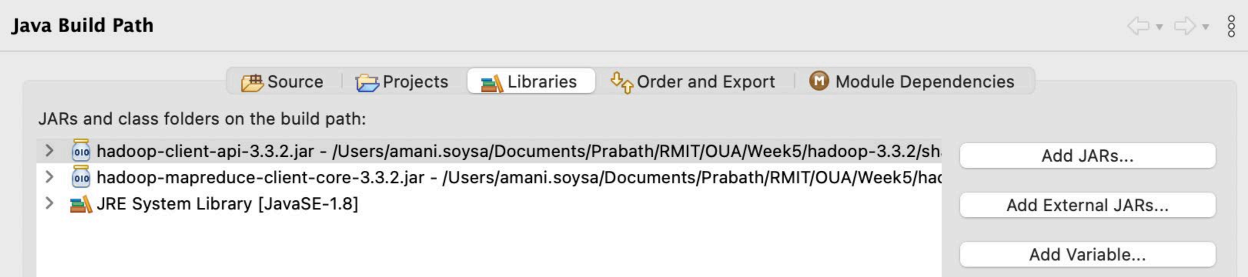
- JRE System Library [JavaSE-1.7]
  - Access rules: No rules defined
  - Native library location: (None)
  - resources.jar - /Library/Java/JavaVirtualMachines/jdk
  - rt.jar - /Library/Java/JavaVirtualMachines/jdk1.7.0_7
  - jsse.jar - /Library/Java/JavaVirtualMachines/jdk1.7.0

c. Search in the folder where you have extracted Hadoop and add the following 2 jar files (Shown below is the full path to those jar files).

Note: Here, **HADOOP_HOME** refers to the folder to which you have downloaded the **hadoop-<version>.tar.gz** package.

```
$HADOOP_HOME/share/hadoop-3.3.2/client/hadoop-client-api-3.3.2.jar
$HADOOP_HOME/share/hadoop-3.3.2/mapreduce/hadoop-mapreduce-client-core-3.3.2.jar
```

d. After you have added those 2 jars file, you should see something like following.

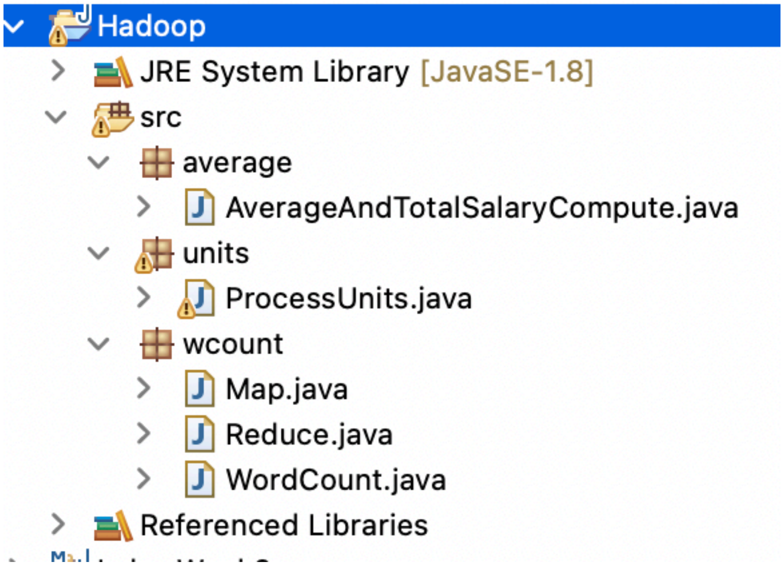


JARs and class folders on the build path:

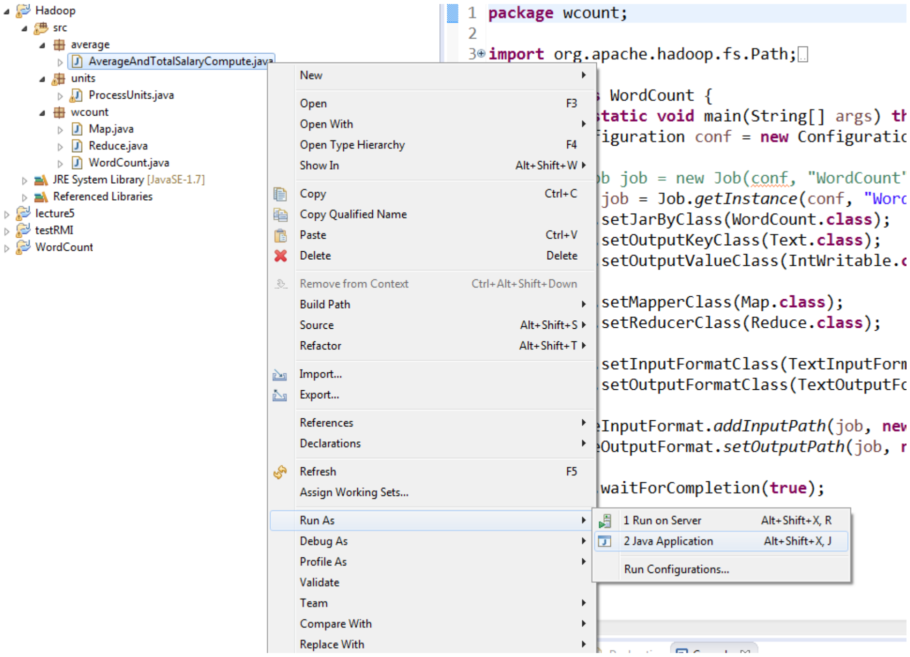
- hadoop-client-api-3.3.2.jar - /Users/amani.soysa/Documents/Prabath/RMIT/OUA/Week5/hadoop-3.3.2/sh
- hadoop-mapreduce-client-core-3.3.2.jar - /Users/amani.soysa/Documents/Prabath/RMIT/OUA/Week5/had
- JRE System Library [JavaSE-1.8]

4. In the src folder of the Java project you created (e.g. Hadoop), add 3 new packages named
   1) average
   2) units
   3) wcount

5. Now download the **mrcode.zip** package from the Canvas via *Modules -> Week 5: additional learning materials* section. Extract this package and copy the **Java class files** inside that package into your **Hadoop** project according to the following structure.

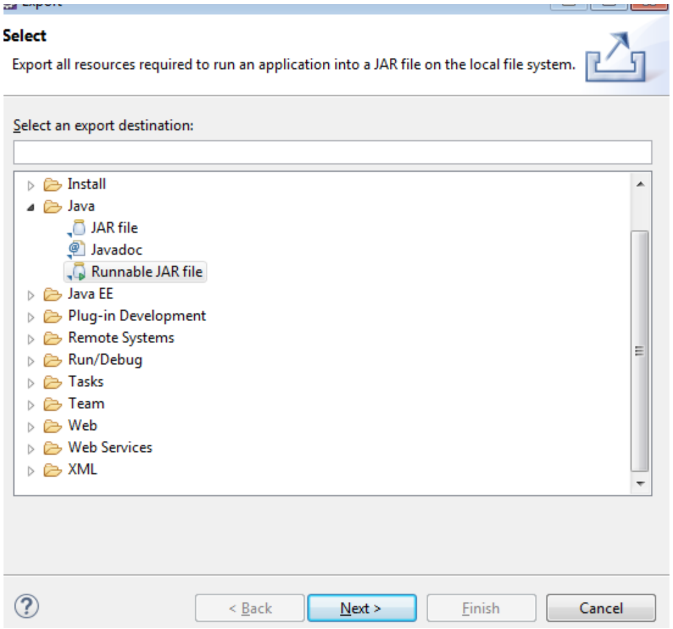


6. Closely examine the implementation of the sample code you copied into your project. Try to understand how the MapReduce paradigm is implemented in these sample applications. Now run each of 3 packages as a **Java application**, as below. Don't worry if you get exceptions.



```
package wcount;

import org.apache.hadoop.fs.Path;

 WordCount {
tatic void main(String[] args) th
iguration conf = new Configuratic

b job = new Job(conf, "WordCount'
job = Job.getInstance(conf, "Worc
setJarByClass(WordCount.class);
setOutputKeyClass(Text.class);
setOutputValueClass(IntWritable.c

setMapperClass(Map.class);
setReducerClass(Reduce.class);

setInputFormatClass(TextInputForm
setOutputFormatClass(TextOutputFc

InputFormat.addInputPath(job, new
OutputFormat.setOutputPath(job, r

waitForCompletion(true);
```

7. Now, export each of the aforementioned 3 packages as runnable jars. For example, to export wcount package as a jar, right click on the package name and select export. In the Export window, select **Java > Runnable jar file**.

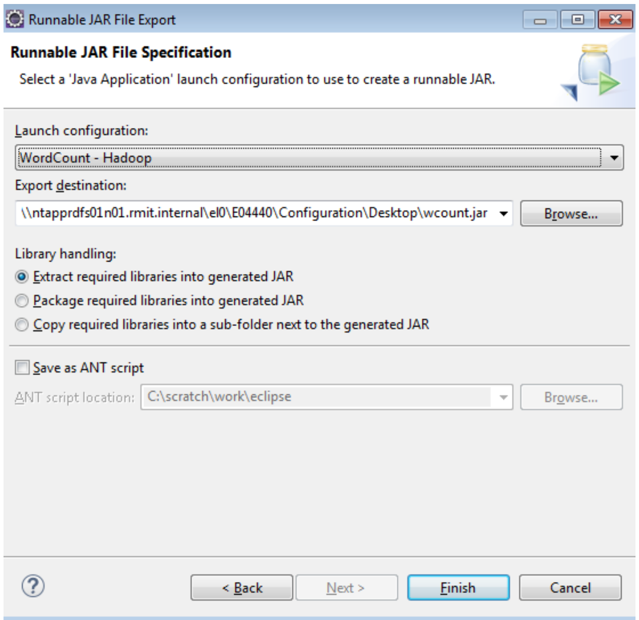


Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export destination:

8. In next window, Select the main file as Launch configuration. For example, for wcount package the main function is in WordCount.java file. Also give the jar file name same as package name (e.g. wcount.jar) as in the following screenshot.



Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: WordCount - Hadoop

Export destination: \\ntapprdfs01n01.rmit.internal\el0\E04440\Configuration\Desktop\wcount.jar

Library handling:

● Extract required libraries into generated JAR

○ Package required libraries into generated JAR

○ Copy required libraries into a sub-folder next to the generated JAR

□ Save as ANT script

ANT script location: C:\scratch\work\eclipse

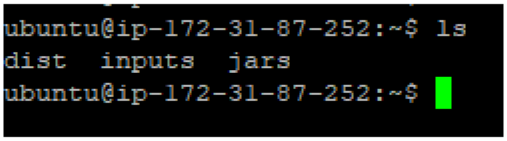
9. Click **Finish**. Click Ok if you see any warning. Now your jar files are ready to be run on Hadoop.

10. Now, create a new directory called **jars** inside the home directory (e.g. /home/ubuntu/) of the logged in user (e.g. ubuntu) within the EC2 instance we

created in a previous step. Copy all generated runnable jar files you have created earlier into this **jars** directory.

Also create another folder called **inputs** inside home directory of the logged in user within your EC2 instance. Now, copy all the text files in **mrcode/inputs** directory you downloaded from Canvas into this directory.

The folder hierarchy within the home directory (e.g. /home/ubuntu/) of the logged in user (e.g. ubuntu) should now look like the following.



```
ubuntu@ip-172-31-87-252:~$ ls
dist  inputs  jars
ubuntu@ip-172-31-87-252:~$
```

Run the first project: Electrical Consumption

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

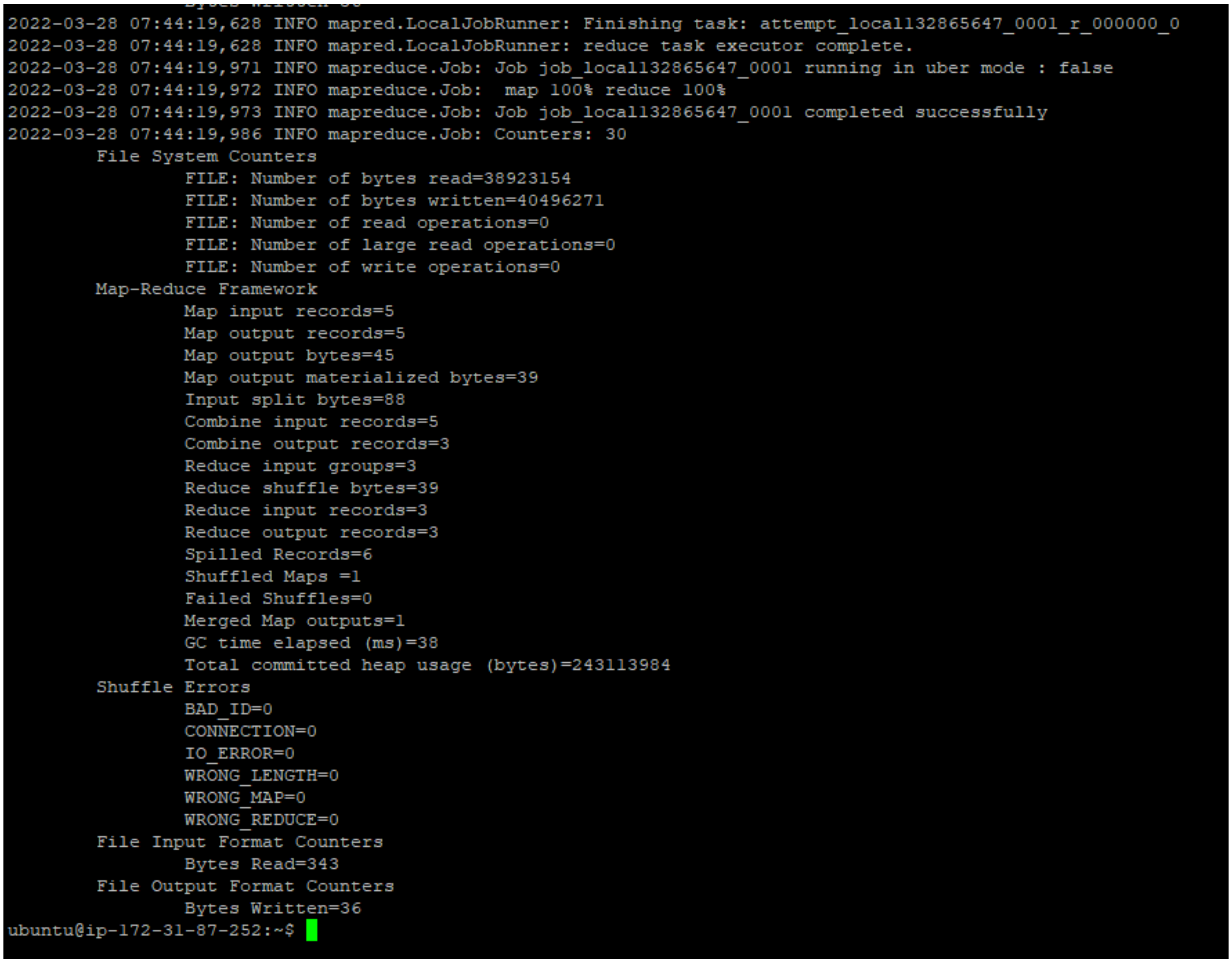
If the above data is given as input, we have to write applications to process it and produce results such as finding the year of average usage that is greater than 30.

Connect to your EC2 instance via SSH. Then run the following command from the home directory (e.g. /home/ubuntu) of the logged in user (e.g. ubuntu).

```
hadoop jar jars/units.jar inputs/sample1.txt output-units
```

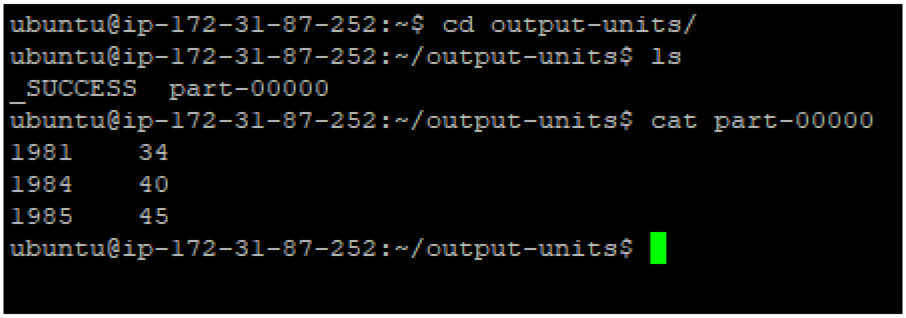
Here, we used the hadoop command line utility (CLI) to run the MapReduce application embedded in **units.jar** file. Hadoop then processes the data inside the **input/sample1.txt** file and generate the output inside **output-units** folder. Note that here you don't need to create an output folder. Hadoop will automatically create it and place the result in it.

After Hadoop finish running you will see output like the following



```
2022-03-28 07:44:19,628 INFO mapred.LocalJobRunner: Finishing task: attempt_local132865647_0001_r_000000_0
2022-03-28 07:44:19,628 INFO mapred.LocalJobRunner: reduce task executor complete.
2022-03-28 07:44:19,971 INFO mapreduce.Job: Job job_local132865647_0001 running in uber mode : false
2022-03-28 07:44:19,972 INFO mapreduce.Job:  map 100% reduce 100%
2022-03-28 07:44:19,973 INFO mapreduce.Job: Job job_local132865647_0001 completed successfully
2022-03-28 07:44:19,986 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=38923154
                FILE: Number of bytes written=40496271
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=5
                Map output records=5
                Map output bytes=45
                Map output materialized bytes=39
                Input split bytes=88
                Combine input records=5
                Combine output records=3
                Reduce input groups=3
                Reduce shuffle bytes=39
                Reduce input records=3
                Reduce output records=3
                Spilled Records=6
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=38
                Total committed heap usage (bytes)=243113984
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=343
        File Output Format Counters
                Bytes Written=36
ubuntu@ip-172-31-87-252:~$
```

Let's now look at the output.



```
ubuntu@ip-172-31-87-252:~$ cd output-units/
ubuntu@ip-172-31-87-252:~/output-units$ ls
_SUCCESS  part-00000
ubuntu@ip-172-31-87-252:~/output-units$ cat part-00000
1981    34
1984    40
1985    45
ubuntu@ip-172-31-87-252:~/output-units$
```

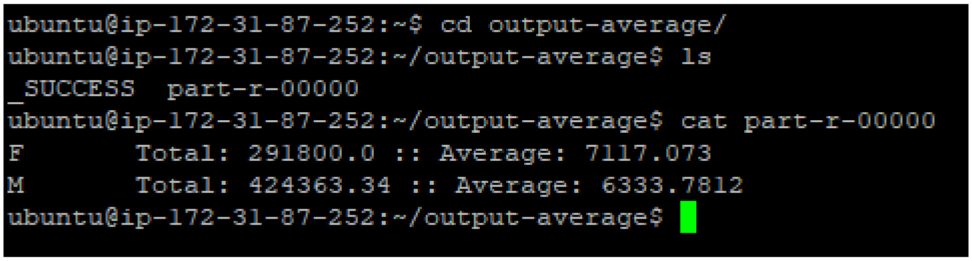
When Hadoop has finished running the MapReduce application successfully, the content of the output folder looks like what is shown above. To view the content of the output file, you use the **cat** command.

Run the second project: Average Salary by Genders

Let's now run another application, which outputs the average salary of employees grouped by their geneder. To do this, run the second runnable jar file named **average.jar** using the following command.

```
hadoop jar jars/average.jar inputs/employee_records.txt
output-average
```

Upon successful execution of the aforementioned, command your output will look like what is shown below.



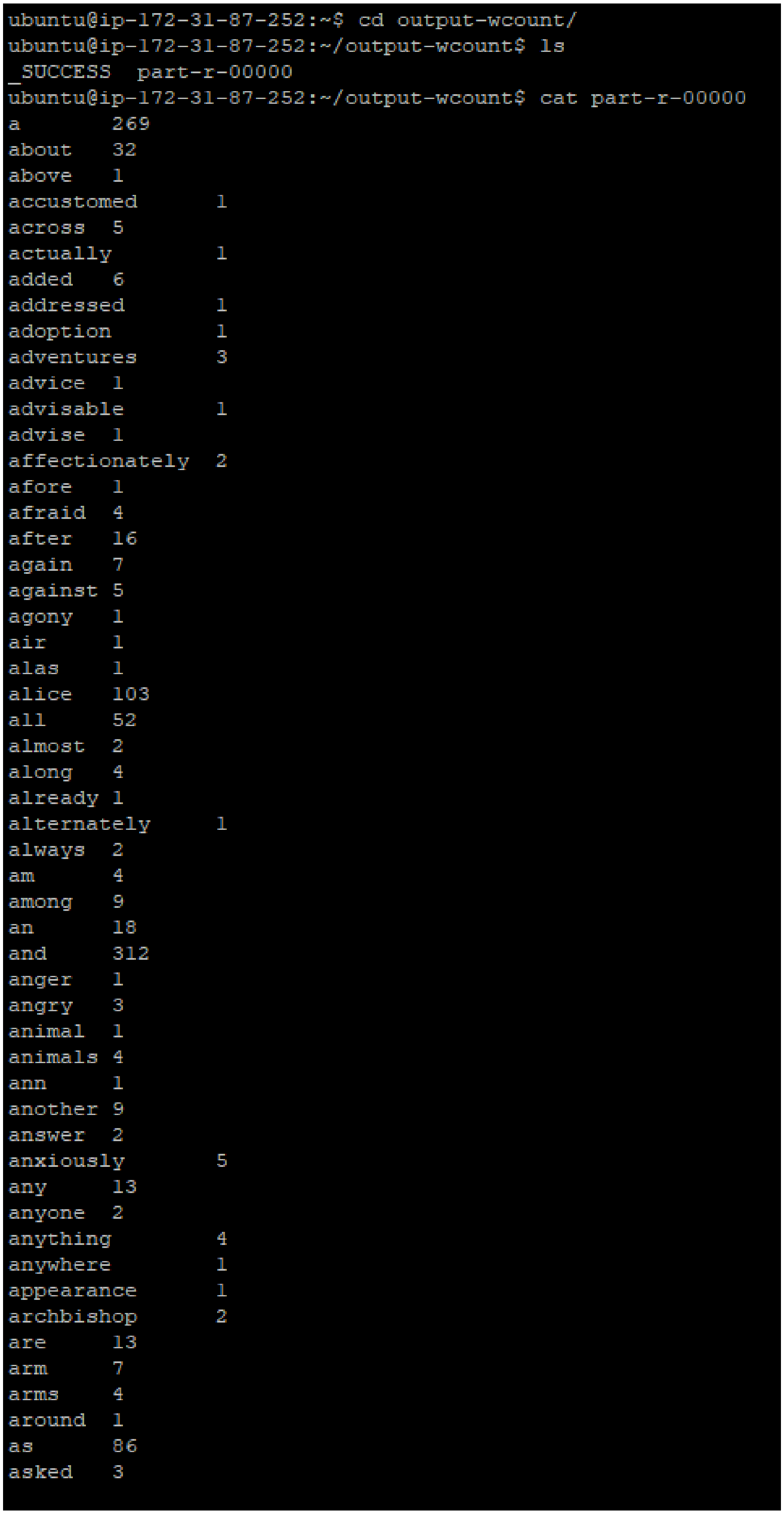
```
ubuntu@ip-172-31-87-252:~$ cd output-average/
ubuntu@ip-172-31-87-252:~/output-average$ ls
_SUCCESS  part-r-00000
ubuntu@ip-172-31-87-252:~/output-average$ cat part-r-00000
F       Total: 291800.0 :: Average: 7117.073
M       Total: 424363.34 :: Average: 6333.7812
ubuntu@ip-172-31-87-252:~/output-average$
```

Run the third project: Word Count

Once again remove the employee_records.txt from input folder and remove the output folder. Then copy alice.txt in input folder. Now test the third package **wcount** by using the command.

```
hadoop jar jars/wcount.jar inputs/alice.txt output-wcount
```

Upon successful execution of the aforementioned, command your output will look like what is shown below.



```
ubuntu@ip-172-31-87-252:~$ cd output-wcount/
ubuntu@ip-172-31-87-252:~/output-wcount$ ls
_SUCCESS  part-r-00000
ubuntu@ip-172-31-87-252:~/output-wcount$ cat part-r-00000
a       269
about   32
above   1
accustomed      1
across  5
actually        1
added   6
addressed       1
adoption        1
adventures      3
advice  1
advisable       1
advise  1
affectionately  2
afore   1
afraid  4
after   16
again   7
against 5
agony   1
air     1
alas    1
alice   103
all     52
almost  2
along   4
already 1
alternately     1
always  2
am      4
among   9
an      18
and     312
anger   1
angry   3
animal  1
animals 4
ann     1
another 9
answer  2
anxiously       5
any     13
anyone  2
anything        4
anywhere        1
appearance      1
archbishop      2
are     13
arm     7
arms    4
around  1
as      86
asked   3
```

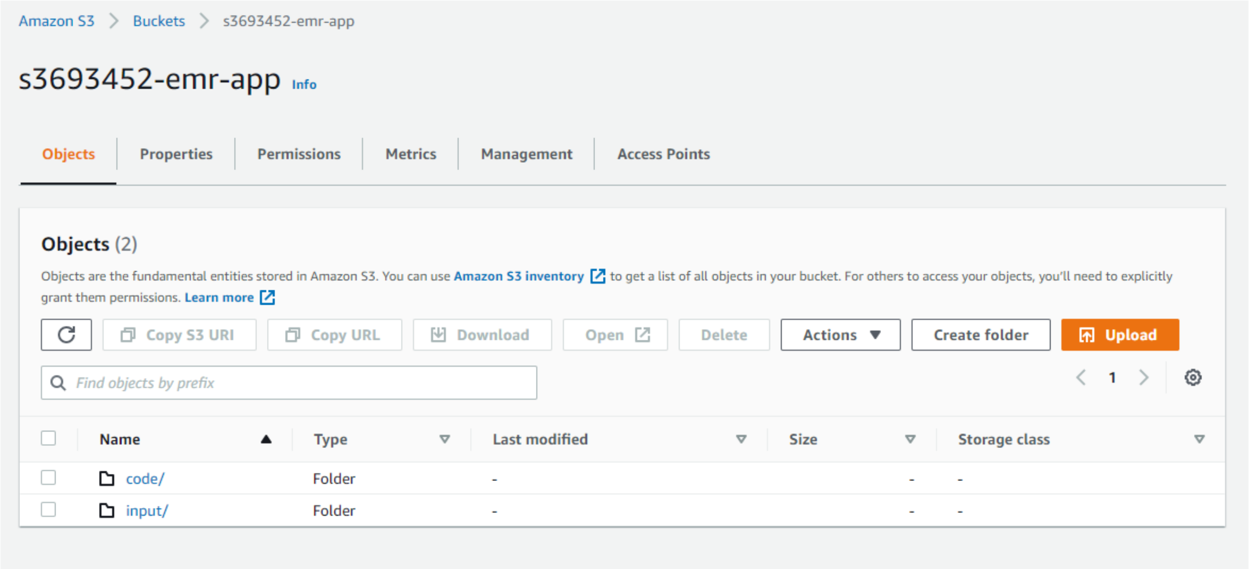
Here, Hadoop is generating only one output file because we are using a single node cluster. That means, there is only one processing node at work. This process is helpful when your dataset is small and when you need to test your developed MapReduce code locally. But if your dataset is large a single node cluster will not be enough, and you need to take the advantage of cloud by using a Multi-Node cluster. AWS has such support in EMR. Now we will use our developed MapReduce code in EMR.

2 Solve word count problem using EMR (Amazon Elastic MapReduce)

2.1 Prepare your scripts and input data

1- Log in to your AWS management console via AWS Academy portal. Select AWS S3 and create a bucket named **sxxxxxx-emr-app** (sxxxxxx refers to RMIT your student id).

2- Create two folders inside your bucket and name them **code** and **input.**



Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name ▲	Type	Last modified	Size	Storage class
□	code/	Folder	-	-	-
□	input/	Folder	-	-	-

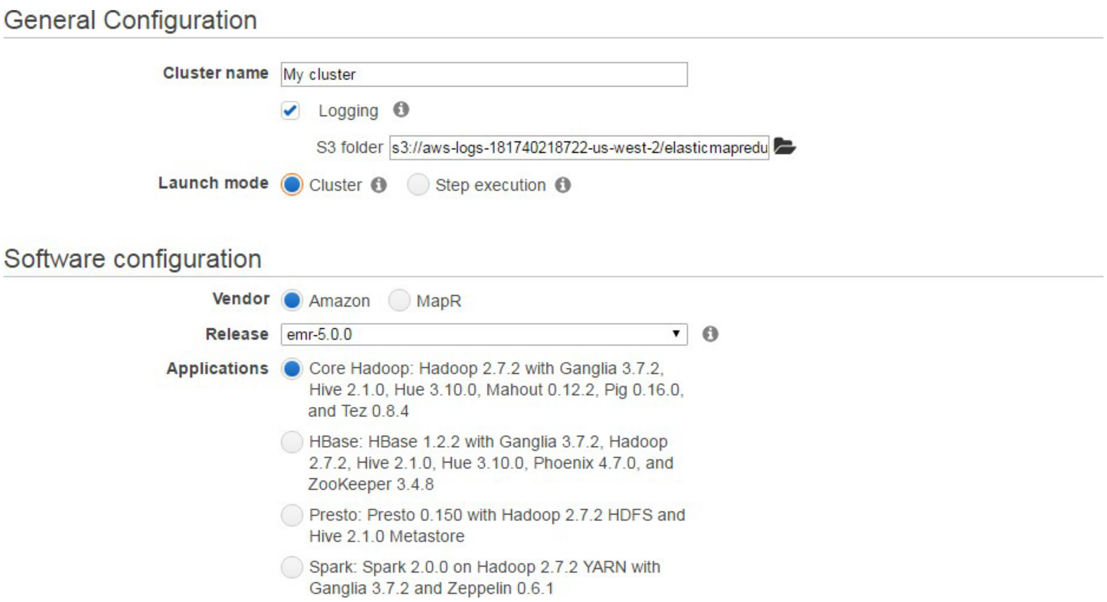
3- Inside **code** folder, upload the **wcount.jar** file you created before.

4- Download the text file from here: http://www.umich.edu/~umfandsf/other/ebooks/alice30.txt and upload alice30.txt into the **input** folder of your Bucket.

2.2 Create Amazon MapReduce cluster

1- Login to AWS management console and from services select "**EMR**".

2- Click on **Create cluster** and you will see a screen like following.



General Configuration

Cluster name: My cluster

☑ Logging

S3 folder: s3://aws-logs-181740218722-us-west-2/elasticmapredu

Launch mode: ● Cluster ○ Step execution

Software configuration

Vendor: ● Amazon ○ MapR

Release: emr-5.0.0

Applications:

● Core Hadoop: Hadoop 2.7.2 with Ganglia 3.7.2, Hive 2.1.0, Hue 3.10.0, Mahout 0.12.2, Pig 0.16.0, and Tez 0.8.4

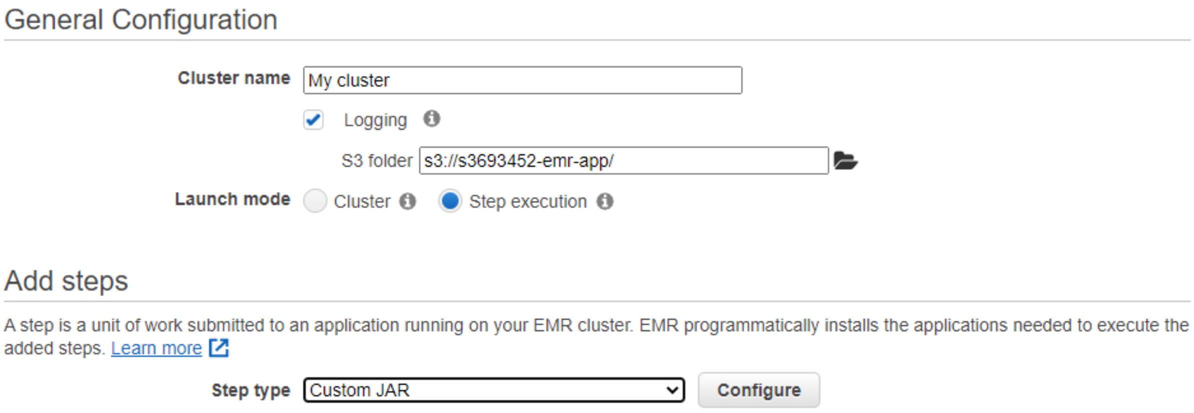
○ HBase: HBase 1.2.2 with Ganglia 3.7.2, Hadoop 2.7.2, Hive 2.1.0, Hue 3.10.0, Phoenix 4.7.0, and ZooKeeper 3.4.8

○ Presto: Presto 0.150 with Hadoop 2.7.2 HDFS and Hive 2.1.0 Metastore

○ Spark: Spark 2.0.0 on Hadoop 2.7.2 YARN with Ganglia 3.7.2 and Zeppelin 0.6.1

3- Give a name to your cluster. In the **Logging S3 folder location** field browse and select s3://sxxxxxxx-emr-app, where sxxxxxxx your RMIT student id.

3- In Launch mode select **Step execution.** Then, select step type as **Custom JAR** as in the screenshot below, and click **Configure**.



General Configuration

Cluster name: My cluster

☑ Logging

S3 folder: s3://s3693452-emr-app/

Launch mode: ○ Cluster ● Step execution

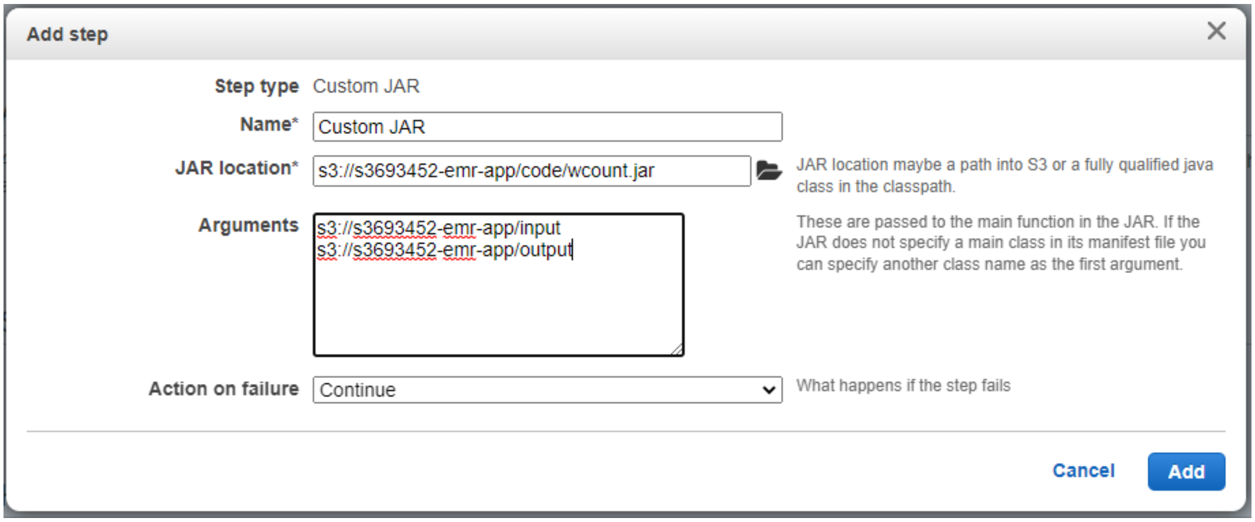
Add steps

A step is a unit of work submitted to an application running on your EMR cluster. EMR programmatically installs the applications needed to execute the added steps. Learn more

Step type: Custom JAR

4- A popup window will open. Fill it as shown in the following figure. In the arguments you have to provide the input location as the first arguments and the output location as the second arguments as shown here.

s3://sxxxxxx-emr-app/input/ s3://sxxxxxx-emr-app/output/



Add step

Step type: Custom JAR

Name*: Custom JAR

JAR location*: s3://s3693452-emr-app/code/wcount.jar

JAR location maybe a path into S3 or a fully qualified java class in the classpath.

Arguments:

```
s3://s3693452-emr-app/input
s3://s3693452-emr-app/output
```

These are passed to the main function in the JAR. If the JAR does not specify a main class in its manifest file you can specify another class name as the first argument.

Action on failure: Continue

What happens if the step fails

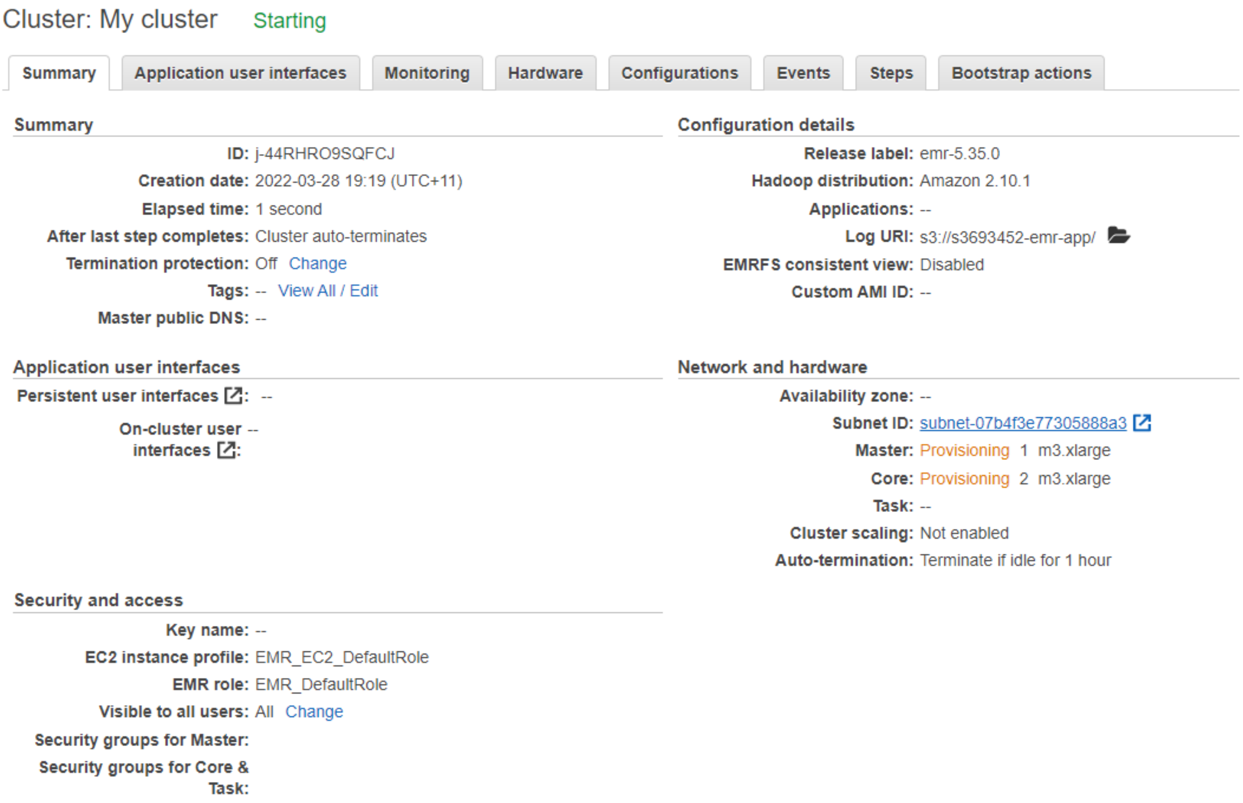
Also select **Terminate cluster** as the **Action on failure**.

Note that, in case of Output **S3 location,** you don't need to create **output** directory explicitly in your bucket. The EMR service will do it for you. However, make sure you type '/output/' after your bucket name in the aforementioned configuration.

Also note that, in this cluster you are using 3 m3.xlarge VM instances. And minimum number of instance also should be 3. But, you can use more powerful instance types and higher number of instances here, which would improve the throughput as well as scalability depending on the requirements of the application. The more resources you use the more you will need to pay. We kept everything at a minimum for this exercise. However, keep in mind that more powerful instances and a large number of instances in your Hadoop cluster will improve the performance of MapReduce applications.

5- Keep **Security and Access** as default.

9-Now click on **Create cluster** and wait until it finishes. It can take 10 minutes or more to complete. It will show "Starting" for some time. It may take more than 5-10 minutes to start. **Starting** means the system is configuring the clusters you have created.



Cluster: My cluster Starting

Summary

ID: j-44RHRO9SQFCJ
Creation date: 2022-03-28 19:19 (UTC+11)
Elapsed time: 1 second
After last step completes: Cluster auto-terminates
Termination protection: Off Change
Tags: -- View All / Edit
Master public DNS: --

Configuration details

Release label: emr-5.35.0
Hadoop distribution: Amazon 2.10.1
Applications: --
Log URI: s3://s3693452-emr-app/
EMRFS consistent view: Disabled
Custom AMI ID: --

Application user interfaces

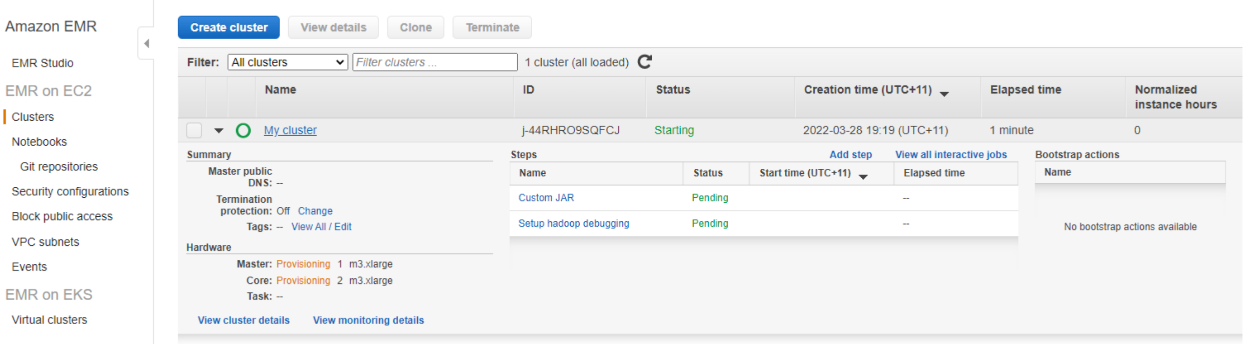
Persistent user interfaces: --
On-cluster user interfaces: --

Network and hardware

Availability zone: --
Subnet ID: subnet-07b4f3e77305888a3
Master: Provisioning 1 m3.xlarge
Core: Provisioning 2 m3.xlarge
Task: --
Cluster scaling: Not enabled
Auto-termination: Terminate if idle for 1 hour

Security and access

Key name: --
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master:
Security groups for Core & Task:



Filter: All clusters | Filter clusters ... | 1 cluster (all loaded)

Name	ID	Status	Creation time (UTC+11)	Elapsed time	Normalized instance hours
My cluster	j-44RHRO9SQFCJ	Starting	2022-03-28 19:19 (UTC+11)	1 minute	0

Summary

Master public DNS: --
Termination protection: Off Change
Tags: -- View All / Edit

Hardware

Master: Provisioning 1 m3.xlarge
Core: Provisioning 2 m3.xlarge
Task: --

View cluster details View monitoring details

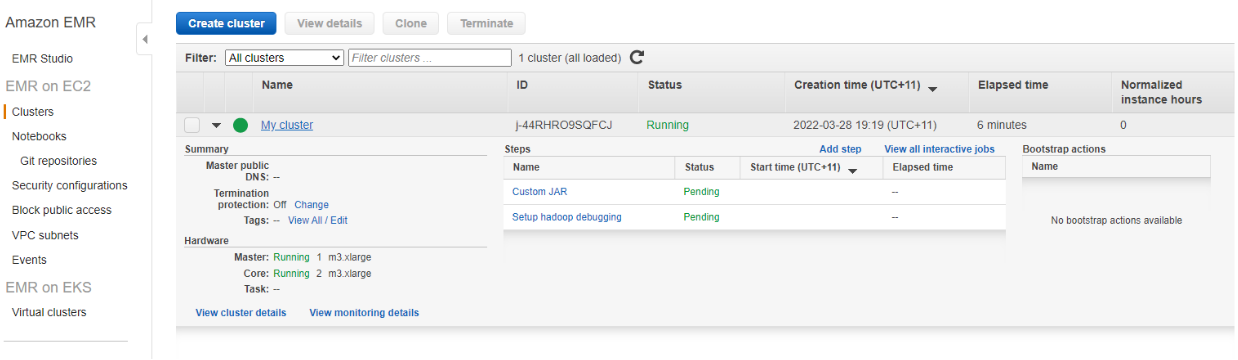
Steps Add step View all interactive jobs

Name	Status	Start time (UTC+11)	Elapsed time
Custom JAR	Pending		--
Setup hadoop debugging	Pending		--

Bootstrap actions

Name
No bootstrap actions available

When MapReduce job will start the status will show "Running" like following screen.



Name	ID	Status	Creation time (UTC+11)	Elapsed time	Normalized instance hours
My cluster	j-44RHRO9SQFCJ	Running	2022-03-28 19:19 (UTC+11)	6 minutes	0

Summary

Master public DNS: --

Termination protection: Off Change

Tags: -- View All / Edit

Hardware

Master: Running 1 m3.xlarge

Core: Running 2 m3.xlarge

Task: --

Steps — Add step — View all interactive jobs

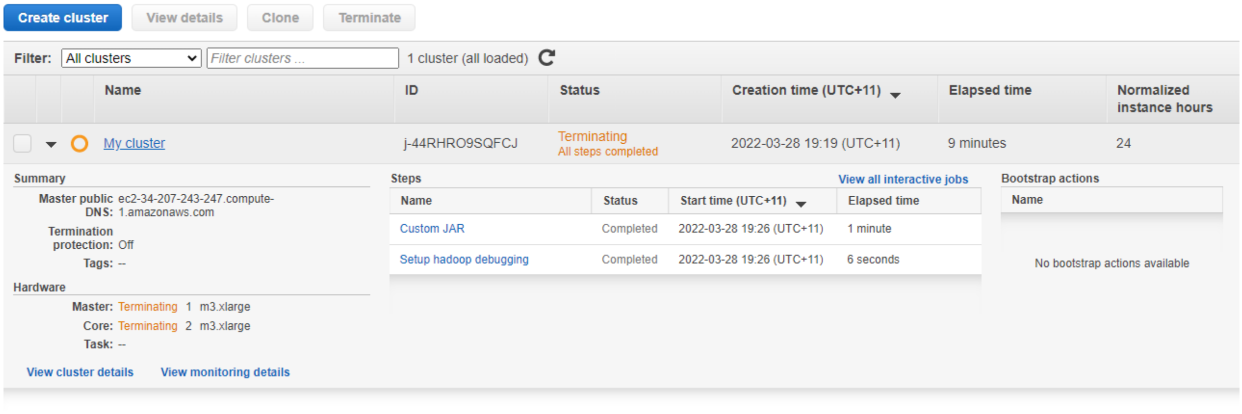
Name	Status	Start time (UTC+11)	Elapsed time
Custom JAR	Pending		--
Setup hadoop debugging	Pending		--

Bootstrap actions

Name
No bootstrap actions available

View cluster details — View monitoring details

Wait for some more time. If the status shows "Terminated steps completed" like following screen, it means MapReduce job has finished and the outputs are now ready for viewing.



Name	ID	Status	Creation time (UTC+11)	Elapsed time	Normalized instance hours
My cluster	j-44RHRO9SQFCJ	Terminating All steps completed	2022-03-28 19:19 (UTC+11)	9 minutes	24

Summary

Master public DNS: ec2-34-207-243-247.compute-1.amazonaws.com

Termination protection: Off

Tags: --

Hardware

Master: Terminating 1 m3.xlarge

Core: Terminating 2 m3.xlarge

Task: --

Steps — View all interactive jobs

Name	Status	Start time (UTC+11)	Elapsed time
Custom JAR	Completed	2022-03-28 19:26 (UTC+11)	1 minute
Setup hadoop debugging	Completed	2022-03-28 19:26 (UTC+11)	6 seconds

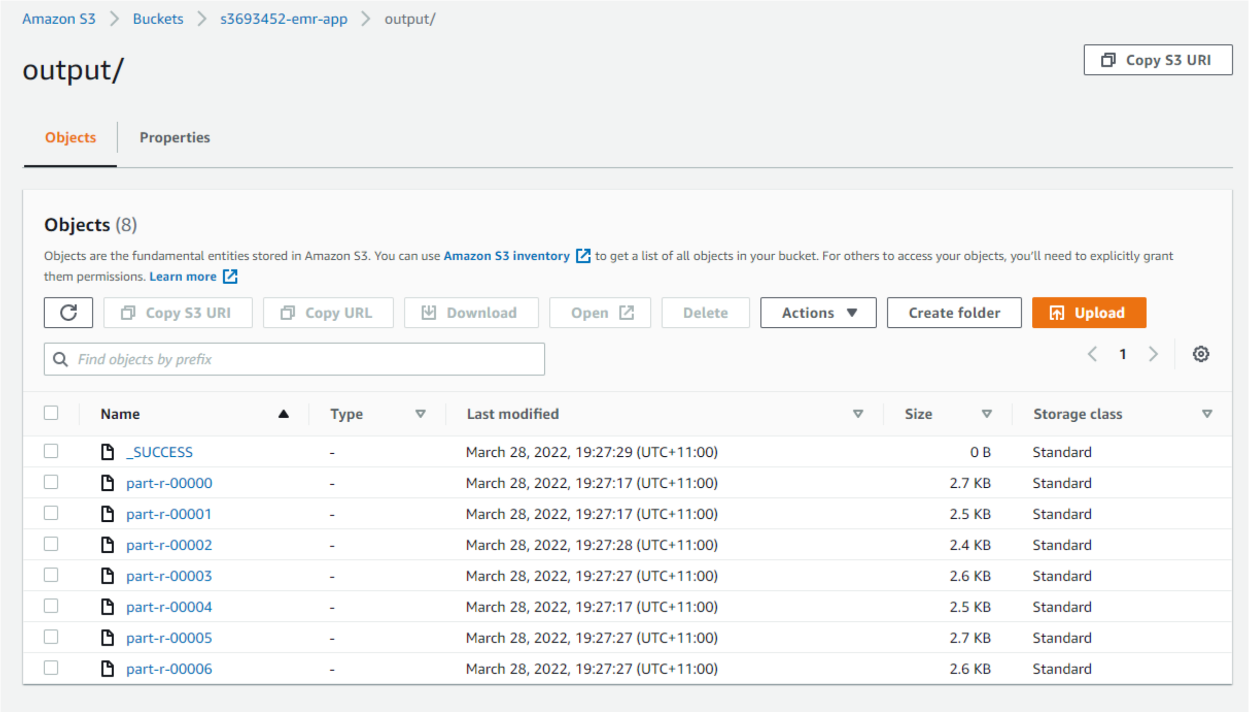
Bootstrap actions

Name
No bootstrap actions available

View cluster details — View monitoring details

2.3 View the output result

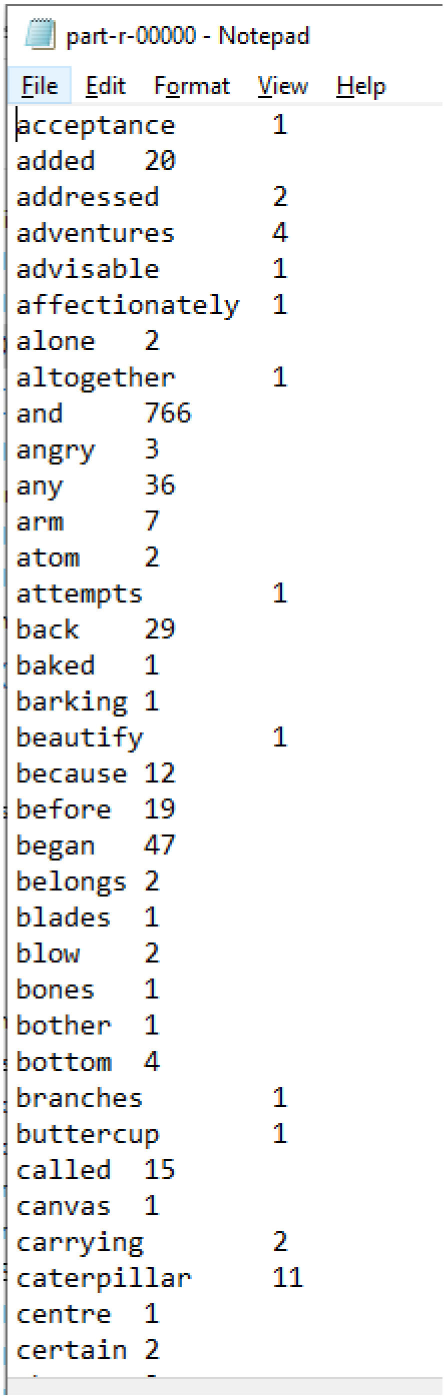
Once your job is marked as completed, go to AWS Management Console via AWS Academy portal, Select **AWS S3**, and click on the **output** folder you specified. You will see some files named part-0000, part-0001,.. which contain the results of your MapReduce job. The number of output files generated depends on the **number of reducers** at work to complete a given MapReduce application. This is managed by EMR.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
□	_SUCCESS	-	March 28, 2022, 19:27:29 (UTC+11:00)	0 B	Standard
□	part-r-00000	-	March 28, 2022, 19:27:17 (UTC+11:00)	2.7 KB	Standard
□	part-r-00001	-	March 28, 2022, 19:27:17 (UTC+11:00)	2.5 KB	Standard
□	part-r-00002	-	March 28, 2022, 19:27:28 (UTC+11:00)	2.4 KB	Standard
□	part-r-00003	-	March 28, 2022, 19:27:27 (UTC+11:00)	2.6 KB	Standard
□	part-r-00004	-	March 28, 2022, 19:27:17 (UTC+11:00)	2.5 KB	Standard
□	part-r-00005	-	March 28, 2022, 19:27:27 (UTC+11:00)	2.7 KB	Standard
□	part-r-00006	-	March 28, 2022, 19:27:27 (UTC+11:00)	2.6 KB	Standard

You can download all the files and open them in a text editor to view the results. The results contain the word and number of occurrences of that word in "alice30.txt" file, as below.



```
acceptance      1
added   20
addressed       2
adventures      4
advisable       1
affectionately  1
alone   2
altogether      1
and     766
angry   3
any     36
arm     7
atom    2
attempts        1
back    29
baked   1
barking 1
beautify        1
because 12
before  19
began   47
belongs 2
blades  1
blow    2
bones   1
bother  1
bottom  4
branches        1
buttercup       1
called  15
canvas  1
carrying        2
caterpillar     11
centre  1
certain 2
```

Important note: The commands we run to launch our MapReduce job in this example are set to automatically terminate the EC2 instances. However, as you get into more custom commands, this will not necessarily be the case. In those situations, you **HAVE TO** terminate the instances yourself. Otherwise, you will be charged for the time these instances are running (even if the job completed). To terminate an instance, you can Go on Amazon Management Console via AWS Academy portal → Elastic Map Reduce, select your Job Flow and click on Terminate.