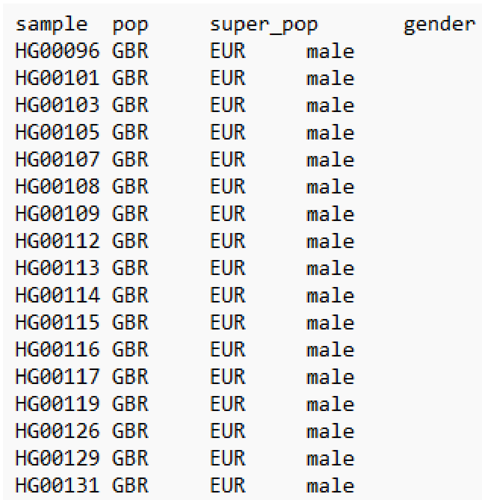
USING MAPREDUCE IN THE 1000 GENOMES DATASET

1. INTRODUCTION

The 1000 genomes project is an international research project that created a catalogue of common human genetic variation. Their data is available for free in their website (https://www.internationalgenome.org/)

This practical session aims to determine the distribution of samples across different populations in the 1000 Genomes dataset. Nevertheless, for the session a small portion of the dataset called “we will use a small sample file called integrated_call_male_samples_v3.20130502.ALL.panel” was used.This file contains information about the samples in the dataset, which are grouped into populations and super populations based on their geographic origin.

This is how the data looks like:



```
sample  pop     super_pop       gender
HG00096 GBR     EUR     male
HG00101 GBR     EUR     male
HG00103 GBR     EUR     male
HG00105 GBR     EUR     male
HG00107 GBR     EUR     male
HG00108 GBR     EUR     male
HG00109 GBR     EUR     male
HG00112 GBR     EUR     male
HG00113 GBR     EUR     male
HG00114 GBR     EUR     male
HG00115 GBR     EUR     male
HG00116 GBR     EUR     male
HG00117 GBR     EUR     male
HG00119 GBR     EUR     male
HG00126 GBR     EUR     male
HG00129 GBR     EUR     male
HG00131 GBR     EUR     male
```

The objective of this session is to count how many samples are there for each country implementing the **MapReduce** with AWS in a simple way.

MapReduce is a programming model and associated implementation for processing and generating large datasets that are parallelizable. Here is a brief overview:

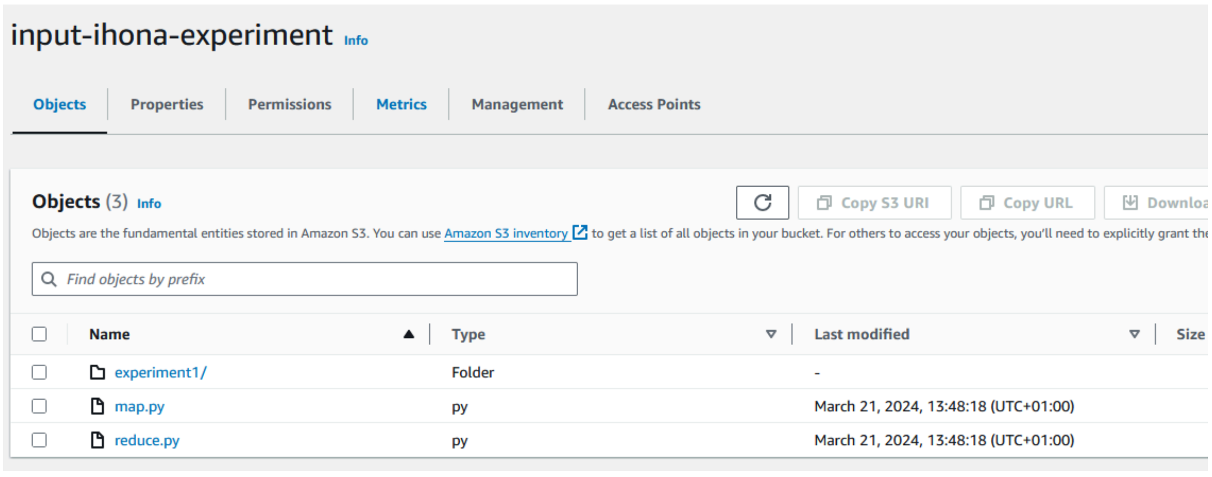
1. *Mapping*: In the mapping phase, data is broken down into smaller chunks and distributed across multiple machines in a cluster. Each machine performs a specific task (map) on its subset of data and generates intermediate key-value pairs.
2. *Shuffling*: In this phase, the intermediate results from the mapping phase are shuffled and sorted based on keys. This process ensures that all values associated with a particular key are grouped together.
3. *Reducing*: In the reducing phase, the shuffled intermediate data is processed to produce the final output. Each reducer machine processes a subset of keys and their associated values, aggregates them, and generates the final output.

On the other hand, **Hadoop** is an open-source distributed computing framework designed for storing and processing large datasets across clusters of commodity hardware, which uses the MapReduce as its processing engine. In AWS, we can use **EMR** (Amazon Elastic MapReduce), that simplifies the deployment and management of Hadoop and other distributed computing frameworks, including support for the MapReduce programming model. It enables users to efficiently process and analyze large datasets using parallel computing techniques across a dynamically scalable cluster of virtual servers.

For this exercise, we will use EMR for processing our dataset. Moreover, **Python** is used for the MapReduce codes that will process the data.

2. CREATING THE BUCKETS

Before starting with the EMR, we must create two buckets, one for storing the input data (dataset, map and reduce python codes) and another one to store the results of the MapReducer. We have to take into consideration that the buckets will need to be in the same region as the cluster.



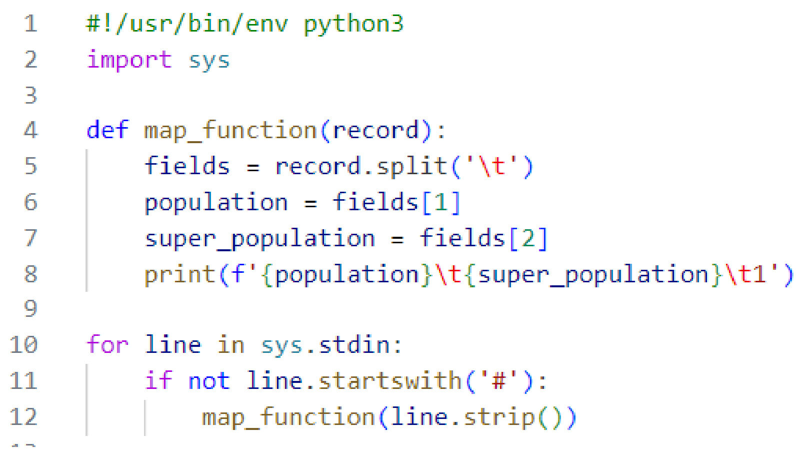
Objects (3) Info

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 the

Name	Type	Last modified	Size
experiment1/	Folder	-	
map.py	py	March 21, 2024, 13:48:18 (UTC+01:00)	
reduce.py	py	March 21, 2024, 13:48:18 (UTC+01:00)	

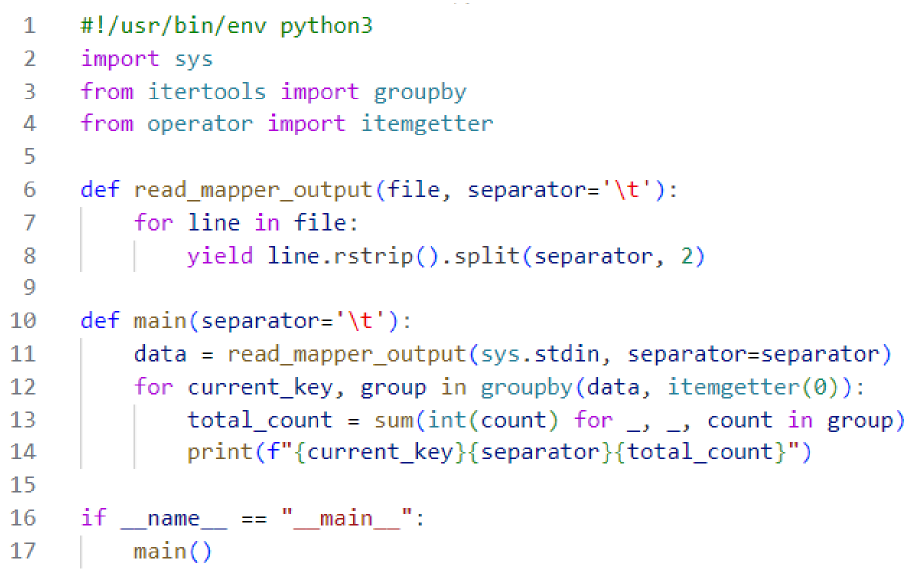
The map and reduce functions can either be in the same code, or separately. In this exercise, it has been used two separate functions, map.py for the map function, and reduce.py for the reducer function:

- map.py: This code goes line by line, and if the line is not a comment (#) it launches the map function. The map function splits the line using a separator and stores the population and the super population (which according to the dataset, they are in the second and third column) in separate variables.



```python
#!/usr/bin/env python3
import sys

def map_function(record):
    fields = record.split('\t')
    population = fields[1]
    super_population = fields[2]
    print(f'{population}\t{super_population}\t1')

for line in sys.stdin:
    if not line.startswith('#'):
        map_function(line.strip())
```

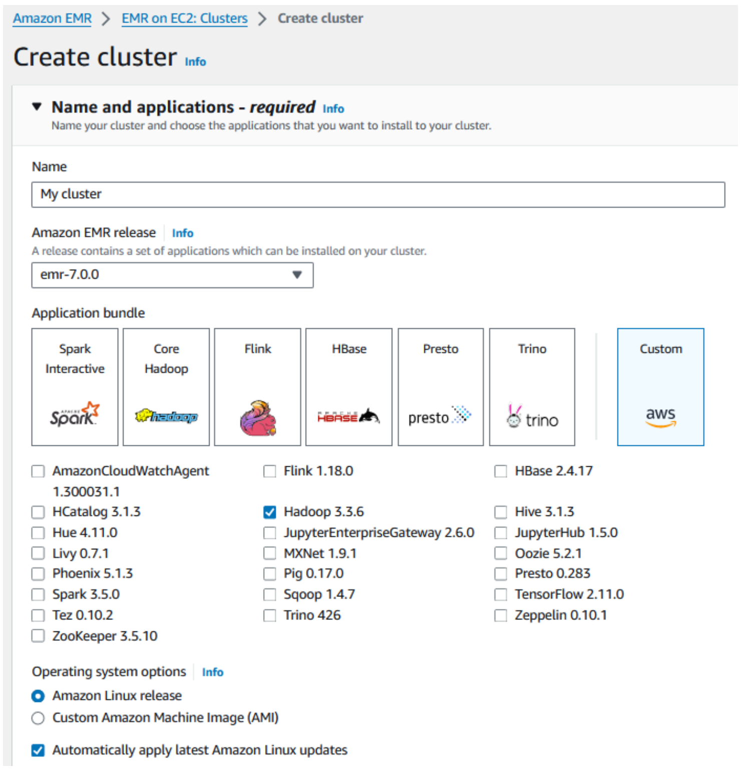
- reduce.py: The reducer reads everything received from the maper and groups it together using the key. Once it has all the groups together, it sums the number of occurrences.



```python
#!/usr/bin/env python3
import sys
from itertools import groupby
from operator import itemgetter

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 2)

def main(separator='\t'):
    data = read_mapper_output(sys.stdin, separator=separator)
    for current_key, group in groupby(data, itemgetter(0)):
        total_count = sum(int(count) for _, _, count in group)
        print(f"{current_key}{separator}{total_count}")

if __name__ == "__main__":
    main()
```

3. EMR

Now, we have to go to the EMR service and configure it to use these buckets. To make it simpler and for the demo purpose, we select the hadoop cluster:



Amazon EMR > EMR on EC2: Clusters > Create cluster

Create cluster Info

▼ **Name and applications - *required*** Info

Name your cluster and choose the applications that you want to install to your cluster.

Name: My cluster

Amazon EMR release Info

A release contains a set of applications which can be installed on your cluster.

emr-7.0.0

Application bundle

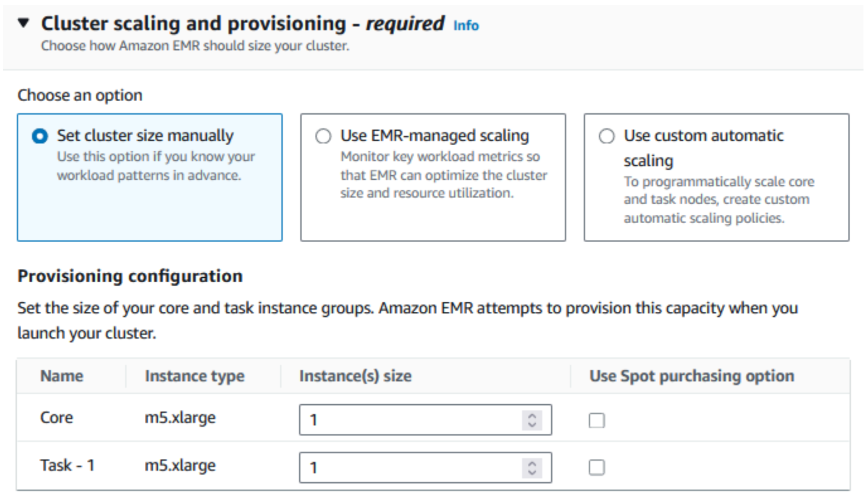
- [ ] AmazonCloudWatchAgent 1.300031.1
- [ ] HCatalog 3.1.3
- [ ] Hue 4.11.0
- [ ] Livy 0.7.1
- [ ] Phoenix 5.1.3
- [ ] Spark 3.5.0
- [ ] Tez 0.10.2
- [ ] ZooKeeper 3.5.10
- [ ] Flink 1.18.0
- [✓] Hadoop 3.3.6
- [ ] JupyterEnterpriseGateway 2.6.0
- [ ] MXNet 1.9.1
- [ ] Pig 0.17.0
- [ ] Sqoop 1.4.7
- [ ] Trino 426
- [ ] HBase 2.4.17
- [ ] Hive 3.1.3
- [ ] JupyterHub 1.5.0
- [ ] Oozie 5.2.1
- [ ] Presto 0.283
- [ ] TensorFlow 2.11.0
- [ ] Zeppelin 0.10.1

Operating system options Info

[✓ Amazon Linux release]
Custom Amazon Machine Image (AMI)

[✓] Automatically apply latest Amazon Linux updates

We set the cluster size manually:



▼ **Cluster scaling and provisioning - *required*** Info

Choose how Amazon EMR should size your cluster.

Choose an option

[✓ Set cluster size manually] Use this option if you know your workload patterns in advance.

Use EMR-managed scaling: Monitor key workload metrics so that EMR can optimize the cluster size and resource utilization.

Use custom automatic scaling: To programmatically scale core and task nodes, create custom automatic scaling policies.

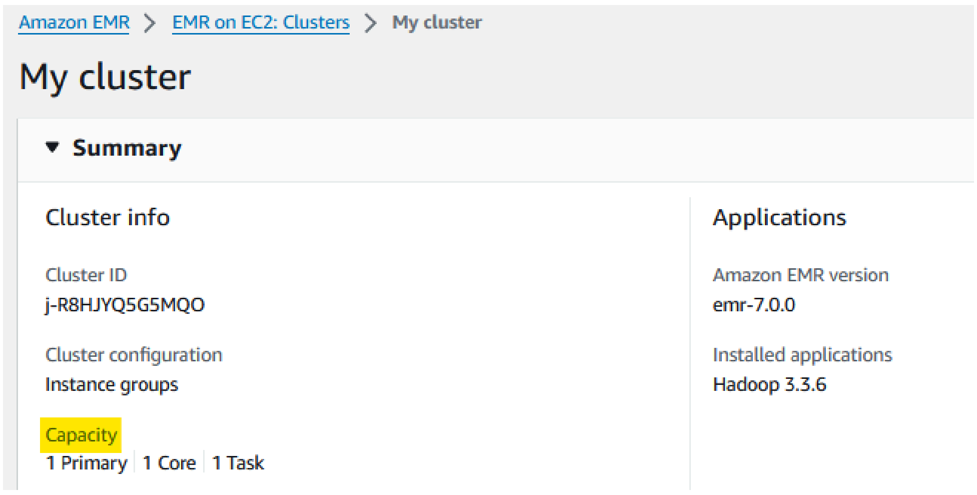
Provisioning configuration

Set the size of your core and task instance groups. Amazon EMR attempts to provision this capacity when you launch your cluster.

Name	Instance type	Instance(s) size	Use Spot purchasing option
Core	m5.xlarge	1	☐
Task - 1	m5.xlarge	1	☐

After created, the cluster can take some minutes to start. When creating a cluster, the logs of that cluster will be saved in an s3 bucket automatically created.

With the configuration selected, we have specified 3 nodes (primary, code and task):


Amazon EMR > EMR on EC2: Clusters > My cluster

My cluster

Summary

Cluster info

Cluster ID
j-R8HJYQ5G5MQO

Cluster configuration
Instance groups

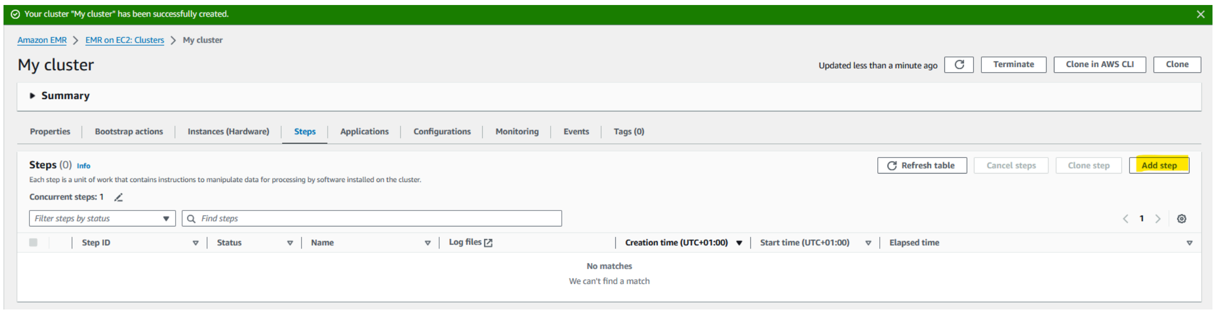
Capacity
1 Primary | 1 Core | 1 Task

Applications

Amazon EMR version
emr-7.0.0

Installed applications
Hadoop 3.3.6

The next step is to prepare the job to be run. For that, we need to add a step to the cluster:


Your cluster "My cluster" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > My cluster

My cluster

Updated less than a minute ago

Summary

Steps (0) Info

Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.

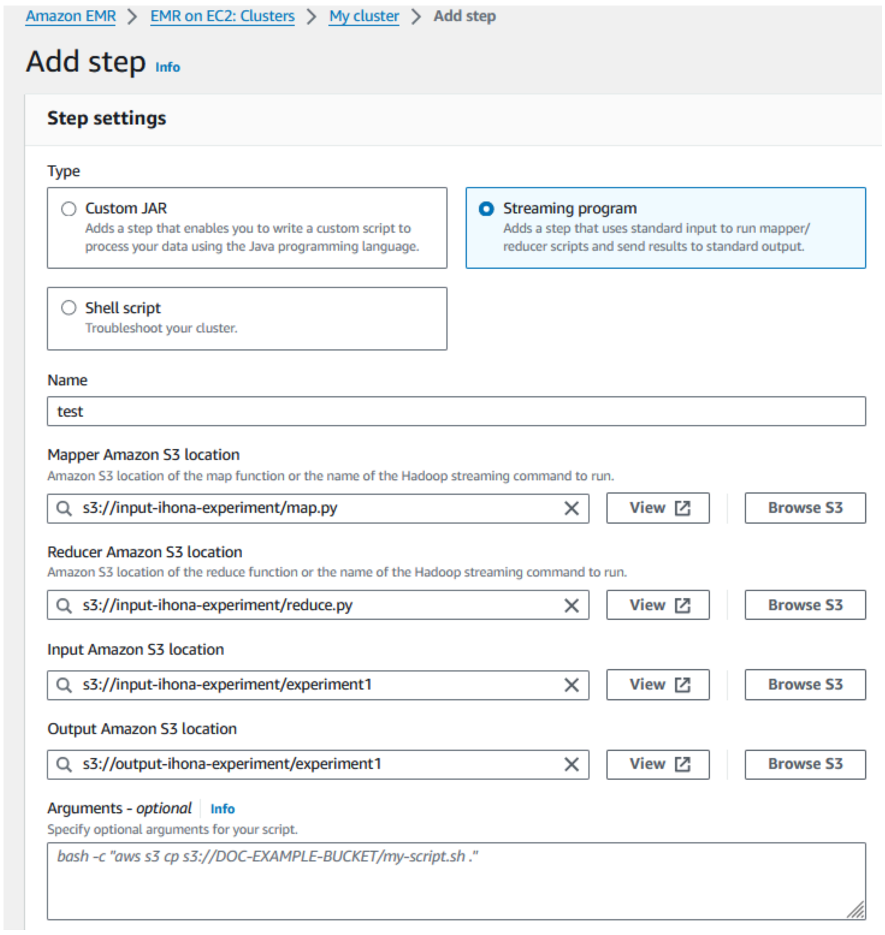
Concurrent steps: 1

	Step ID	Status	Name	Log files	Creation time (UTC+01:00)	Start time (UTC+01:00)	Elapsed time

No matches
We can't find a match

When creating the step, we have to select the “streaming program” because in this way we will be able to use the standard input (remember that tmap.py and reduce.py read and write in the standard input). Besides the streaming program, we specify:

- The mapper (map.py), located in the input bucket.
- The reducer (reduce.py), also located in the input bucket.
- The bucket where the dataset is located.
- The output bucket to store the results. It is really customizable, we can specify that in the output file, it should create a new folder named “experiment1” to store there the results.



Amazon EMR > EMR on EC2: Clusters > My cluster > Add step

Add step Info

Step settings

Type

- Custom JAR — Adds a step that enables you to write a custom script to process your data using the Java programming language.
- [✓ Streaming program] — Adds a step that uses standard input to run mapper/reducer scripts and send results to standard output.
- Shell script — Troubleshoot your cluster.

Name: test

Mapper Amazon S3 location
Amazon S3 location of the map function or the name of the Hadoop streaming command to run.
s3://input-ihona-experiment/map.py

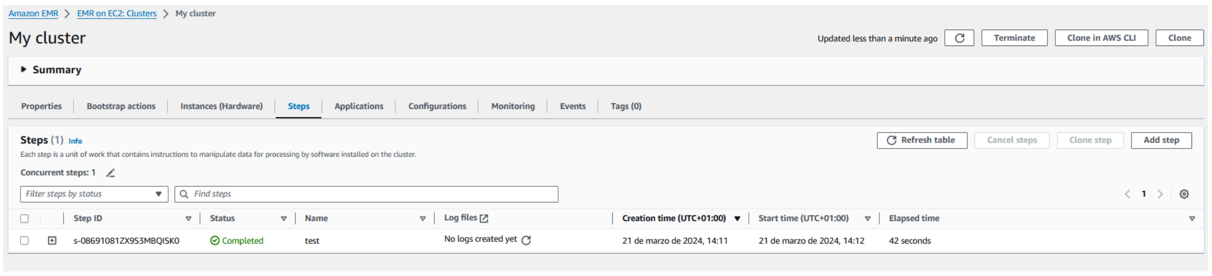
Reducer Amazon S3 location
Amazon S3 location of the reduce function or the name of the Hadoop streaming command to run.
s3://input-ihona-experiment/reduce.py

Input Amazon S3 location
s3://input-ihona-experiment/experiment1

Output Amazon S3 location
s3://output-ihona-experiment/experiment1

Arguments - *optional* Info
Specify optional arguments for your script.
bash -c "aws s3 cp s3://DOC-EXAMPLE-BUCKET/my-script.sh ."

We save the task:



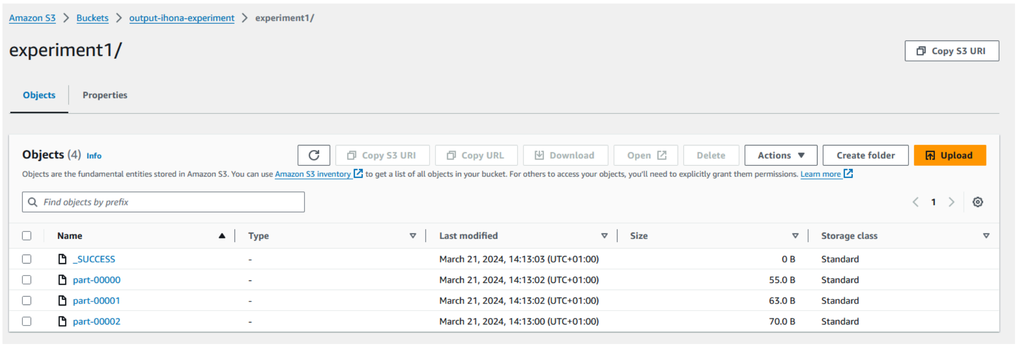
Amazon EMR > EMR on EC2: Clusters > My cluster

My cluster — Updated less than a minute ago

Steps (1) Info
Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.
Concurrent steps: 1

Step ID	Status	Name	Log files	Creation time (UTC+01:00)	Start time (UTC+01:00)	Elapsed time
s-08691081ZX9S3MBQISKO	Completed	test	No logs created yet	21 de marzo de 2024, 14:11	21 de marzo de 2024, 14:12	42 seconds

Once the task has finished, it stores the result in the bucket specified:



Amazon S3 > Buckets > output-ihona-experiment > experiment1/

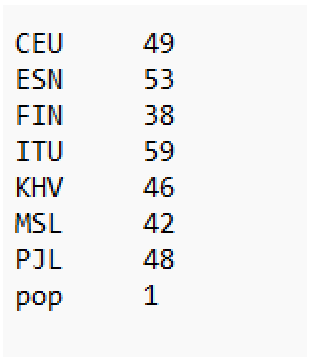
experiment1/

Objects (4) Info
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 21, 2024, 14:13:03 (UTC+01:00)	0 B	Standard
part-00000	-	March 21, 2024, 14:13:02 (UTC+01:00)	55.0 B	Standard
part-00001	-	March 21, 2024, 14:13:02 (UTC+01:00)	63.0 B	Standard
part-00002	-	March 21, 2024, 14:13:00 (UTC+01:00)	70.0 B	Standard

It is possible to see that the MapReducer has created inside the output bucket a folder named experimetn1 and it has stored there the results. There are 3 results (or parts), because we have a cluster with 3 nodes.

If we open one of the files we can see the count for each country:



```
CEU     49
ESN     53
FIN     38
ITU     59
KHV     46
MSL     42
PJL     48
pop     1
```

4. VISUALIZE RESULTS WITH JUPYTER NOTEBOOK

Finally, we will visualize the results of the MapReducer in a jupyter notebook inside an EC2 instance.

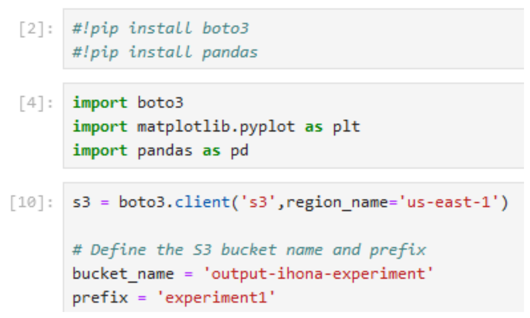
To do that, we first need to connect from our terminal, via SSH, to that EC2 instance. Then, we activate our Python environment, which has Jupyter already installed. Afterwards, we open jupyter from the terminal by running the following command:

jupyter notebook --ip=0.0.0.0

After that, we can access jupyter via the Public IPv4 DNS of the instance, specifying the port :8888 and entering the token received in the terminal.

Since we will have to retrieve the data through the s3 bucket, we have to also run the command “aws configure” and set the aws_access_key_id , aws_secret_access_key and aws_session_token to grant access for different AWS services to interact.

Once everything is set, we create a new notebook, install and import the necessary libraries and set the destination where our results from the MapReducer were saved:

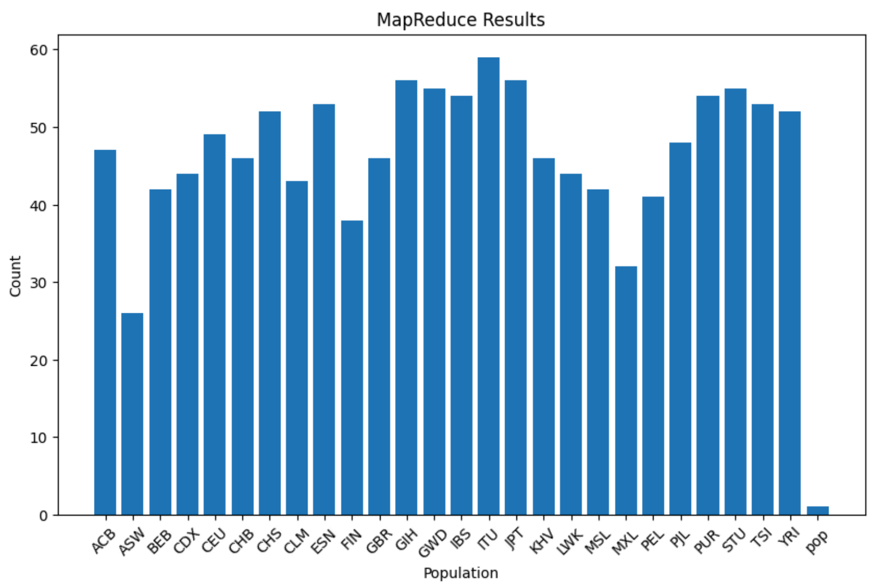


```
[2]:  #!pip install boto3
      #!pip install pandas

[4]:  import boto3
      import matplotlib.pyplot as plt
      import pandas as pd

[10]: s3 = boto3.client('s3',region_name='us-east-1')

      # Define the S3 bucket name and prefix
      bucket_name = 'output-ihona-experiment'
      prefix = 'experiment1'
```

Then, we get a list of all object keys in the bucket with the specified prefix and initialize an empty list to store the data. The code loops through the object keys and reads the data into a list. Afterwards, we parse the line into the population and count columns.

Finally, the data is transformed into a dataframe, we group the data by the population column and sum the count column.

We can create a bar plot with the results:



5. REFERENCES

This exercise was proposed and provided by Jordi Mateo, professor of the subject High Performance and Distributed Computing for Big Data (URV).

https://github.com/JordiMateoUdL