Assignment-1 (Fractal - 3)
M22MA003

Question 1: MapReduce mathematical algorithm for the mapper stage and reducer stage to perform various analyses on the dataset efficiently.

Given Data Format (Weblog Dataset) :-

IP	Time	URL	Staus
10.128.2.1	[29/Nov/2017:06:58:55	GET /login.php HTTP/1.1	200
10.128.2.1	[29/Nov/2017:06:59:02	POST /process.php HTTP/1.1	302
10.128.2.1	[29/Nov/2017:06:59:03	GET /home.php HTTP/1.1	200
10.131.2.1	[29/Nov/2017:06:59:04	GET /js/vendor/moment.min.js HTTP/1.1	200
10.130.2.1	[29/Nov/2017:06:59:06	GET /bootstrap-3.3.7/js/bootstrap.js HTTP/1.1	200
10.130.2.1	[29/Nov/2017:06:59:19	GET /profile.php?user=bala HTTP/1.1	200
10.128.2.1	[29/Nov/2017:06:59:19	GET /js/jquery.min.js HTTP/1.1	200
10.131.2.1	[29/Nov/2017:06:59:19	GET /js/chart.min.js HTTP/1.1	200
10.131.2.1	[29/Nov/2017:06:59:30	GET /edit.php?name=bala HTTP/1.1	200
10.131.2.1	[29/Nov/2017:06:59:37	GET /logout.php HTTP/1.1	302

218-count invalid entries are found in the dataset. Few examples are:-
chmod:,cannot,'a.out':,No
chmod:,cannot,'error.txt':,No
rm:,cannot,'*.o':,No

Task 1. Count requests by IP address:

Objective: The goal of this analysis is to count the number of requests made by each unique IP address in the given dataset.

Mapper Function:
The Mapper function accepts as input a single record, which is a log entry with fields like IP address, date, URL, and success code. The Mapper function's job is to pull the IP address out of each record and send it out as a key-value pair, with the IP address as the key and 1 as the value. The MapReduce framework will automatically group the key-value pairs based on the keys, which are the IP addresses, and send them to the Reducer function to be processed further.

```
function map(record):
    ip_address = record['IP']
    emit(ip_address, 1)
```

Reducer Function:
The IP address is used as the key, and a list of numbers (in this case, a list of 1s) is used as the value list. The Reducer's job is to add up all of the data for each IP address. This tells us how many requests were made by that IP address. The Reducer method will send out the IP address and the total amount for each IP address.

```
function reduce(ip_address, counts):
    total_count = sum(counts)
    emit(ip_address, total_count)
```

The final result of this MapReduce job will be a list of key-value pairs. Each key will be an IP address, and each value will be the number of times that IP address made a request in the dataset.

Logs:-

Job ID: job_1690633086408_0004

```
23/07/29 14:49:48 INFO exec.Task: Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1690633086408_0004
23/07/29 14:50:00 INFO exec.Task: Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
23/07/29 14:50:00 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapredu
ce.TaskCounter instead
23/07/29 14:50:00 INFO exec.Task: 2023-07-29 14:50:00,803 Stage-1 map = 0%,  reduce = 0%
23/07/29 14:50:10 INFO exec.Task: 2023-07-29 14:50:10,727 Stage-1 map = 100%,  reduce = 0%
23/07/29 14:50:22 INFO exec.Task: 2023-07-29 14:50:22,762 Stage-1 map = 100%,  reduce = 100%
23/07/29 14:50:23 INFO exec.Task: Ended Job = job_1690633086408_0004
23/07/29 14:50:23 INFO ql.Driver: MapReduce Jobs Launched:
23/07/29 14:50:23 INFO ql.Driver: Stage-Stage-1: Map: 1  Reduce: 1   HDFS Read: 1073710 HDFS Write: 166 SUCCESS
23/07/29 14:50:23 INFO ql.Driver: Total MapReduce CPU Time Spent: 0 msec
23/07/29 14:50:23 INFO ql.Driver: Completed executing command(queryId=hive_20230729144949_8b0b03bf-f233-46a2-8066-3cfcf879232e); Ti
me taken: 36.814 seconds
23/07/29 14:50:23 INFO ql.Driver: OK
```

(Observation : There are 218 invalid entries/records in the given dataset).

Output:-

ip	request_count
10.128.2.1	4257
10.129.2.1	1652
10.130.2.1	4056
10.131.0.1	4198
10.131.2.1	1626

Task 2. Identify the top 10 IP addresses with the most requests:

Objective: The goal of this analysis is to find the top 10 IP addresses with the highest number of requests in the given dataset.

Mapper Function:
The Mapper method takes in a single record, which is a log entry with different fields, including the IP address. The Mapper function's job is to pull the IP address out of each record and send it out as a key-value pair, with the IP address as the key and 1 as the value. The MapReduce

framework will automatically group the key-value pairs based on the keys, which are the IP addresses, and send them to the Reducer function to be processed further.

```
function map(record):
    ip_address = record['IP']
    emit(ip_address, 1)
```

Reducer Function:
The IP address is used as the key, and a list of numbers (in this case, a list of 1s) is used as the value list. The Reducer's job is to add up all of the data for each IP address. This tells us how many requests were made by that IP address. But this time, instead of sending out the total number, we will use a heap data structure to keep track of the top 10 IP addresses with the most requests.

```
function reduce(ip_address, counts):
    total_count = sum(counts)
    save_to_heap(ip_address, total_count)  # Keep track of the top 10 IP addresses in a heap
```

The "save_to_heap" method is a placeholder for the code that will keep track of the top 10 IP addresses with the most requests. It is usually done with a min heap that has a set amount of 10 elements. As new IP addresses are handled, the method will add them to the heap while making sure that the heap size stays at 10. If a new IP address has more requests than the smallest value in the list, the smallest value will be replaced by the new IP address. At the end, the top 10 IP addresses with the most requests will be in the heap.

Logs:-

Job ID: job_1690633086408_0006

```
23/07/29 14:55:52 INFO exec.Task: Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
23/07/29 14:55:53 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapredu
ce.TaskCounter instead
23/07/29 14:55:53 INFO exec.Task: 2023-07-29 14:55:53,016 Stage-2 map = 0%,  reduce = 0%
23/07/29 14:56:02 INFO exec.Task: 2023-07-29 14:56:02,806 Stage-2 map = 100%,  reduce = 0%
23/07/29 14:56:14 INFO exec.Task: 2023-07-29 14:56:14,742 Stage-2 map = 100%,  reduce = 100%
23/07/29 14:56:15 INFO exec.Task: Ended Job = job_1690633086408_0006
23/07/29 14:56:15 INFO ql.Driver: MapReduce Jobs Launched:
23/07/29 14:56:15 INFO ql.Driver: Stage-Stage-1: Map: 1  Reduce: 1   HDFS Read: 1073134 HDFS Write: 510 SUCCESS
23/07/29 14:56:15 INFO ql.Driver: Stage-Stage-2: Map: 1  Reduce: 1   HDFS Read: 5185 HDFS Write: 118 SUCCESS
23/07/29 14:56:15 INFO ql.Driver: Total MapReduce CPU Time Spent: 0 msec
23/07/29 14:56:15 INFO ql.Driver: Completed executing command(queryId=hive_20230729145555_87b4360b-d3c2-46bf-92ab-77b8adb6a202); Ti
me taken: 72.439 seconds
23/07/29 14:56:15 INFO ql.Driver: OK
```

Output:-

ip	request_count
10.128.2.1	4257
10.131.0.1	4198
10.130.2.1	4056
10.129.2.1	1652
10.131.2.1	1626

Task 3. Find the top 10 URLs that were requested the most:

Objective: The goal of this analysis is to find the top 10 URLs with the highest number of requests in the given dataset.

Mapper Function:
The Mapper function takes in a single record, which is a log entry with different values, such as the URL asked. The Mapper function's job is to get the URL from each record and send it out as a key-value pair, with the URL as the key and 1 as the value. The MapReduce framework will automatically group the key-value pairs based on the keys (URLs), and then send them to the Reducer function for further processing.

```
function map(record):
    url = record['URL']
    emit(url, 1)
```

Reducer Function:
The key for the Reducer method is a URL, and the numbers for that key are a list of 1s in this case. The Reducer's job is to add up all of the numbers for each URL. This tells us how many times that URL has been asked for. But this time, instead of sending out the total number, we will use a heap data structure to keep track of the top 10 URLs with the most calls.

```
function reduce(url, counts):
    total_count = sum(counts)
    save_to_heap(url, total_count)
```

The "save_to_heap" method is a placeholder for the code that will keep track of the top 10 URLs with the most requests. It is usually done with a min heap that has a set amount of 10 elements. As new URLs are handled, the method will add them to the heap while making sure the heap size stays at 10. If a new URL has more requests than the smallest value in the list, the smallest value will be replaced by the new URL. In the end, the top 10 URLs with the most hits will be in the heap.

Logs:-

Job ID: job_1690633086408_0008

```
23/07/29 15:02:07 INFO exec.Task: Starting Job = job_1690633086408_0008, Tracking URL = http://quickstart.cl
cation_1690633086408_0008/
23/07/29 15:02:07 INFO exec.Task: Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1690633086408_000
23/07/29 15:02:19 INFO exec.Task: Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
23/07/29 15:02:19 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapredu
ce.TaskCounter instead
23/07/29 15:02:19 INFO exec.Task: 2023-07-29 15:02:19,712 Stage-2 map = 0%,  reduce = 0%
23/07/29 15:02:29 INFO exec.Task: 2023-07-29 15:02:29,461 Stage-2 map = 100%,  reduce = 0%
23/07/29 15:02:40 INFO exec.Task: 2023-07-29 15:02:40,284 Stage-2 map = 100%,  reduce = 100%
23/07/29 15:02:41 INFO exec.Task: Ended Job = job_1690633086408_0008
23/07/29 15:02:41 INFO ql.Driver: MapReduce Jobs Launched:
23/07/29 15:02:41 INFO ql.Driver: Stage-Stage-1: Map: 1  Reduce: 1   HDFS Read: 1073141 HDFS Write: 17879 SUCCESS
23/07/29 15:02:41 INFO ql.Driver: Stage-Stage-2: Map: 1  Reduce: 1   HDFS Read: 22558 HDFS Write: 391 SUCCESS
23/07/29 15:02:41 INFO ql.Driver: Total MapReduce CPU Time Spent: 0 msec
23/07/29 15:02:41 INFO ql.Driver: Completed executing command(queryId=hive_20230729150101_de2bb3e8-dd1c-419c-9063-7384a492cb9d); Ti
me taken: 71.529 seconds
23/07/29 15:02:41 INFO ql.Driver: OK
```

Output:-

url	request_count
GET /login.php HTTP/1.1	3284
GET /home.php HTTP/1.1	2640
GET /js/vendor/modernizr-2.8.3.min.js HTTP/1.1	1415
GET / HTTP/1.1	861
GET /contestproblem.php?name=RUET%20OJ%20Server%20Testing%20Contest HTTP/1.1	467
GET /css/normalize.css HTTP/1.1	408
GET /css/bootstrap.min.css HTTP/1.1	404
GET /css/font-awesome.min.css HTTP/1.1	399
GET /css/style.css HTTP/1.1	395
GET /css/main.css HTTP/1.1	394

Task 4. Discover the top 10 status codes returned:

Objective: The goal of this analysis is to find the top 10 status codes with the highest occurrence in the given dataset.

Mapper Function:
The input for the Mapper method is a single record, which is a log entry with different fields, including the status code given. The Mapper function's job is to pull the status code out of each record and send it out as a key-value pair, with the status code as the key and 1 as the value.The MapReduce framework will automatically put the key-value pairs that are sent out into groups based on the keys, which are the status codes. These groups will then be sent to the Reducer function to be processed further.

```
function map(record):
    status_code = record['Status']
    emit(status_code, 1)
```

Reducer Function:
The key for the Reducer method is a status code, and the numbers for that key are a list of 1s in this case. The Reducer's job is to add up the numbers for each status code. This tells us how many times that status code has been seen. But this time, instead of sending out the total number, we will use a heap data structure to keep track of the top 10 status codes that are used the most.

```
function reduce(status_code, counts):
    total_count = sum(counts)
    save_to_heap(status_code, total_count)  # Keep track of the top 10 status codes in a heap
```

The "save_to_heap" method is a placeholder for the reasoning that will keep track of the top 10 most-used status codes. It is usually done with a min heap that has a set amount of 10 elements. As new status codes are handled, the method will add them to the heap while making sure the heap size stays at 10. If a new status code has more occurrences than the smallest value in the heap, the smallest value will be replaced by the new status code. At the end, the stack will have the top 10 status numbers that are used the most.

Logs:-

Job ID: job_1690633086408_0009

```
23/07/29 15:07:42 INFO exec.Task: Starting Job = job_1690633086408_0009, Tracking URL = http://quickstart.cloudera:8088/proxy/appli
cation_1690633086408_0009/
23/07/29 15:07:42 INFO exec.Task: Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1690633086408_0009
23/07/29 15:07:54 INFO exec.Task: Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
23/07/29 15:07:54 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapredu
ce.TaskCounter instead
23/07/29 15:07:54 INFO exec.Task: 2023-07-29 15:07:54,811 Stage-1 map = 0%,  reduce = 0%
23/07/29 15:08:05 INFO exec.Task: 2023-07-29 15:08:05,691 Stage-1 map = 100%,  reduce = 0%
23/07/29 15:08:16 INFO exec.Task: 2023-07-29 15:08:16,479 Stage-1 map = 100%,  reduce = 100%
```

```
23/07/29 15:08:29 INFO exec.Task: 2023-07-29 15:08:29,898 Stage-2 map = 0%,  reduce = 0%
23/07/29 15:08:39 INFO exec.Task: 2023-07-29 15:08:39,850 Stage-2 map = 100%,  reduce = 0%
23/07/29 15:08:49 INFO exec.Task: 2023-07-29 15:08:49,633 Stage-2 map = 100%,  reduce = 100%
23/07/29 15:08:50 INFO exec.Task: Ended Job = job_1690633086408_0010
23/07/29 15:08:50 INFO ql.Driver: MapReduce Jobs Launched:
23/07/29 15:08:50 INFO ql.Driver: Stage-Stage-1: Map: 1  Reduce: 1   HDFS Read: 1073152 HDFS Write: 225 SUCCESS
23/07/29 15:08:50 INFO ql.Driver: Stage-Stage-2: Map: 1  Reduce: 1   HDFS Read: 4896 HDFS Write: 49 SUCCESS
23/07/29 15:08:50 INFO ql.Driver: Total MapReduce CPU Time Spent: 0 msec
23/07/29 15:08:50 INFO ql.Driver: Completed executing command(queryId=hive_20230729150707_2ae0247f-5a9d-4e55-93c6-b81ff584b97d); Ti
me taken: 69.305 seconds
23/07/29 15:08:50 INFO ql.Driver: OK
```

Output:-

(Observation : There are 218 invalid entries/records in the given dataset).

status	status_count
200	11330
302	3498
304	658
404	251
NULL	218
206	52

Question 2: Create a Hadoop multi-node cluster. (Solution is provided by using Docker).

- ➔ There are various methods via which one can easily install a hadoop single node or multi node cluster.
- ➔ One of the ways is creating datanodes and namenode separately and installing hadoop on every single of these nodes.
- ➔ And, may also need to set up other components like ResourceManager and NodeManager (for YARN, the resource management layer of Hadoop), and various auxiliary services like DataNode services, Secondary NameNode (if used), etc.
- ➔ Setup the environment by customizing a few params in configuration files like below:-

Hdfs-site.xml: This configuration file contains properties specific to HDFS, which is the distributed file system used by Hadoop. Some of the important properties that can be configured in hdfs-site.xml include:

- dfs.replication: The default replication factor for files in HDFS. It determines how many copies of each data block should be stored in the cluster.
- dfs.namenode.name.dir: The directory where the NameNode stores its metadata (e.g., file system namespace, block information).
- dfs.datanode.data.dir: The directory where DataNodes store the actual data blocks.
- dfs.permissions.enabled: Whether to enable HDFS permissions for files and directories.
- dfs.blocksize: The default size of HDFS data blocks.

Core-site.xml: This configuration file contains properties that are common across various Hadoop components.Some of the key properties that can be set in core-site.xml are:

- fs.defaultFS: The default file system URI. It specifies the default file system used by Hadoop, which can be HDFS or other supported file systems like Local File System (file://), Amazon S3 (s3://), etc.
- hadoop.tmp.dir: The base directory for storing temporary files used by Hadoop.
- io.file.buffer.size: The buffer size for file I/O operations in Hadoop.
- mapred-site.xml:
- This configuration file is used to specify properties related to the Hadoop MapReduce framework. Some important properties include:
- mapreduce.framework.name: Specifies the MapReduce framework to use. For YARN, this should be set to "yarn".
- mapreduce.jobtracker.address: The address of the JobTracker (only applicable in older Hadoop versions).
- mapreduce.jobhistory.address: The address of the JobHistory server.

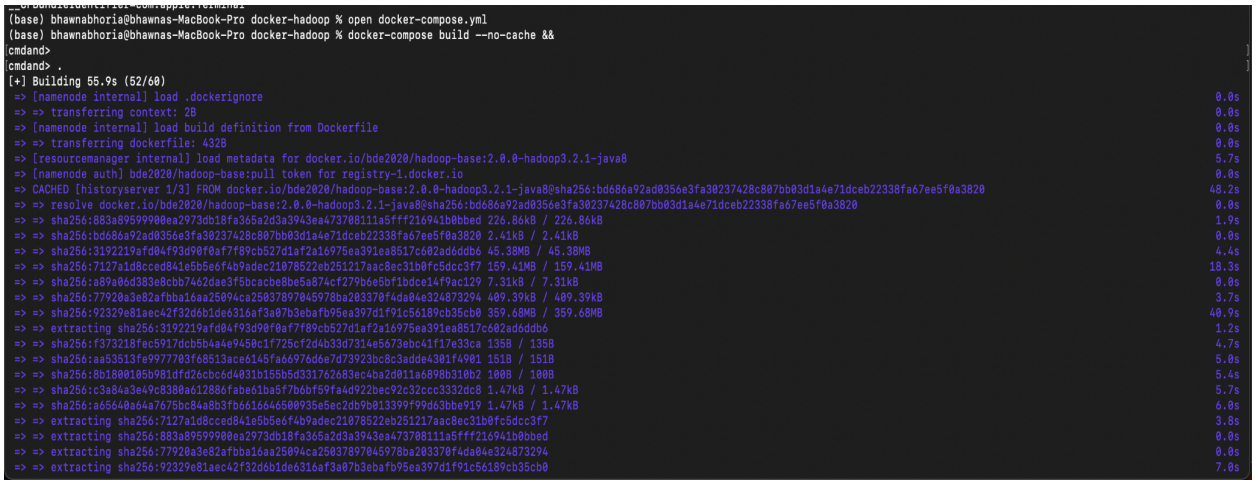
Another way is by using a docker image which has a multi node architecture and customize the yml file and installing the image using the yml file. **-> This method is followed in the presented solution. Steps** :-

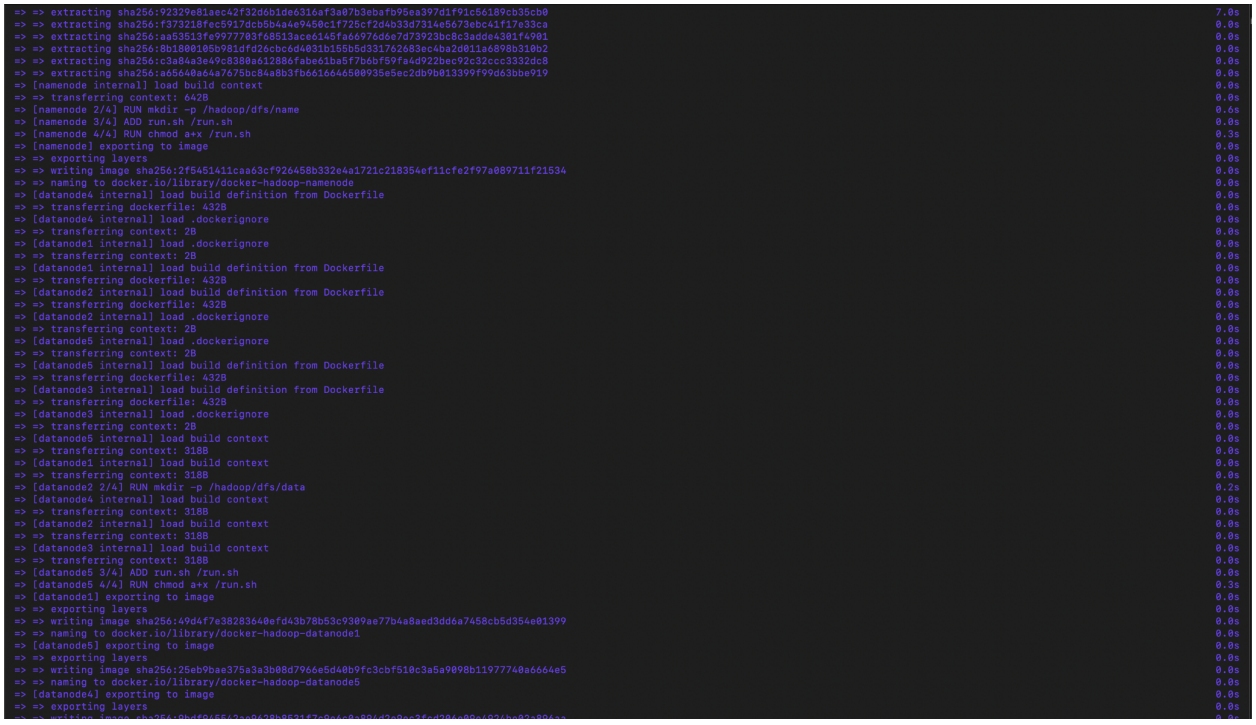
1. Downloading and verifying the Docker installation. Use the command ‘docker ps’.
2. Use the git clone command to get the image or manually download it.
   ```
   git clone git@github.com:big-data-europe/docker-hadoop.git
   ```
3. Open docker-compose.yml file and edit it.
   ```
   open docker-compose.yml
   ```

4. Use the Docker compose command below to read the docker-compose.yml file and set up all components in our Hadoop cluster as individual containers.

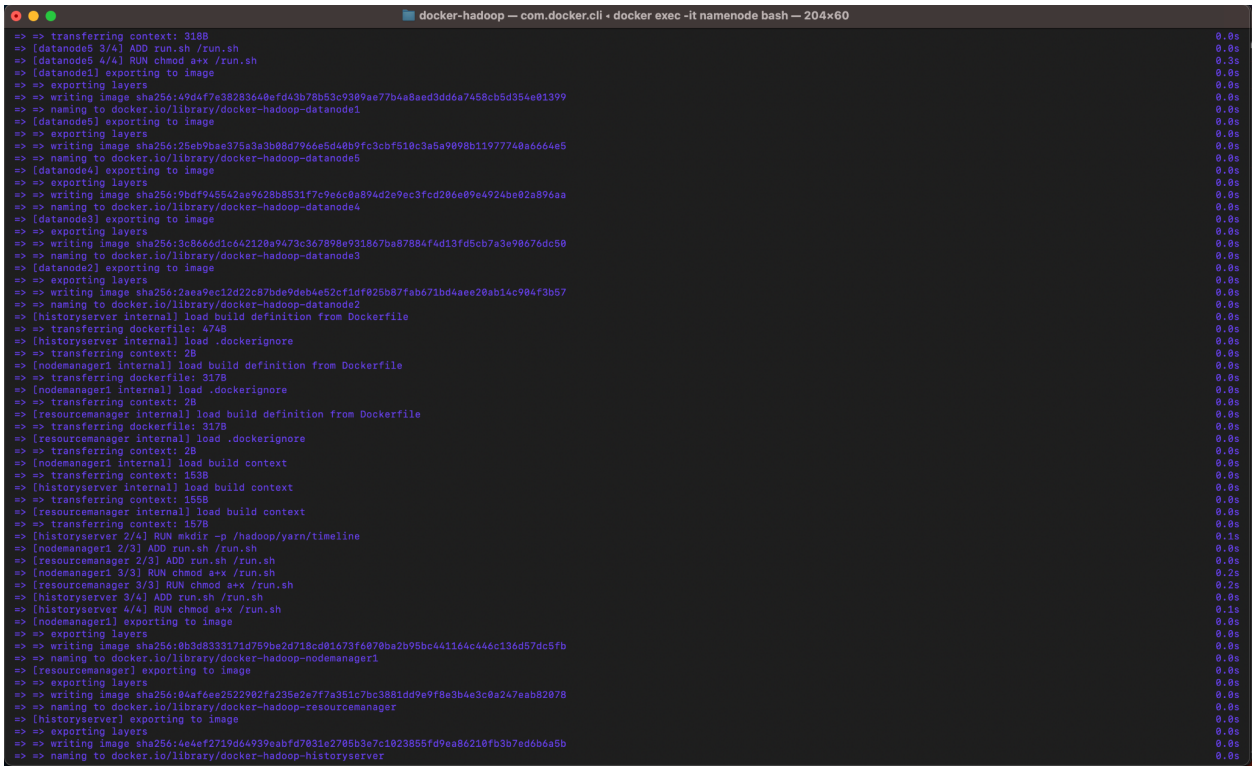
```
docker-compose build --no-cache &&
docker-compose up -d --force-recreate
```

5. Screenshots for yml execution and containers installation.

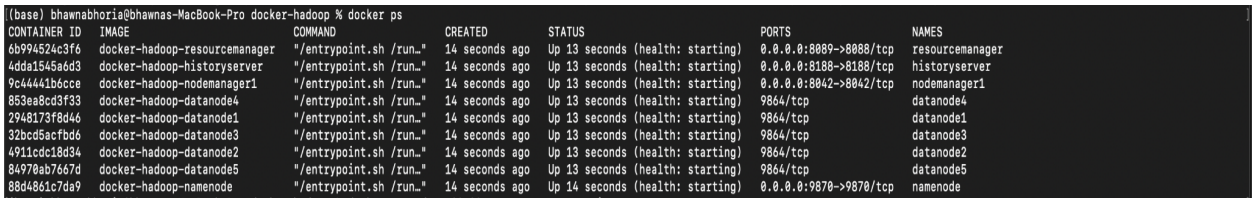




5. Screenshots for yml execution and containers installation.

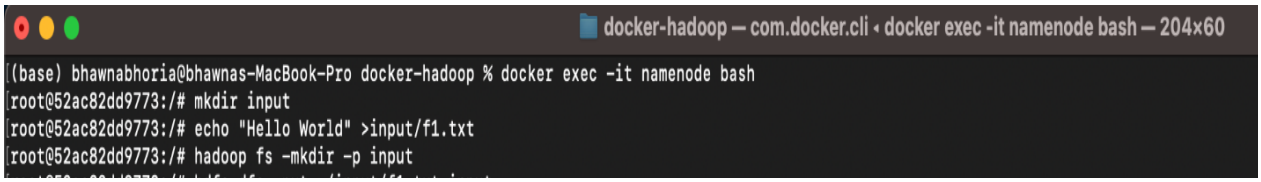


6. We can see multiple containers in the screenshot below. Here, we can see a nodemanager, resourcemanager, historyserver, 5 different datanodes, and a namenode.



7. Now, taking an example on accessing HDFS. We'll create an input directory on HDFS and put a file into it by using namenode container.

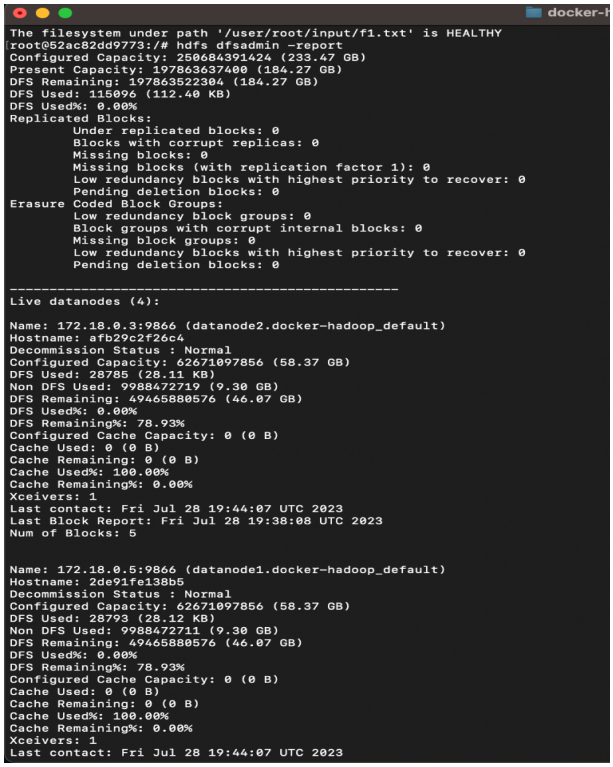
8. First, entering into the namenode container and making the local directory “input” and adding a file say “f1.txt”.



9. Secondly, putting the text file on HDFS and verifying the presence.

```
root@52ac82dd9773:/# hdfs dfs -put ./input/f1.txt input
2023-07-28 19:41:51,006 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
root@52ac82dd9773:/# hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - root supergroup          0 2023-07-28 19:38 /rmstate
drwxr-xr-x   - root supergroup          0 2023-07-28 19:41 /user
root@52ac82dd9773:/# hdfs dfs -ls /user
Found 1 items
drwxr-xr-x   - root supergroup          0 2023-07-28 19:41 /user/root
root@52ac82dd9773:/# hdfs dfs -ls ./input
Found 1 items
-rw-r--r--   3 root supergroup         12 2023-07-28 19:41 input/f1.txt
```

10. Lastly, below is another screenshot to display the cluster details using **dfsadmin report** command.

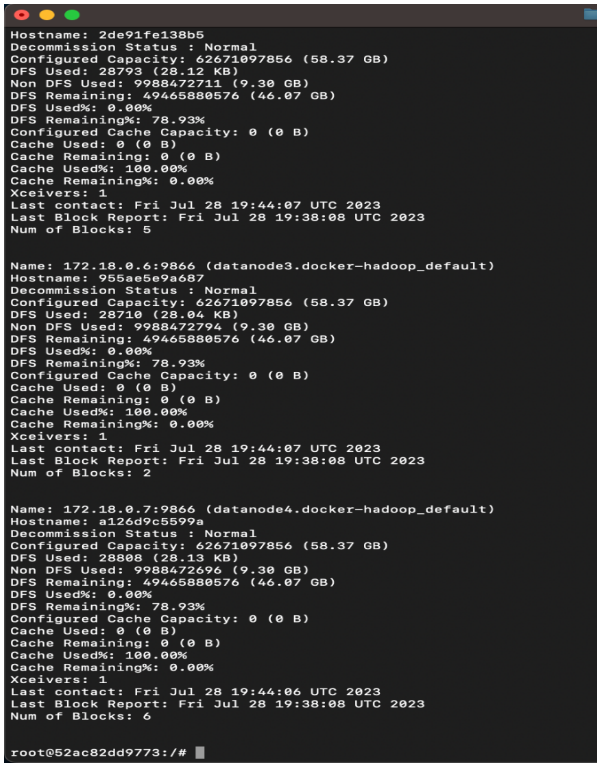


```
The filesystem under path '/user/root/input/f1.txt' is HEALTHY
root@52ac82dd9773:/# hdfs dfsadmin -report
Configured Capacity: 250684391424 (233.47 GB)
Present Capacity: 197863637400 (184.27 GB)
DFS Remaining: 197863522304 (184.27 GB)
DFS Used: 115096 (112.40 KB)
DFS Used%: 0.00%
Replicated Blocks:
        Under replicated blocks: 0
        Blocks with corrupt replicas: 0
        Missing blocks: 0
        Missing blocks (with replication factor 1): 0
        Low redundancy blocks with highest priority to recover: 0
        Pending deletion blocks: 0
Erasure Coded Block Groups:
        Low redundancy block groups: 0
        Block groups with corrupt internal blocks: 0
        Missing block groups: 0
        Low redundancy blocks with highest priority to recover: 0
        Pending deletion blocks: 0

-------------------------------------------------
Live datanodes (4):

Name: 172.18.0.3:9866 (datanode2.docker-hadoop_default)
Hostname: afb29c2f26c4
Decommission Status : Normal
Configured Capacity: 62671097856 (58.37 GB)
DFS Used: 28785 (28.11 KB)
Non DFS Used: 9988472719 (9.30 GB)
DFS Remaining: 49465880576 (46.07 GB)
DFS Used%: 0.00%
DFS Remaining%: 78.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Jul 28 19:44:07 UTC 2023
Last Block Report: Fri Jul 28 19:38:08 UTC 2023
Num of Blocks: 5


Name: 172.18.0.5:9866 (datanode1.docker-hadoop_default)
Hostname: 2de91fe138b5
Decommission Status : Normal
Configured Capacity: 62671097856 (58.37 GB)
DFS Used: 28793 (28.12 KB)
Non DFS Used: 9988472711 (9.30 GB)
DFS Remaining: 49465880576 (46.07 GB)
DFS Used%: 0.00%
DFS Remaining%: 78.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Jul 28 19:44:07 UTC 2023
```



```
Hostname: 2de91fe138b5
Decommission Status : Normal
Configured Capacity: 62671097856 (58.37 GB)
DFS Used: 28793 (28.12 KB)
Non DFS Used: 9988472711 (9.30 GB)
DFS Remaining: 49465880576 (46.07 GB)
DFS Used%: 0.00%
DFS Remaining%: 78.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Jul 28 19:44:07 UTC 2023
Last Block Report: Fri Jul 28 19:38:08 UTC 2023
Num of Blocks: 5


Name: 172.18.0.6:9866 (datanode3.docker-hadoop_default)
Hostname: 955ae5e9a687
Decommission Status : Normal
Configured Capacity: 62671097856 (58.37 GB)
DFS Used: 28710 (28.04 KB)
Non DFS Used: 9988472794 (9.30 GB)
DFS Remaining: 49465880576 (46.07 GB)
DFS Used%: 0.00%
DFS Remaining%: 78.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Jul 28 19:44:07 UTC 2023
Last Block Report: Fri Jul 28 19:38:08 UTC 2023
Num of Blocks: 2


Name: 172.18.0.7:9866 (datanode4.docker-hadoop_default)
Hostname: a126d9c5599a
Decommission Status : Normal
Configured Capacity: 62671097856 (58.37 GB)
DFS Used: 28808 (28.13 KB)
Non DFS Used: 9988472696 (9.30 GB)
DFS Remaining: 49465880576 (46.07 GB)
DFS Used%: 0.00%
DFS Remaining%: 78.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Jul 28 19:44:06 UTC 2023
Last Block Report: Fri Jul 28 19:38:08 UTC 2023
Num of Blocks: 6


root@52ac82dd9773:/#
```