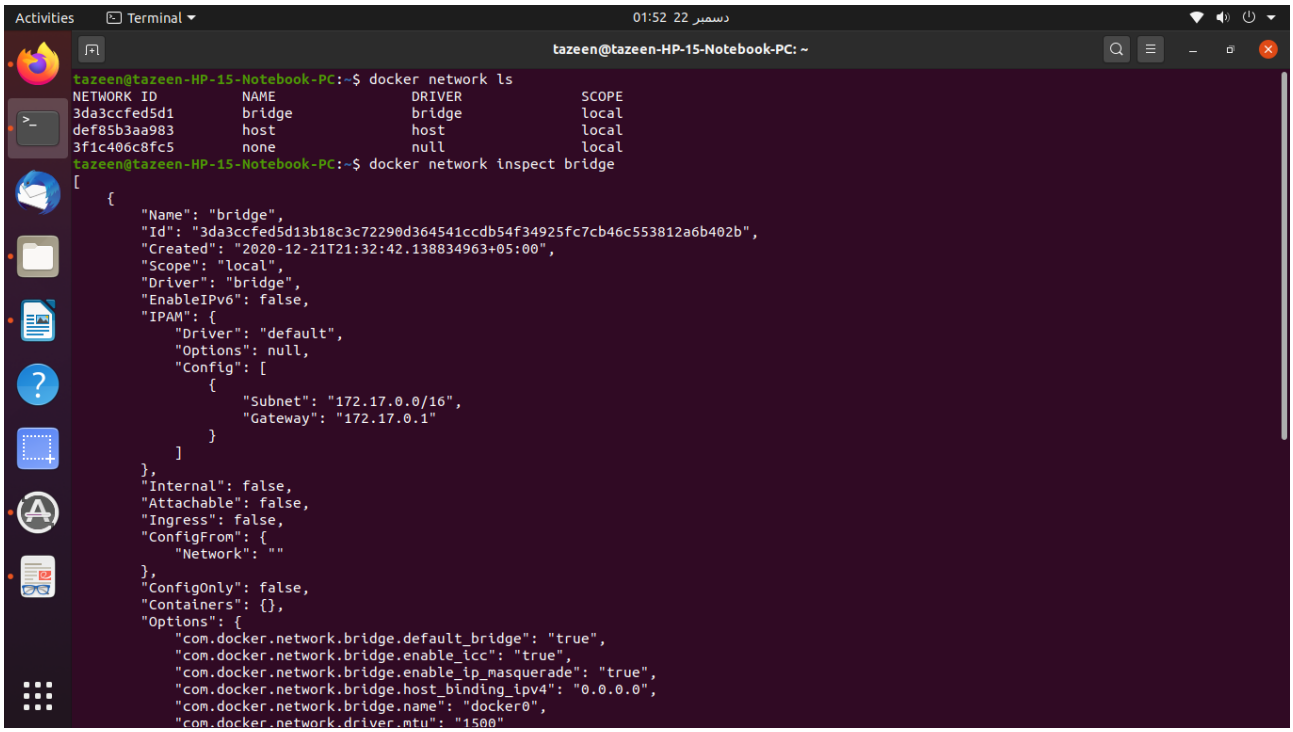
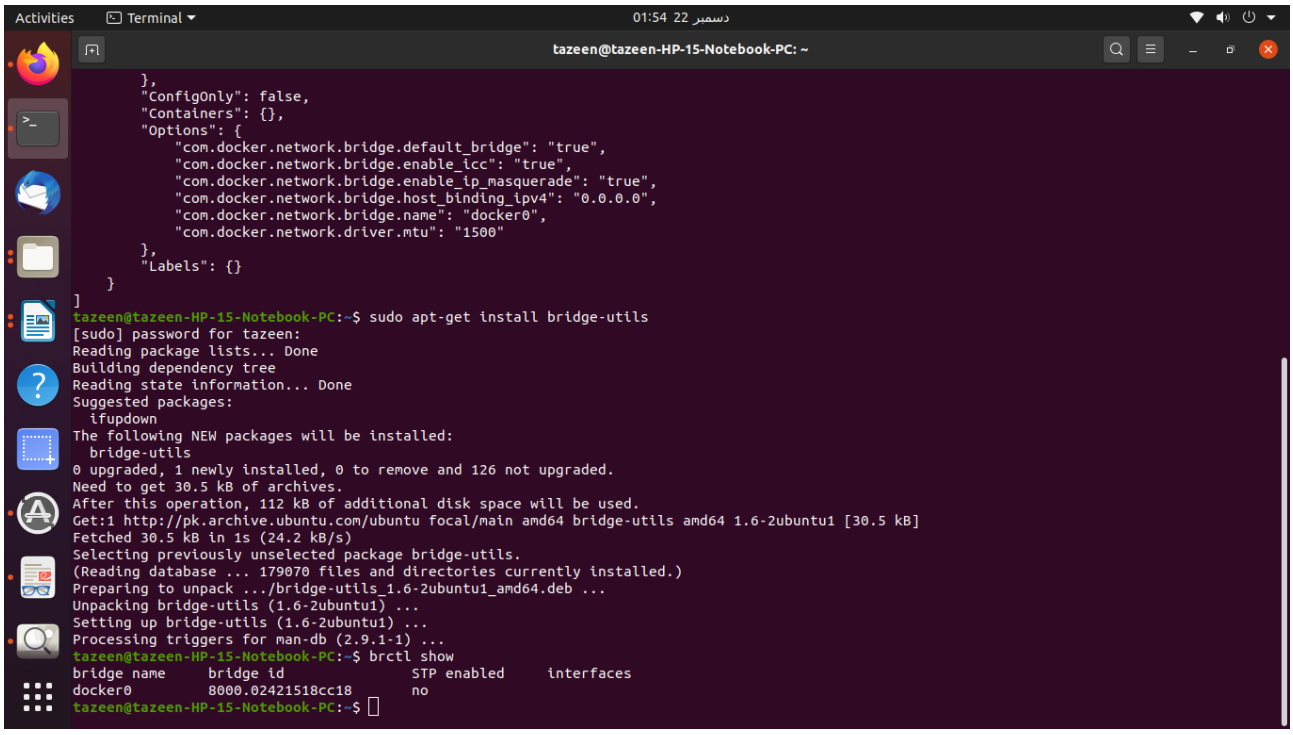
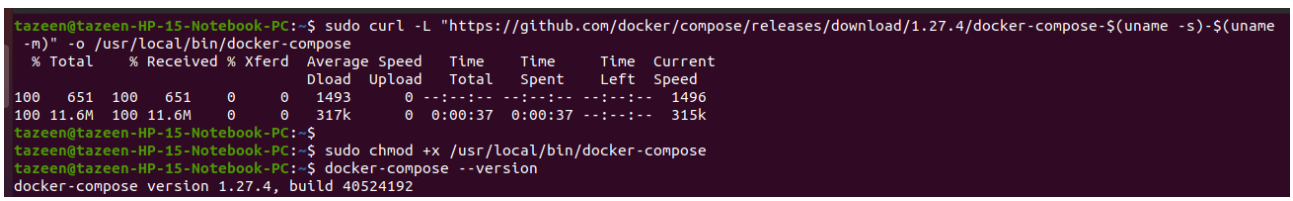
Lambda Architecture:

1- Database (mongodb layer)

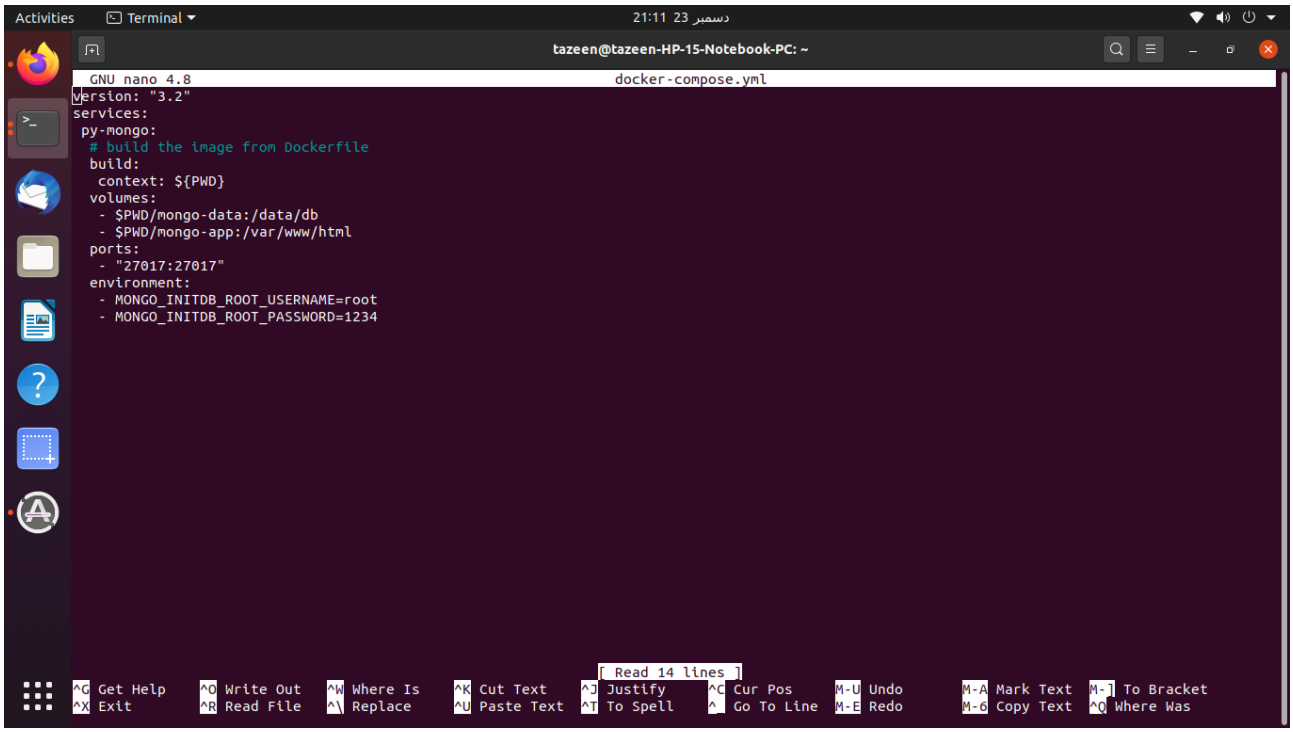
Creating docker-compose.yml file for mongo db container



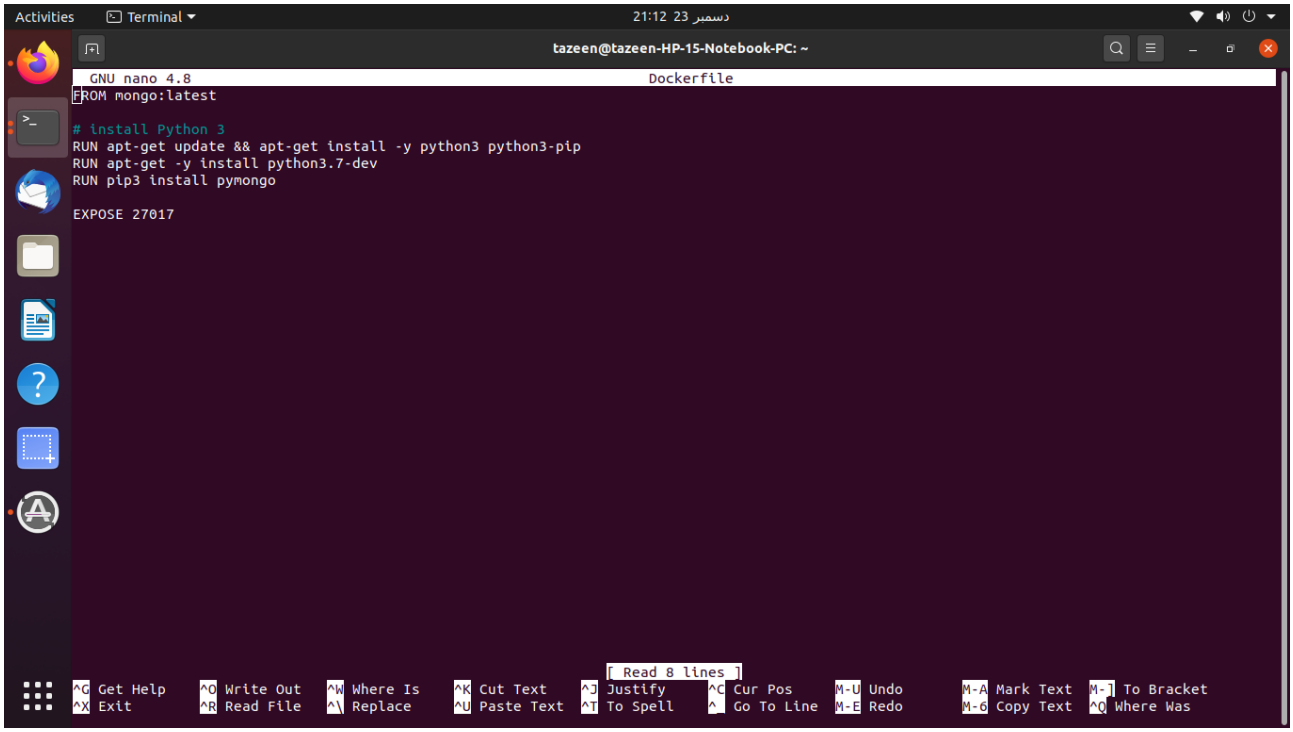


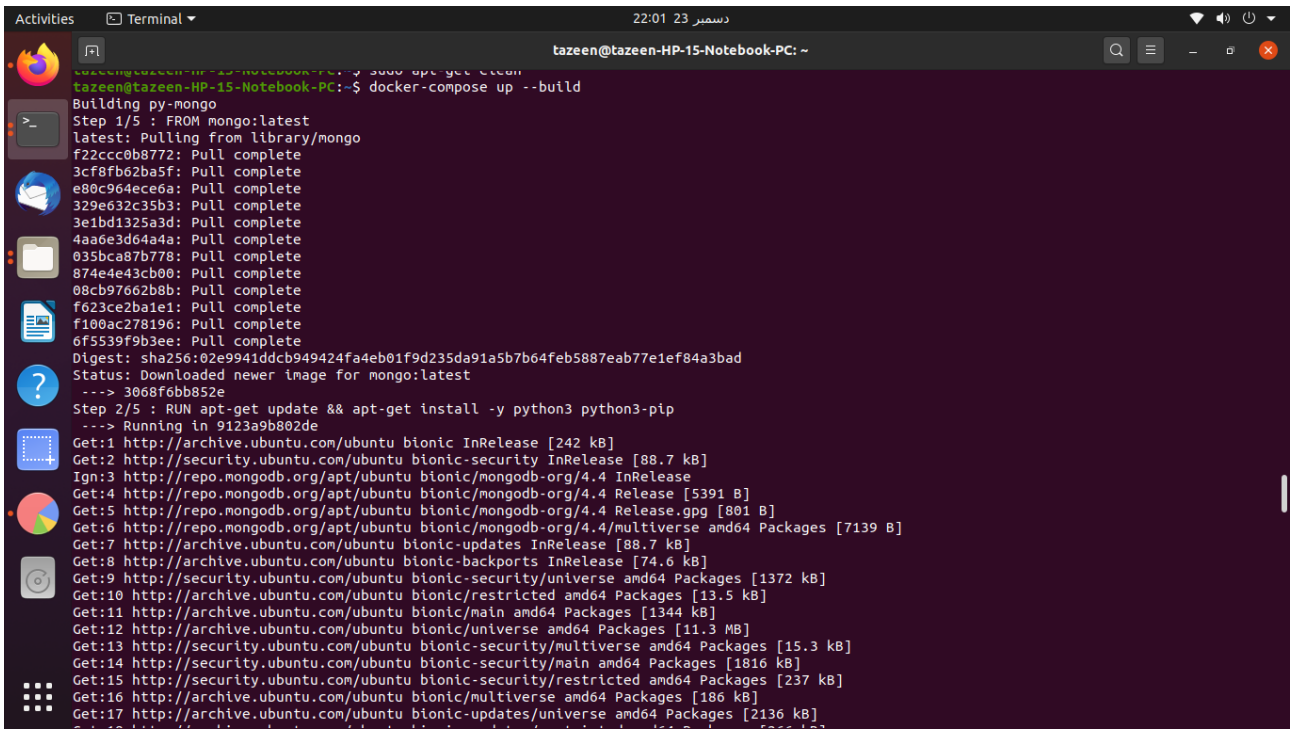


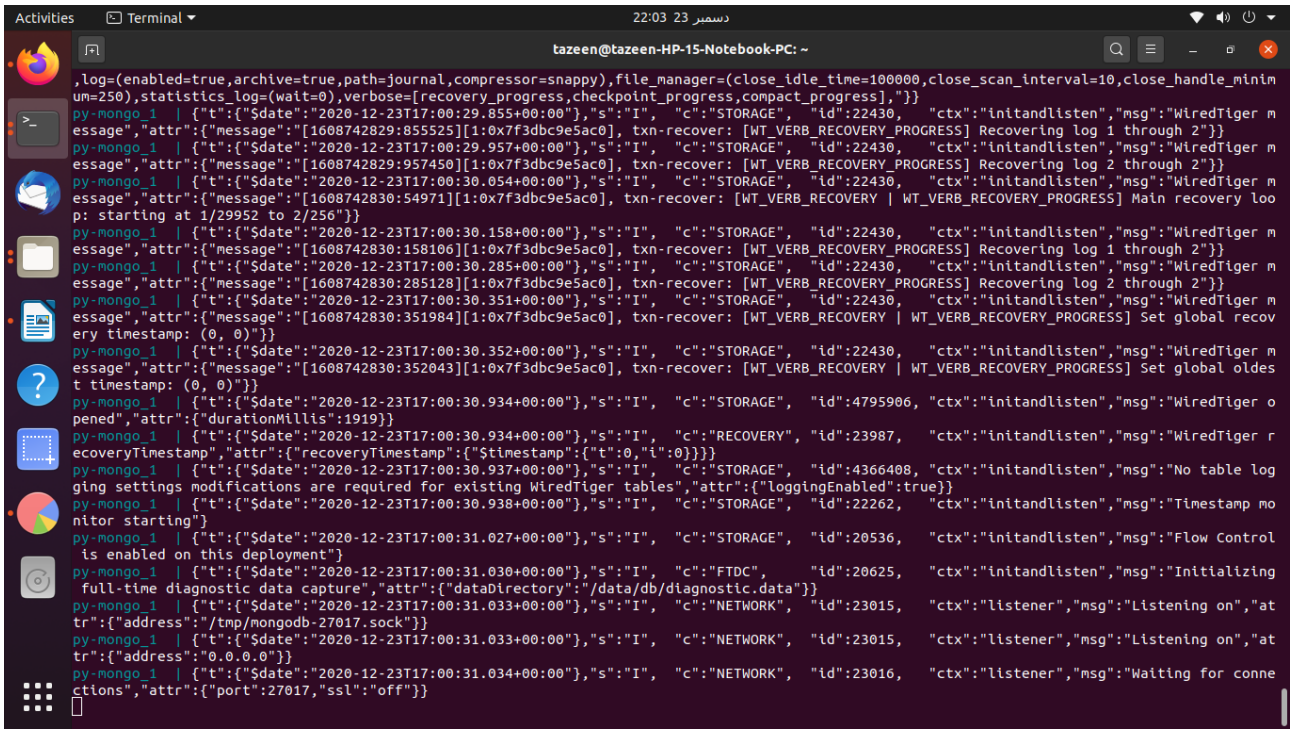
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano docker-compose.yml
[sudo] password for tazeen:
```



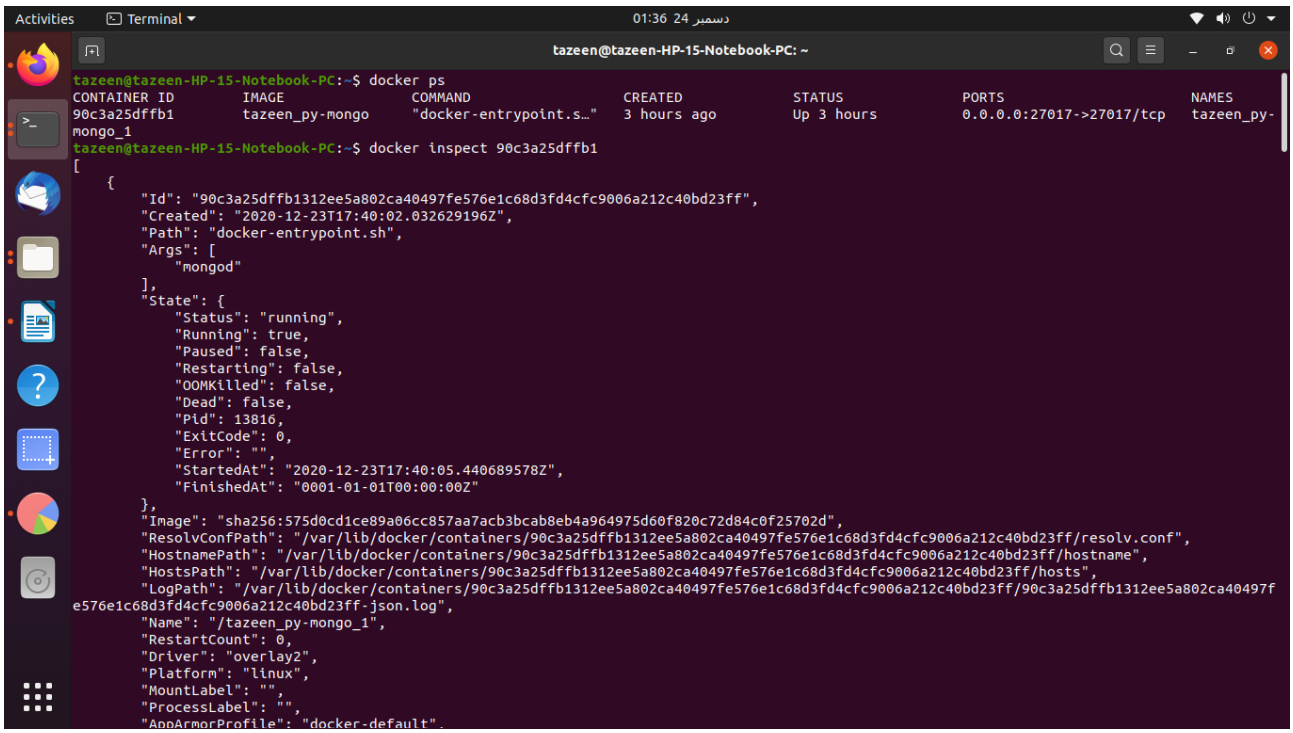

```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```



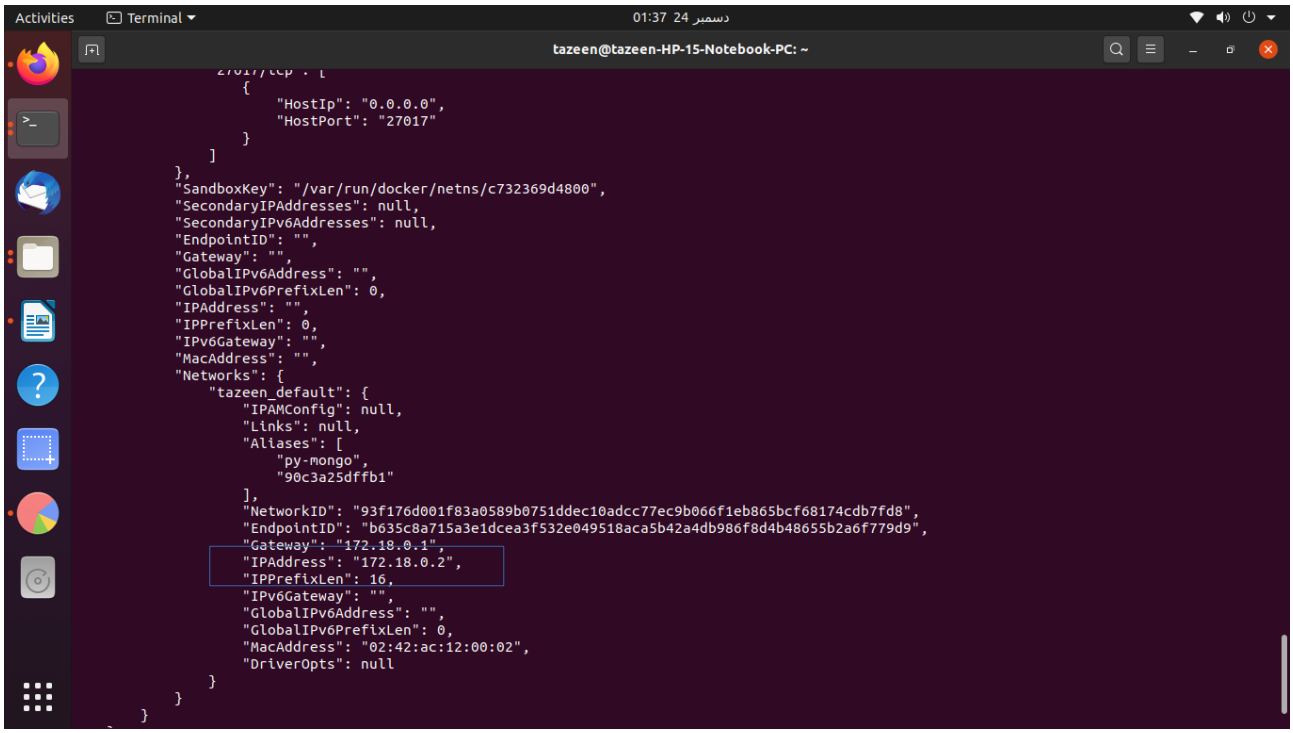




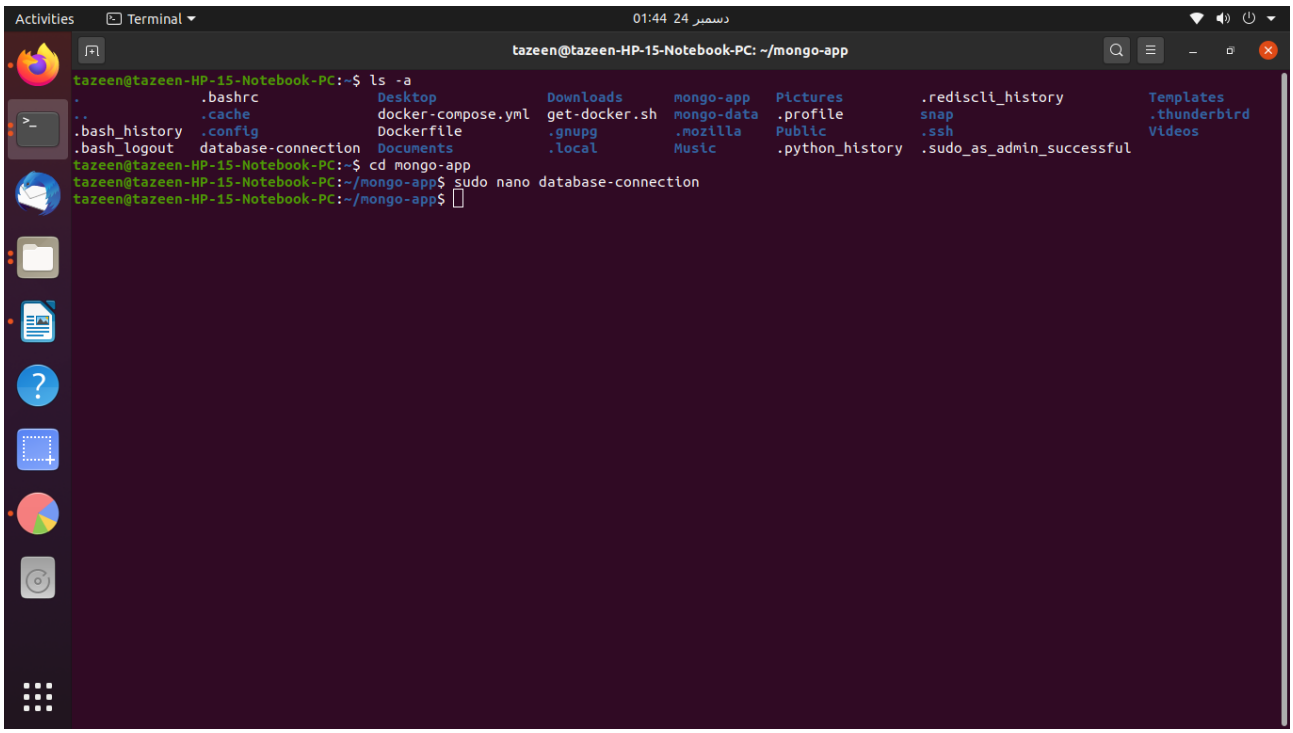
Get the docker container id and inspect it to see the ip address

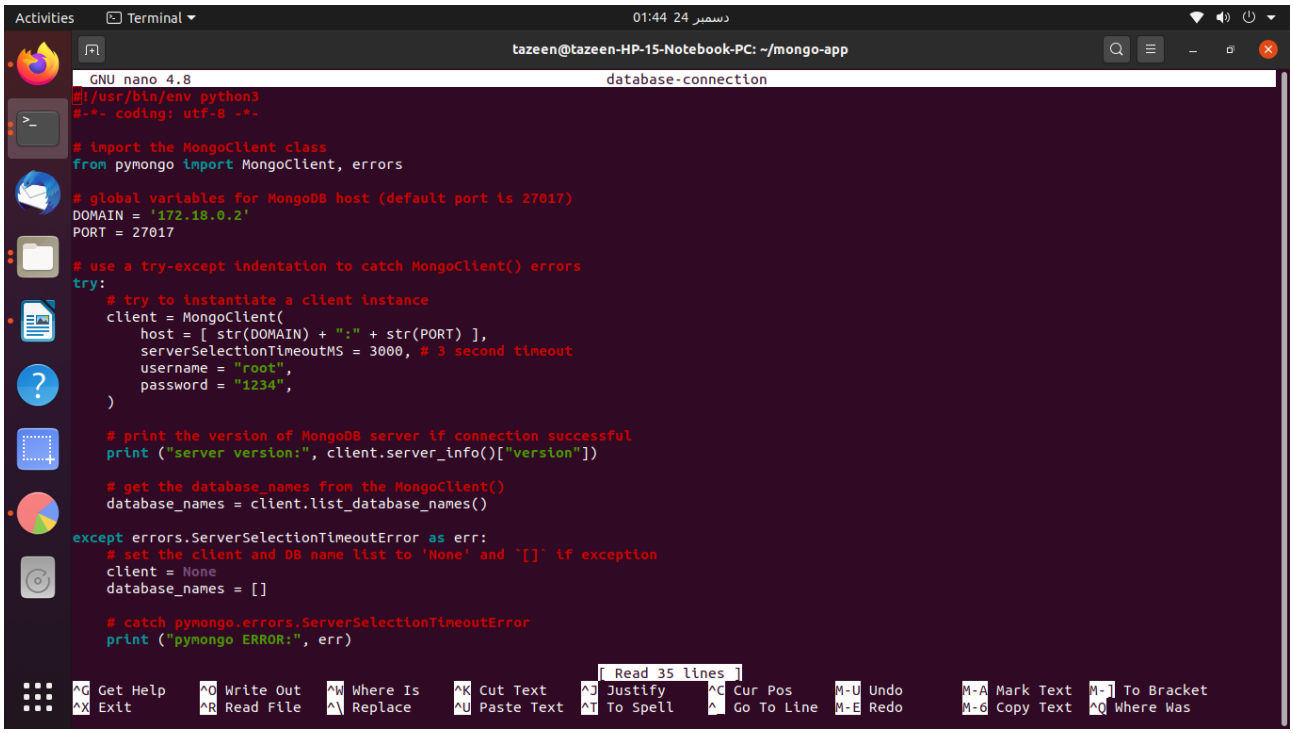



```
tazeen@tazeen-HP-15-Notebook-PC:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
90c3a25dffb1        tazeen_py-mongo     "docker-entrypoint.s…"   3 hours ago         Up 3 hours          0.0.0.0:27017->27017/tcp   tazeen_py-
mongo_1
tazeen@tazeen-HP-15-Notebook-PC:~$ docker inspect 90c3a25dffb1
[
    {
        "Id": "90c3a25dffb1312ee5a802ca40497fe576e1c68d3fd4cfc9006a212c40bd23ff",
        "Created": "2020-12-23T17:40:02.032629196Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mongod"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 13816,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-12-23T17:40:05.4406895787Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:575d0cd1ce89a06cc857aa7acb3bcab8eb4a964975d60f820c72d84c0f25702d",
        "ResolvConfPath": "/var/lib/docker/containers/90c3a25dffb1312ee5a802ca40497fe576e1c68d3fd4cfc9006a212c40bd23ff/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/90c3a25dffb1312ee5a802ca40497fe576e1c68d3fd4cfc9006a212c40bd23ff/hostname",
        "HostsPath": "/var/lib/docker/containers/90c3a25dffb1312ee5a802ca40497fe576e1c68d3fd4cfc9006a212c40bd23ff/hosts",
        "LogPath": "/var/lib/docker/containers/90c3a25dffb1312ee5a802ca40497fe576e1c68d3fd4cfc9006a212c40bd23ff/90c3a25dffb1312ee5a802ca40497f
e576e1c68d3fd4cfc9006a212c40bd23ff-json.log",
        "Name": "/tazeen_py-mongo_1",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
```

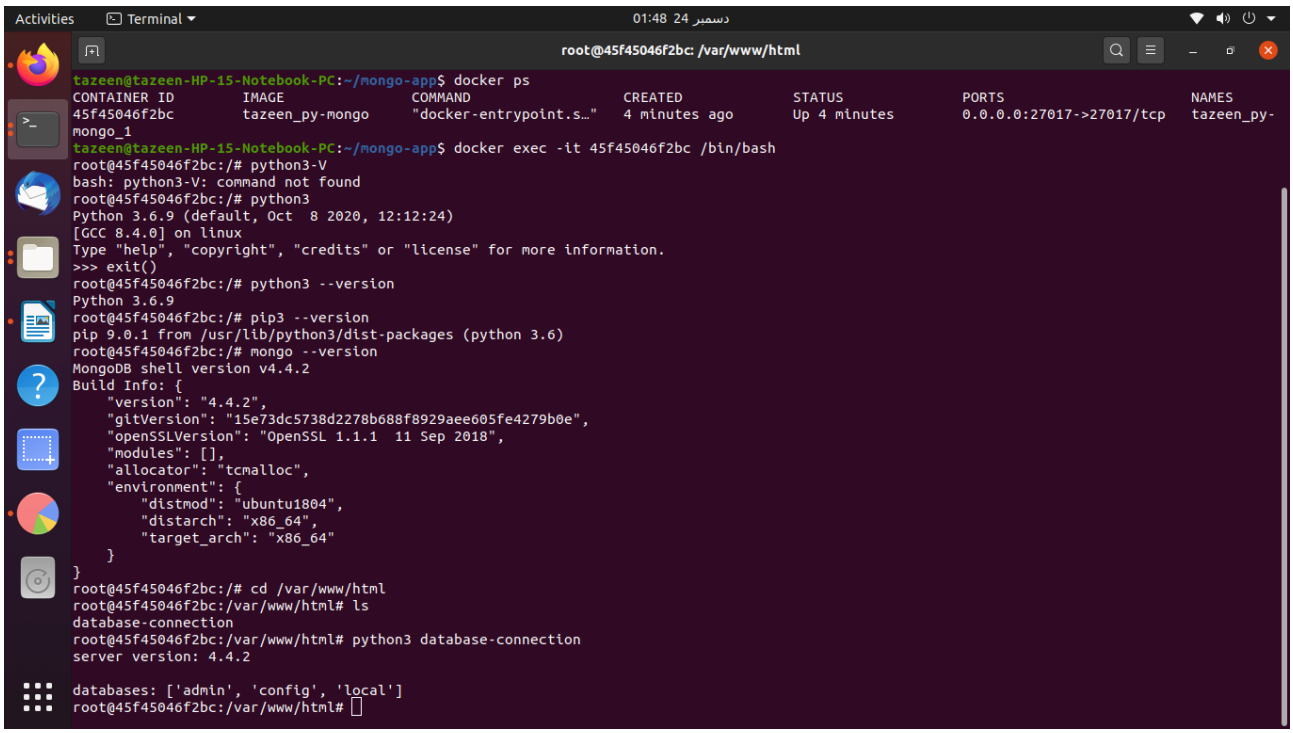



```
                "HostIp": "0.0.0.0",
                "HostPort": "27017"
            }
        ]
    },
    "SandboxKey": "/var/run/docker/netns/c732369d4800",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "",
    "Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
        "tazeen_default": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": [
                "py-mongo",
                "90c3a25dffb1"
            ],
            "NetworkID": "93f176d001f83a0589b0751ddec10adcc77ec9b066f1eb865bcf68174cdb7fd8",
            "EndpointID": "b635c8a715a3e1dcea3f532e049518aca5b42a4db986f8d4b48655b2a6f779d9",
            "Gateway": "172.18.0.1",
            "IPAddress": "172.18.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:12:00:02",
            "DriverOpts": null
        }
    }
}
```

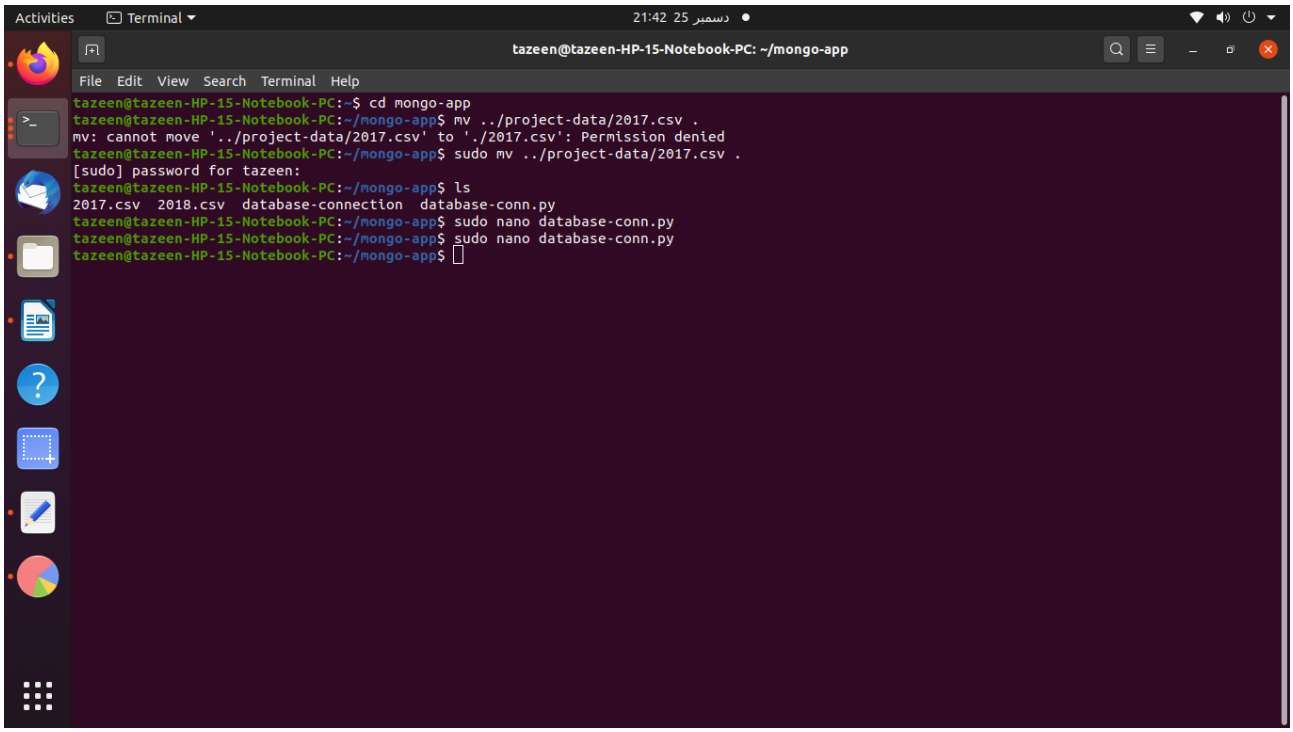




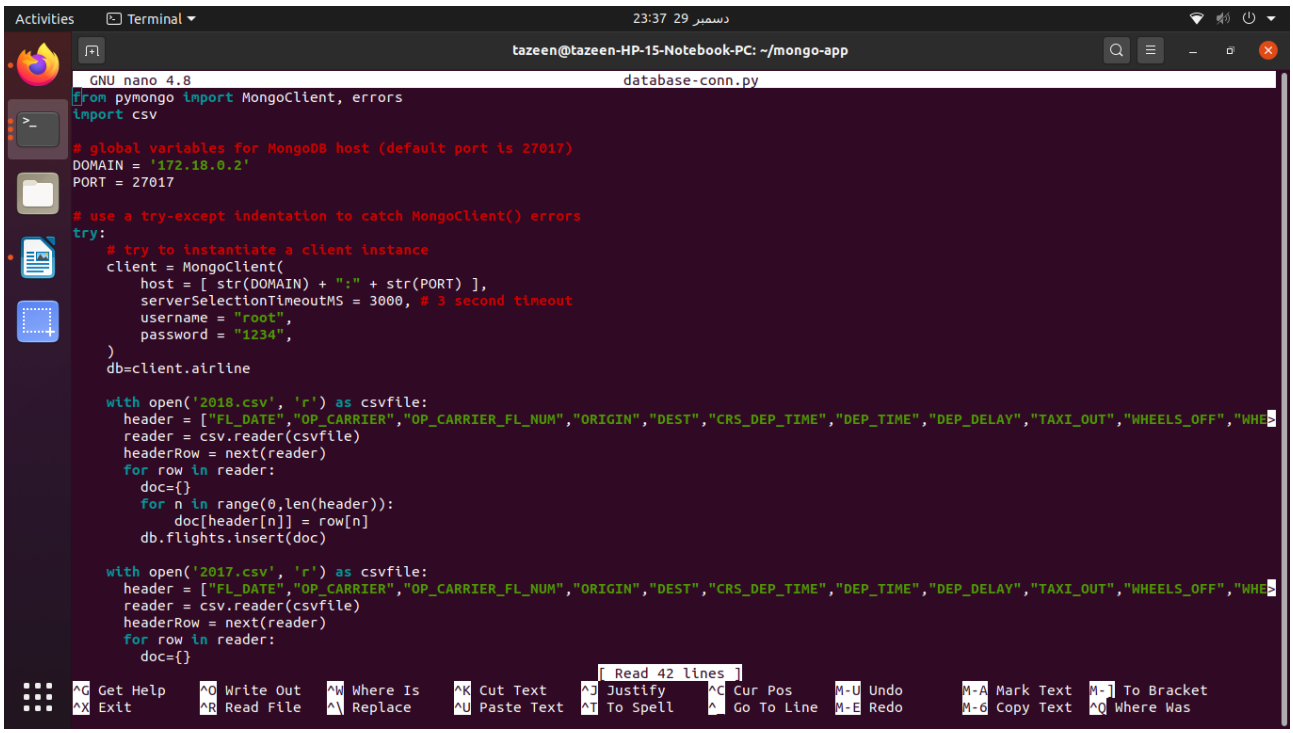
Execute the python script from docker to test the mongodb connection

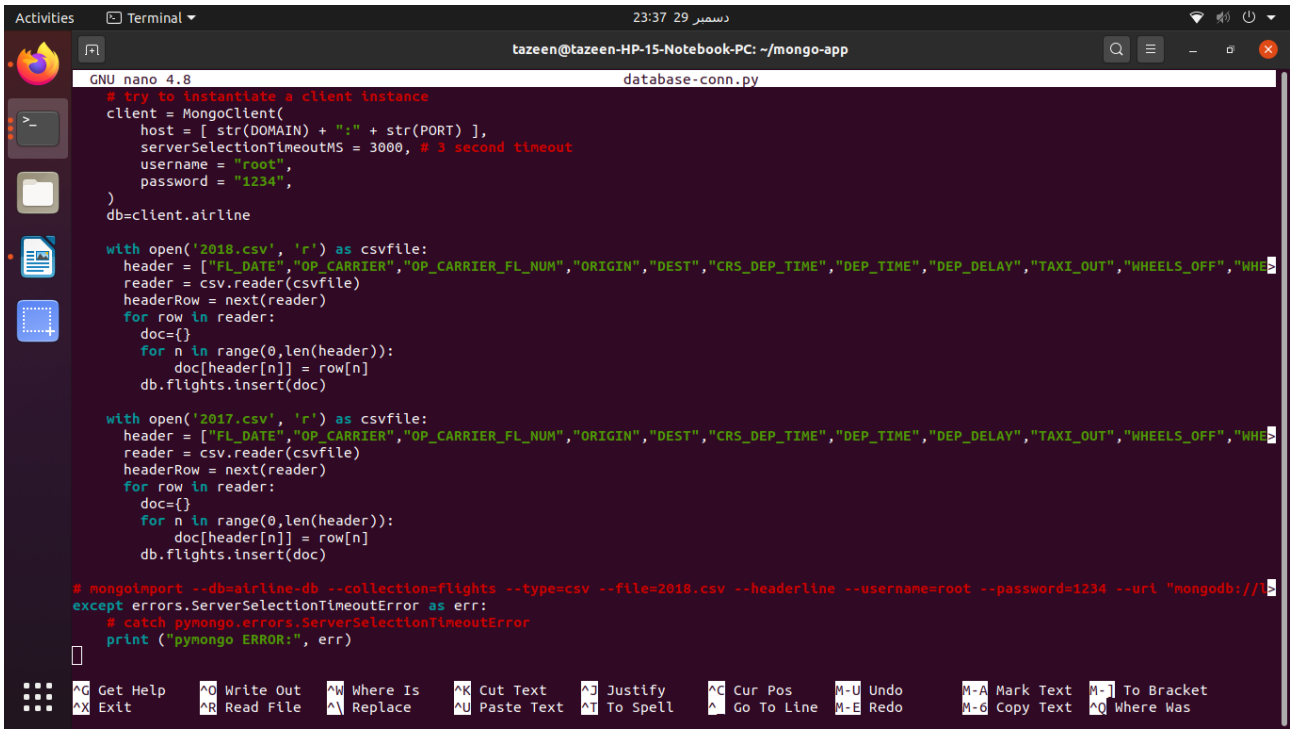


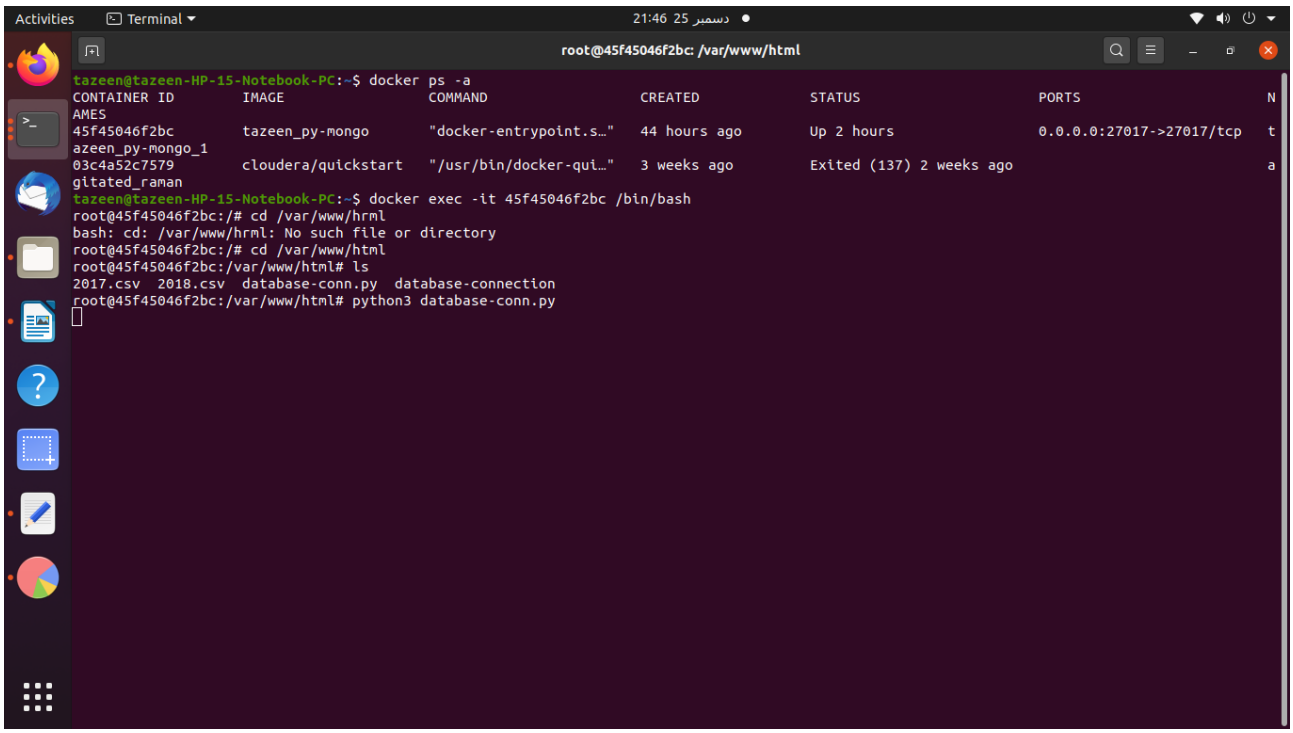
Creating python script to save data in mongodb



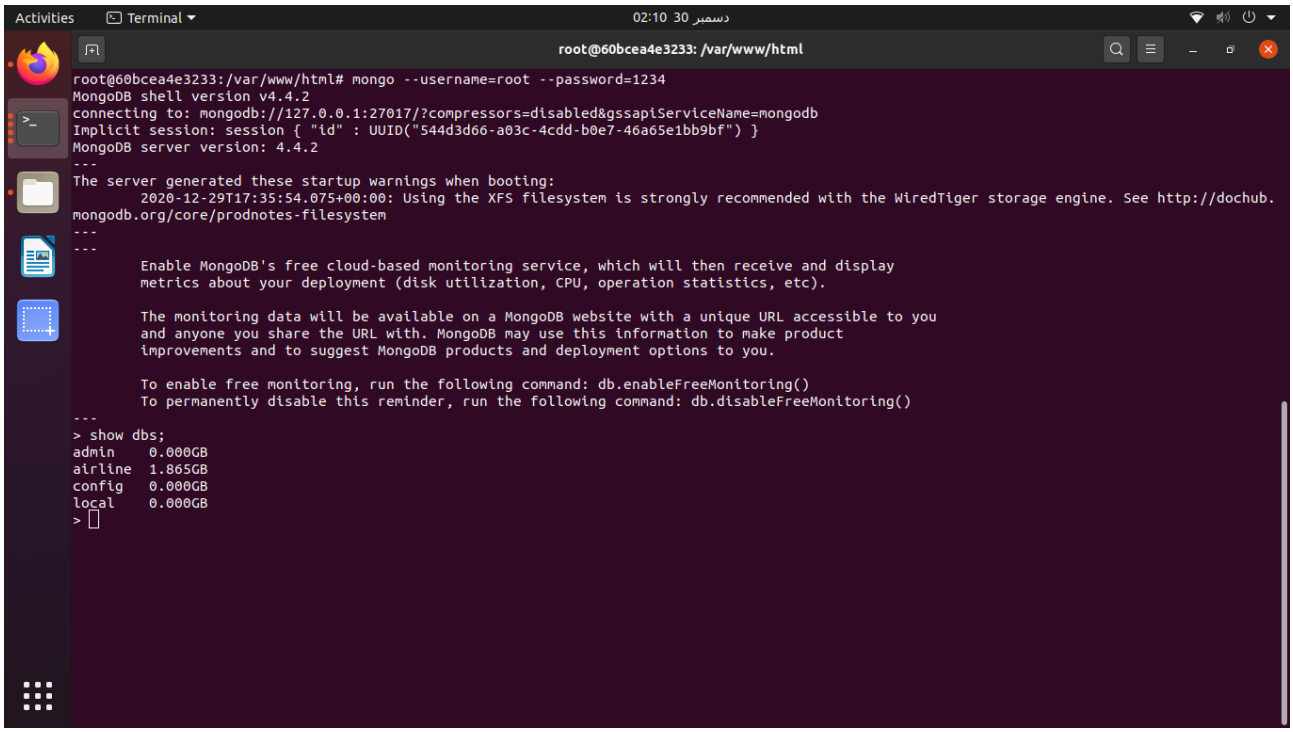
Saving data from 2018.csv and 2017.csv which is placed in mongo-app folder. These csv files are header based so starting with first index and inserting data into database.


Activities
Terminal
23:37 29 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~/mongo-app
GNU nano 4.8
database-conn.py
from pymongo import MongoClient, errors
import csv
# global variables for MongoDB host (default port is 27017)
DOMAIN = '172.18.0.2'
PORT = 27017
# use a try-except indentation to catch MongoClient() errors
try:
# try to instantiate a client instance
client = MongoClient(
host = [str(DOMAIN) + ":" + str(PORT)],
serverSelectionTimeoutMS = 3000, # 3 second timeout
username = "root",
password = "1234",
)
db=client.airline
with open('2018.csv', 'r') as csvfile:
header = ["FL_DATE","OP_CARRIER","OP_CARRIER_FL_NUM","ORIGIN","DEST","CRS_DEP_TIME","DEP_TIME","DEP_DELAY","TAXI_OUT","WHEELS_OFF","WHE
reader = csv.reader(csvfile)
headerRow = next(reader)
for row in reader:
doc={}
for n in range(0,len(header)):
doc[header[n]] = row[n]
db.flights.insert(doc)
with open('2017.csv', 'r') as csvfile:
header = ["FL_DATE","OP_CARRIER","OP_CARRIER_FL_NUM","ORIGIN","DEST","CRS_DEP_TIME","DEP_TIME","DEP_DELAY","TAXI_OUT","WHEELS_OFF","WHE
reader = csv.reader(csvfile)
headerRow = next(reader)
for row in reader:
doc={}
[Read 42 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo M-A Mark Text M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line M-E Redo M-6 Copy Text ^Q Where Was


Activities
Terminal
23:37 29 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~/mongo-app
GNU nano 4.8
database-conn.py
# try to instantiate a client instance
client = MongoClient(
host = [str(DOMAIN) + ":" + str(PORT)],
serverSelectionTimeoutMS = 3000, # 3 second timeout
username = "root",
password = "1234",
)
db=client.airline
with open('2018.csv', 'r') as csvfile:
header = ["FL_DATE","OP_CARRIER","OP_CARRIER_FL_NUM","ORIGIN","DEST","CRS_DEP_TIME","DEP_TIME","DEP_DELAY","TAXI_OUT","WHEELS_OFF","WHE
reader = csv.reader(csvfile)
headerRow = next(reader)
for row in reader:
doc={}
for n in range(0,len(header)):
doc[header[n]] = row[n]
db.flights.insert(doc)
with open('2017.csv', 'r') as csvfile:
header = ["FL_DATE","OP_CARRIER","OP_CARRIER_FL_NUM","ORIGIN","DEST","CRS_DEP_TIME","DEP_TIME","DEP_DELAY","TAXI_OUT","WHEELS_OFF","WHE
reader = csv.reader(csvfile)
headerRow = next(reader)
for row in reader:
doc={}
for n in range(0,len(header)):
doc[header[n]] = row[n]
db.flights.insert(doc)
# mongoimport --db=airline-db --collection=flights --type=csv --file=2018.csv --headerline --username=root --password=1234 --uri "mongodb://l
except errors.ServerSelectionTimeoutError as err:
catch pymongo.errors.ServerSelectionTimeoutError
print ("pymongo ERROR:", err)
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo M-A Mark Text M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line M-E Redo M-6 Copy Text ^Q Where Was

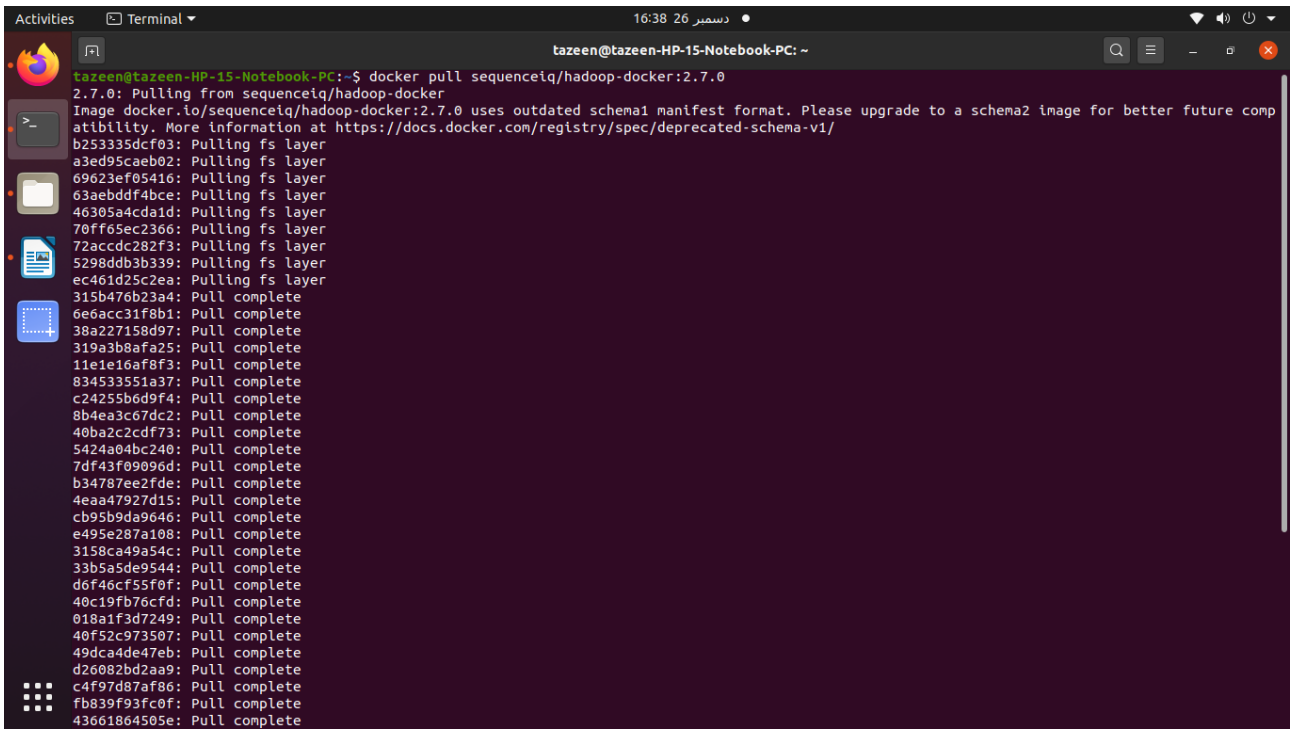


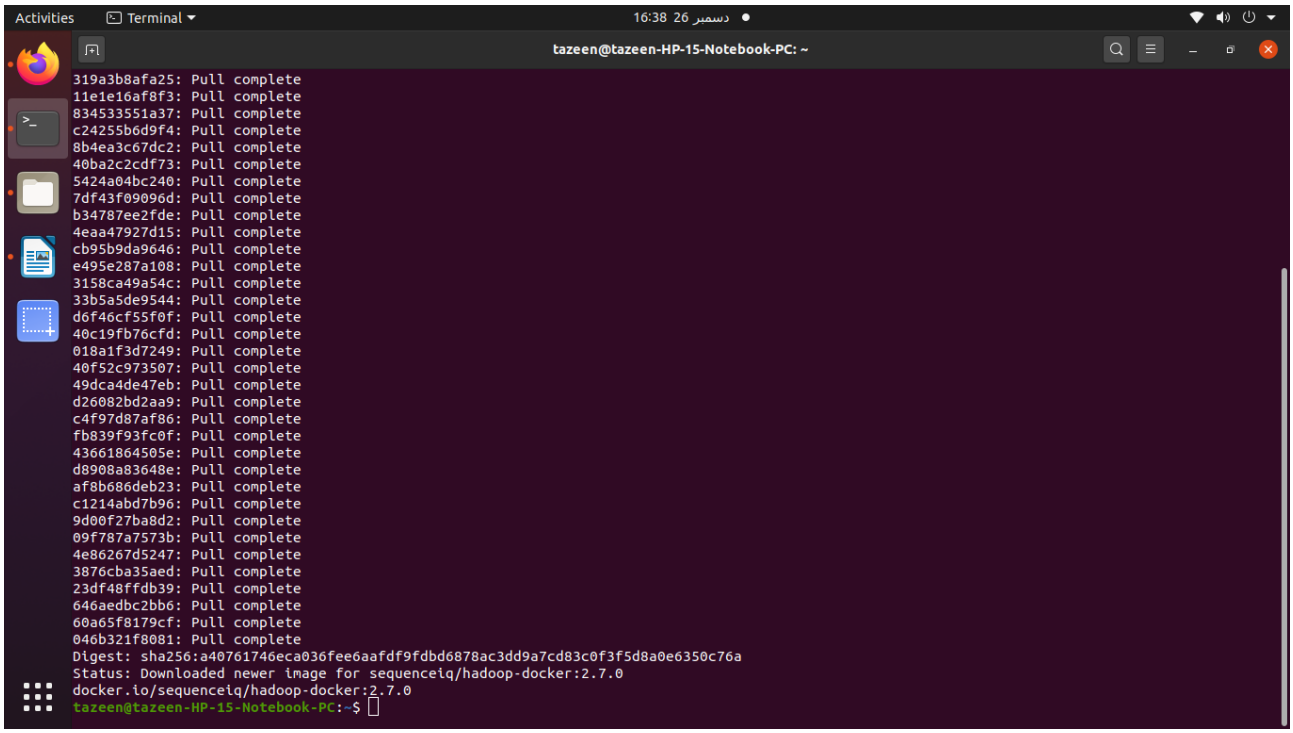
Connecting to mongodb to check that data is inserted in airline database.



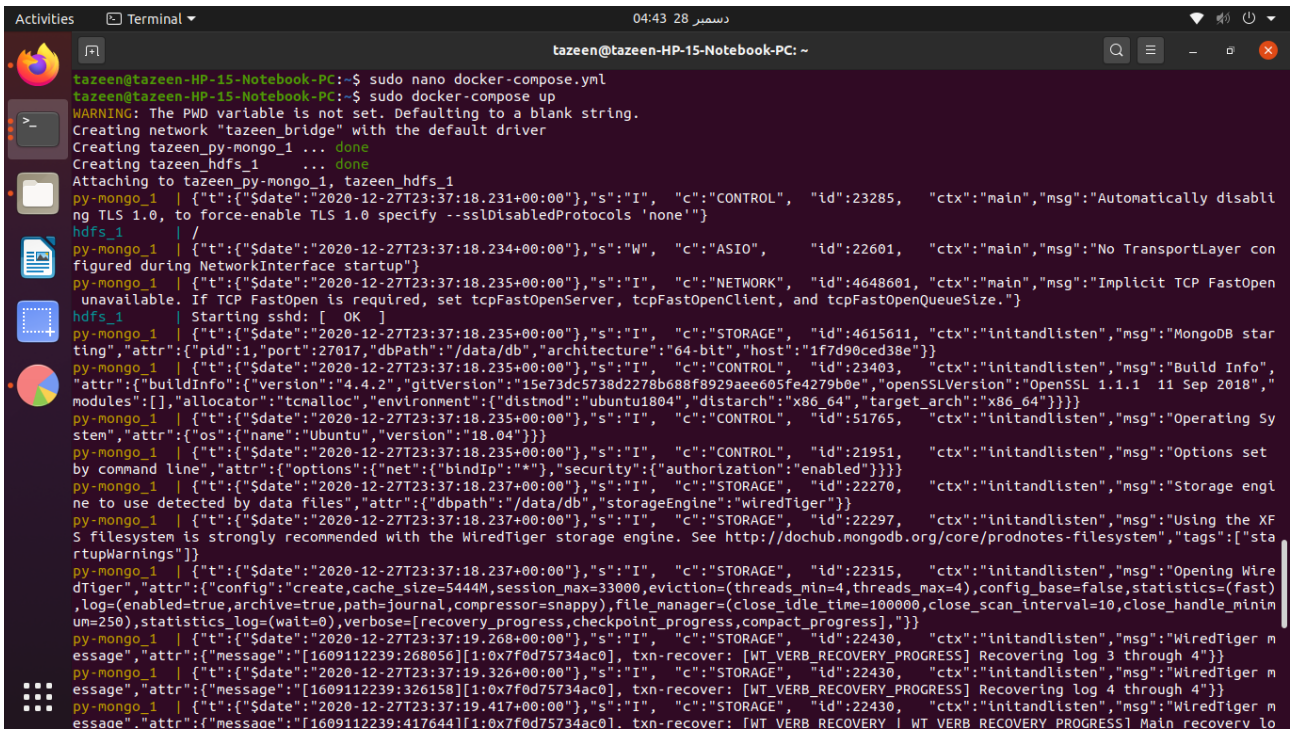
2- Batch processing layer:

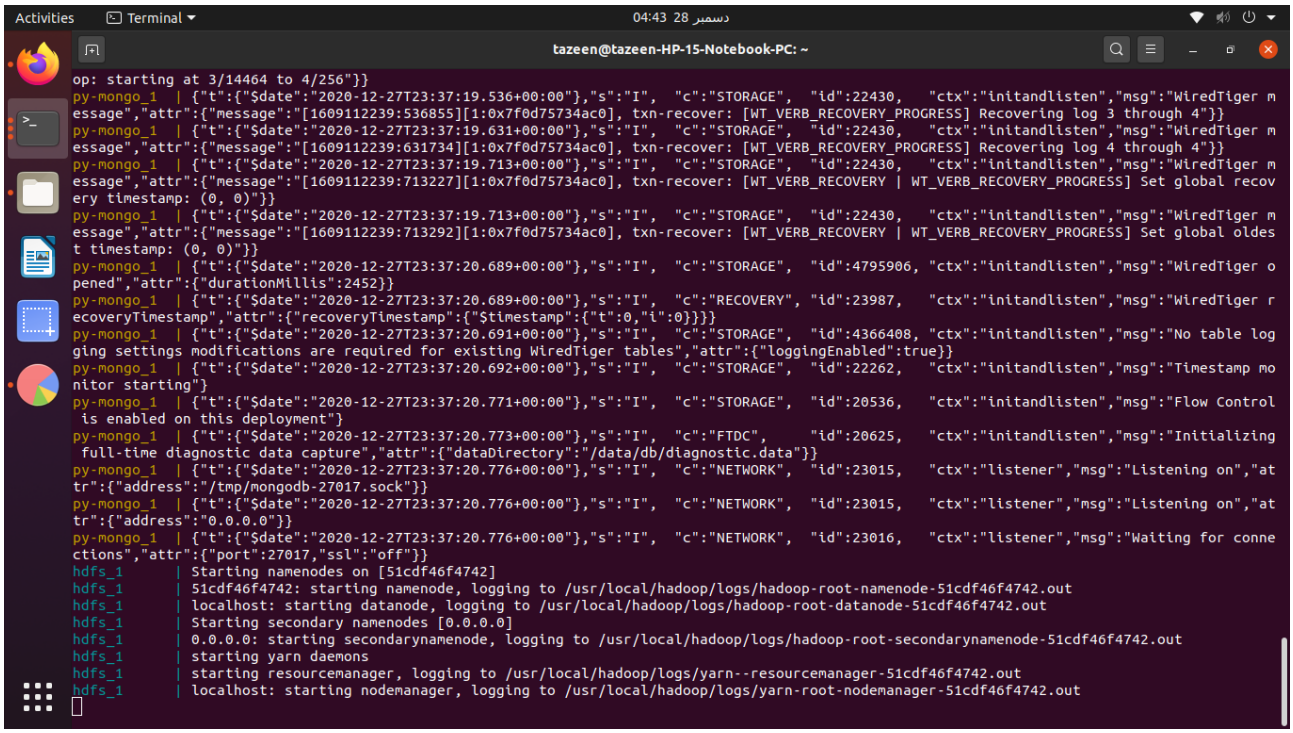
Pulling image for hadoop container

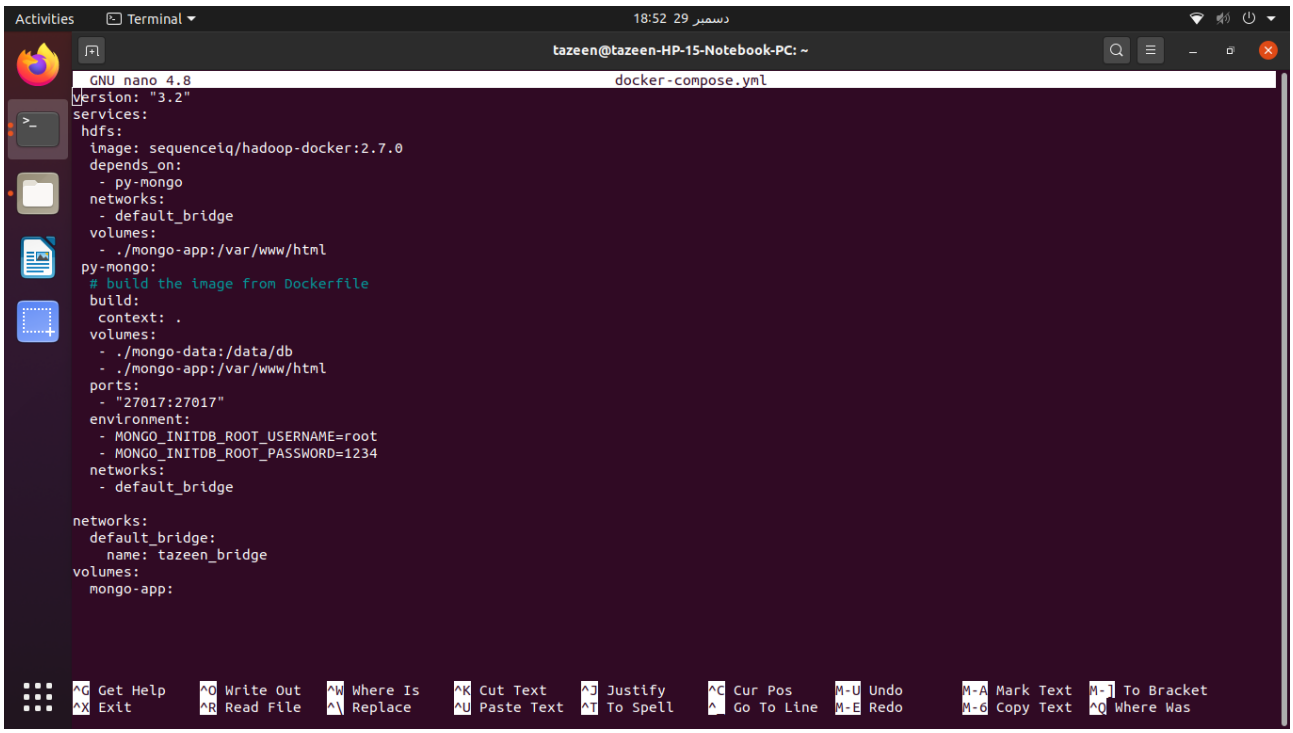




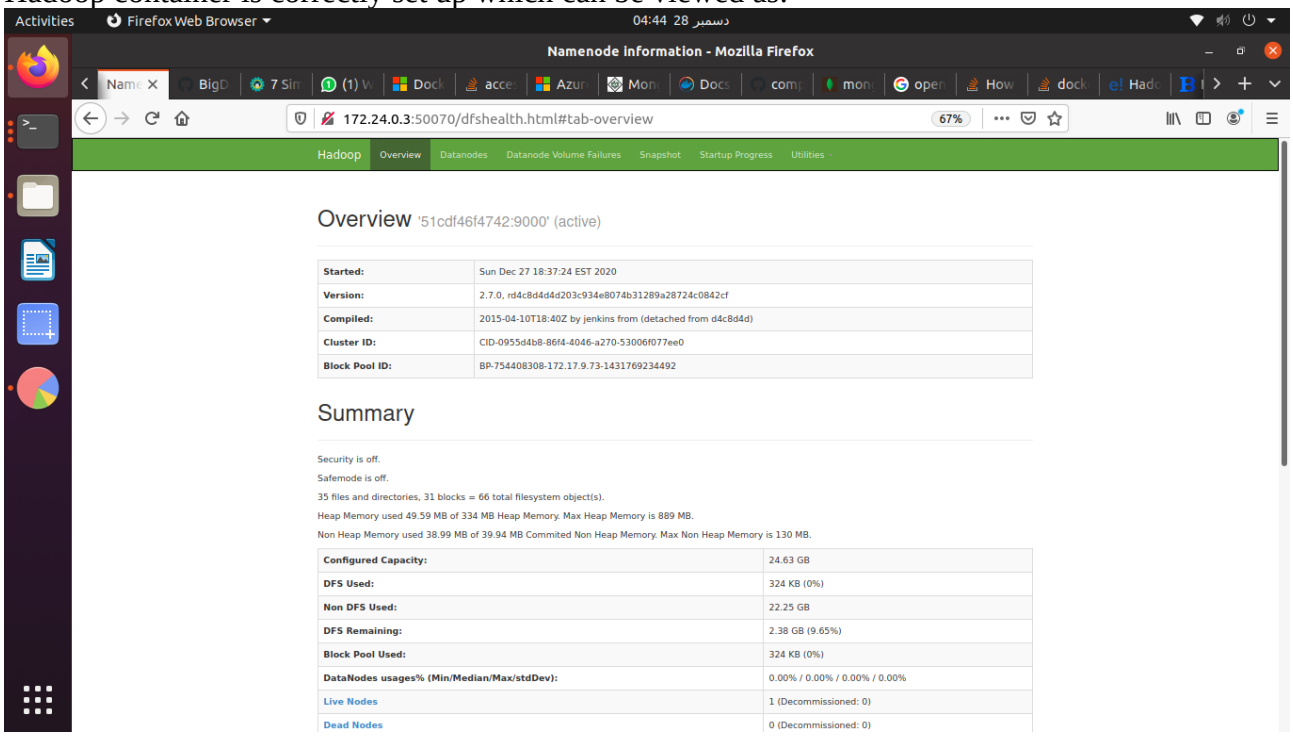
Updating docker compose to add another container for hadoop which links to pymongo container


Activities
Terminal
04:43 28 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano docker-compose.yml
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo docker-compose up
WARNING: The PWD variable is not set. Defaulting to a blank string.
Creating network "tazeen_bridge" with the default driver
Creating tazeen_py-mongo_1 ... done
Creating tazeen_hdfs_1   ... done
Attaching to tazeen_py-mongo_1, tazeen_hdfs_1
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.231+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
hdfs_1 | /
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.234+00:00"},"s":"W", "c":"ASIO",   "id":22601, "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.235+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
hdfs_1 | Starting sshd: [OK]
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.235+00:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"1f7d90ced38e"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.235+00:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"4.4.2","gitVersion":"15e73dc5738d2278b688f8929aee605fe4279b0e","openSSLVersion":"OpenSSL 1.1.1 11 Sep 2018","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu1804","distarch":"x86_64","target_arch":"x86_64"}}}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.235+00:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"18.04"}}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.235+00:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{"net":{"bindIp":"*"},"security":{"authorization":"enabled"}}}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.237+00:00"},"s":"I", "c":"STORAGE", "id":22270, "ctx":"initandlisten","msg":"Storage engine to use detected by data files","attr":{"dbpath":"/data/db","storageEngine":"wiredTiger"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.237+00:00"},"s":"I", "c":"STORAGE", "id":22297, "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem","tags":["startupWarnings"]}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:18.237+00:00"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Opening WiredTiger","attr":{"config":"create,cache_size=5444M,session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statistics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress,compact_progress],"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.268+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:268056][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 3 through 4"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.326+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:326158][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 4 through 4"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.417+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:417644][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Main recovery lo

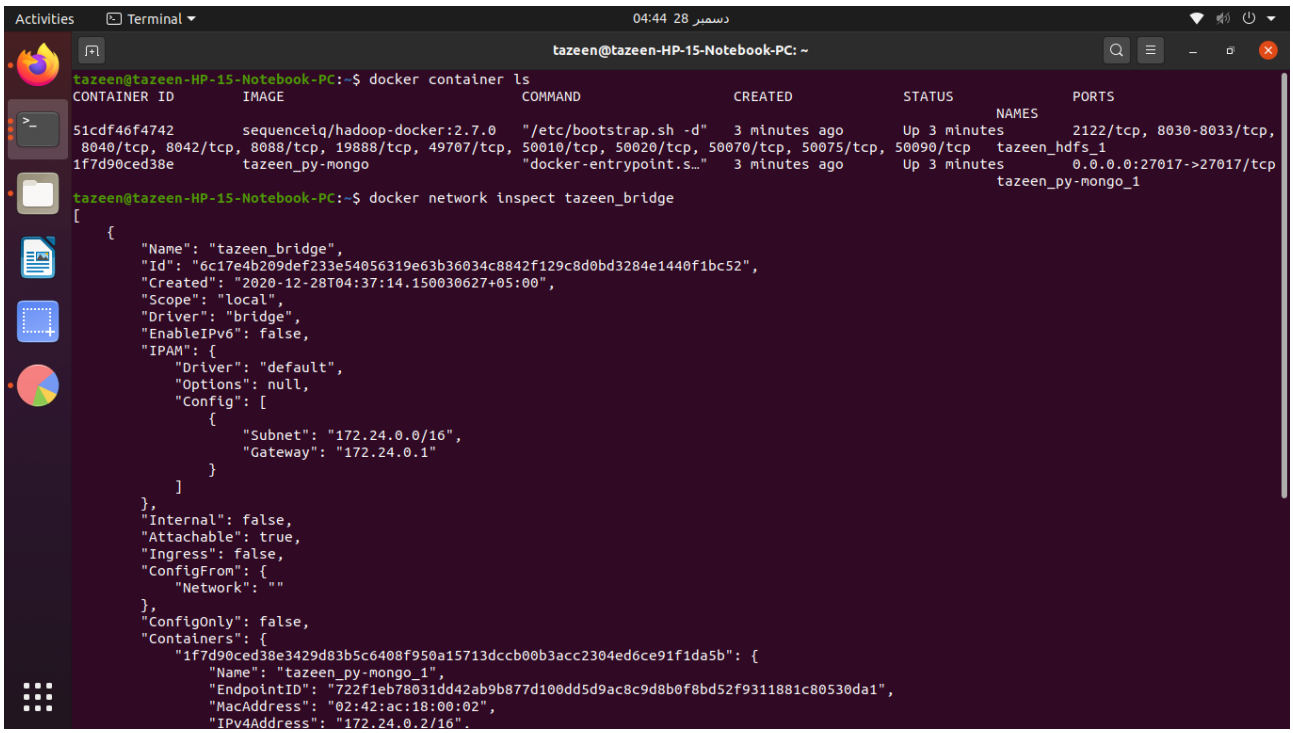

Activities
Terminal
04:43 28 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
op: starting at 3/14464 to 4/256"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.536+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:536855][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 3 through 4"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.631+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:631734][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 4 through 4"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.713+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:713227][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global recovery timestamp: (0, 0)"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:19.713+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609112239:713292][1:0x7f0d75734ac0], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global oldest timestamp: (0, 0)"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.689+00:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":{"durationMillis":2452}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.689+00:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WiredTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$timestamp":{"t":0,"i":0}}}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.691+00:00"},"s":"I", "c":"STORAGE", "id":4366408, "ctx":"initandlisten","msg":"No table logging settings modifications are required for existing WiredTiger tables","attr":{"loggingEnabled":true}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.692+00:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Timestamp monitor starting"}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.771+00:00"},"s":"I", "c":"STORAGE", "id":20536, "ctx":"initandlisten","msg":"Flow Control is enabled on this deployment"}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.773+00:00"},"s":"I", "c":"FTDC",   "id":20625, "ctx":"initandlisten","msg":"Initializing full-time diagnostic data capture","attr":{"dataDirectory":"/data/db/diagnostic.data"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.776+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"/tmp/mongodb-27017.sock"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.776+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"0.0.0.0"}}
py-mongo_1 | {"t":{"$date":"2020-12-27T23:37:20.776+00:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"listener","msg":"Waiting for connections","attr":{"port":27017,"ssl":"off"}}
hdfs_1 | Starting namenodes on [51cdf46f4742]
hdfs_1 | 51cdf46f4742: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-51cdf46f4742.out
hdfs_1 | localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-51cdf46f4742.out
hdfs_1 | Starting secondary namenodes [0.0.0.0]
hdfs_1 | 0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-51cdf46f4742.out
hdfs_1 | starting yarn daemons
hdfs_1 | starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanager-51cdf46f4742.out
hdfs_1 | localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-51cdf46f4742.out

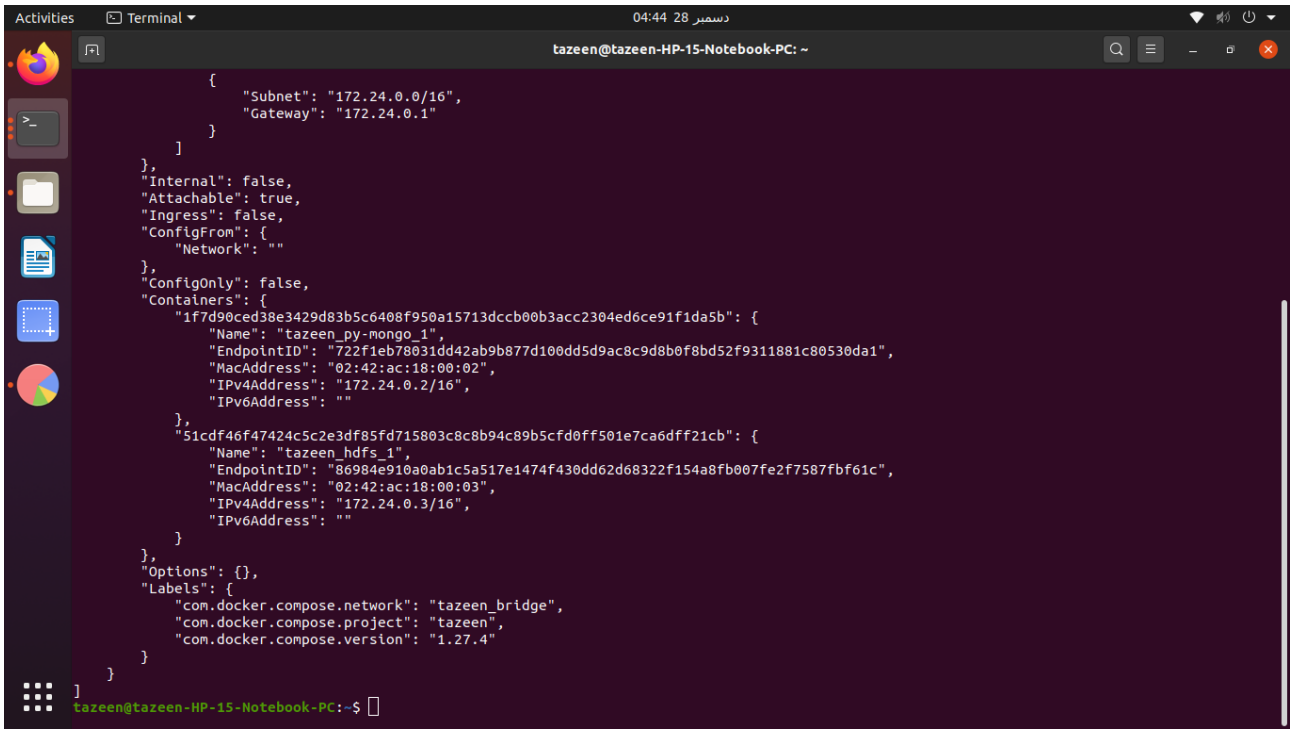


Hadoop container is correctly set up which can be viewed as:



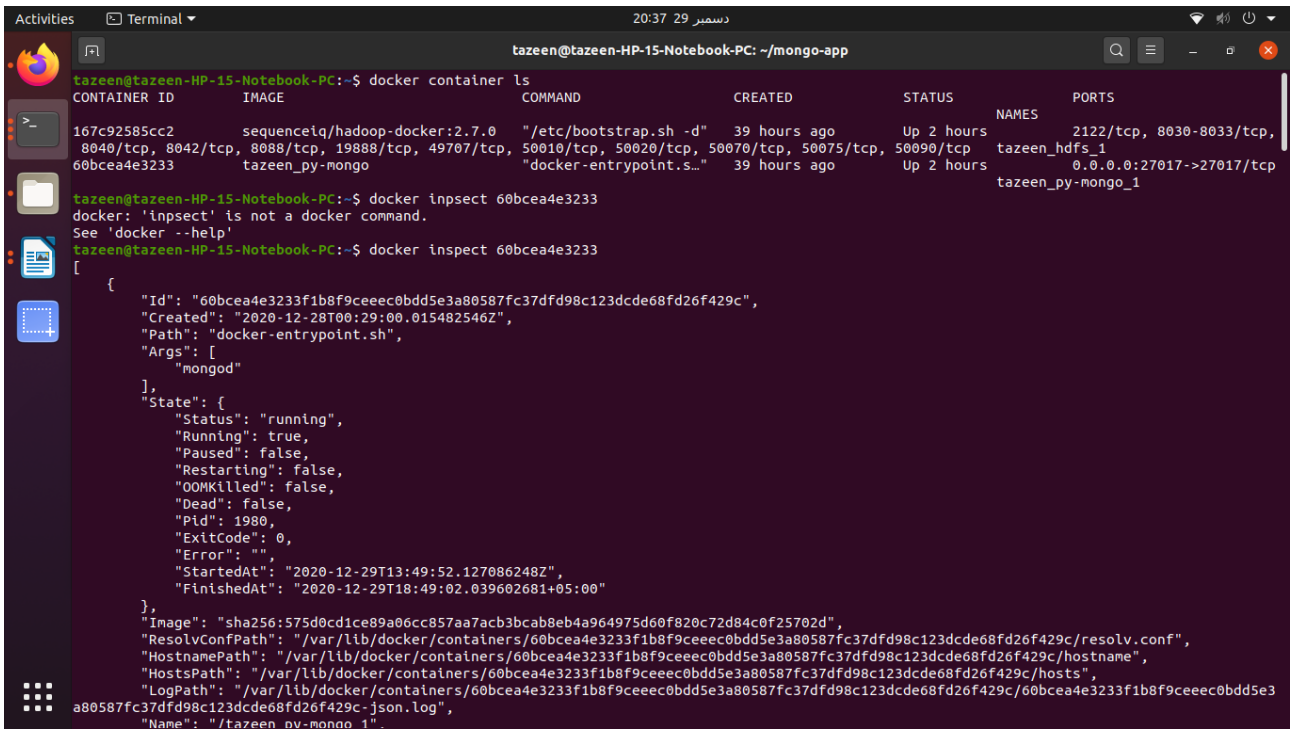
The network used to connect both containers is: tazeen_bridge which is a user defined bridge network. Both containers are connected on the same network which can be seen in below screenshot.

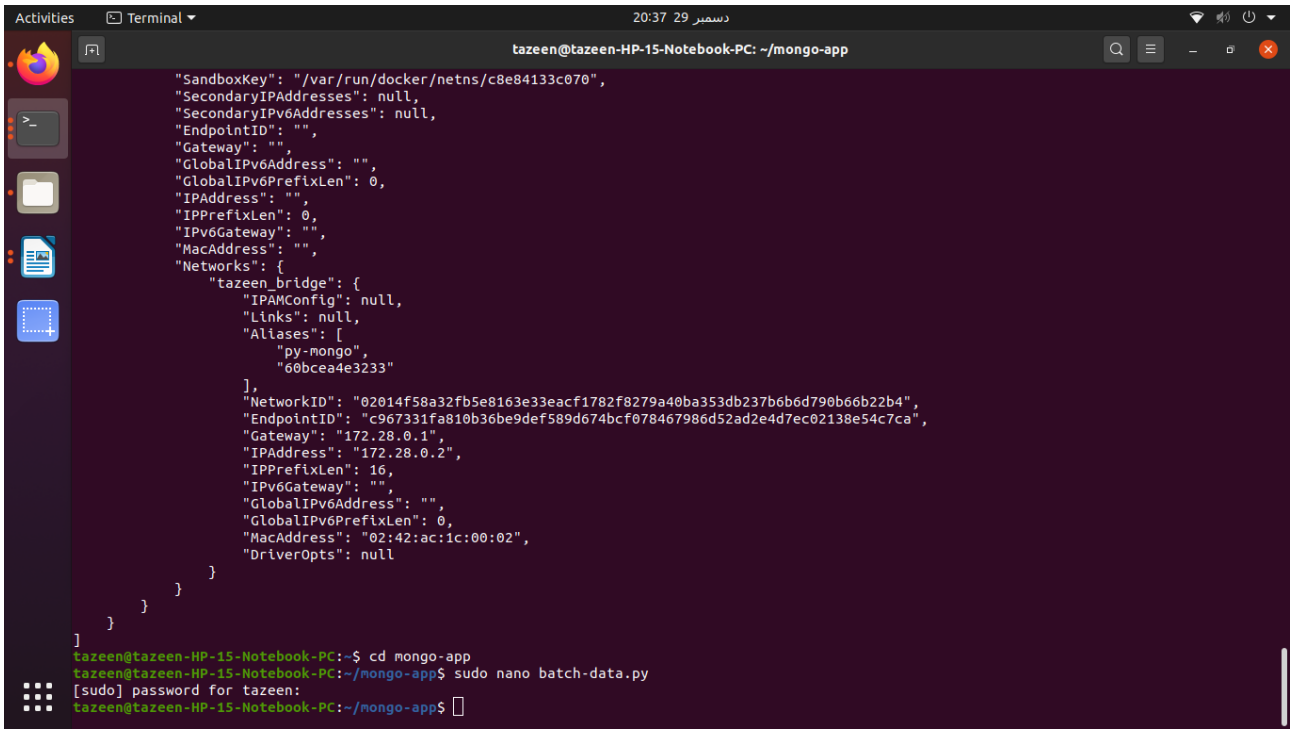




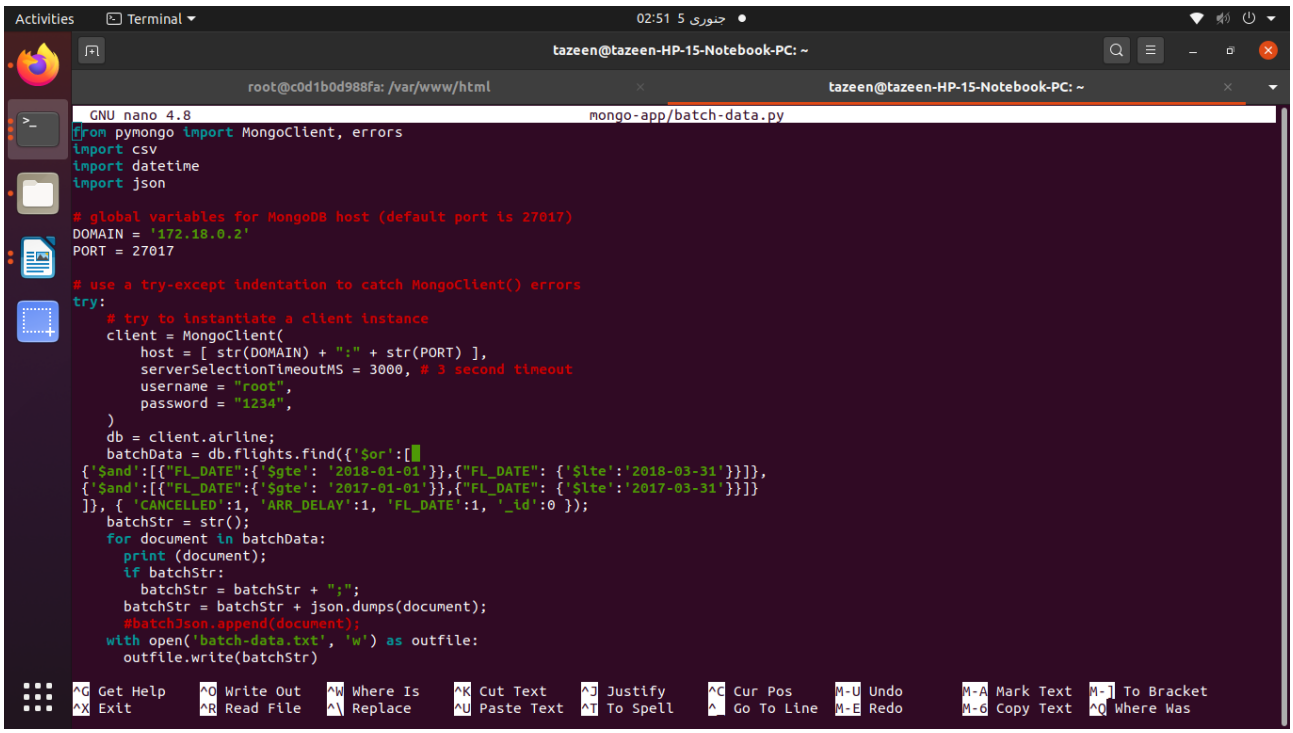
Now getting data for hadoop batch processing from mongodb. First checking the ip address to connect to database using pymongo container

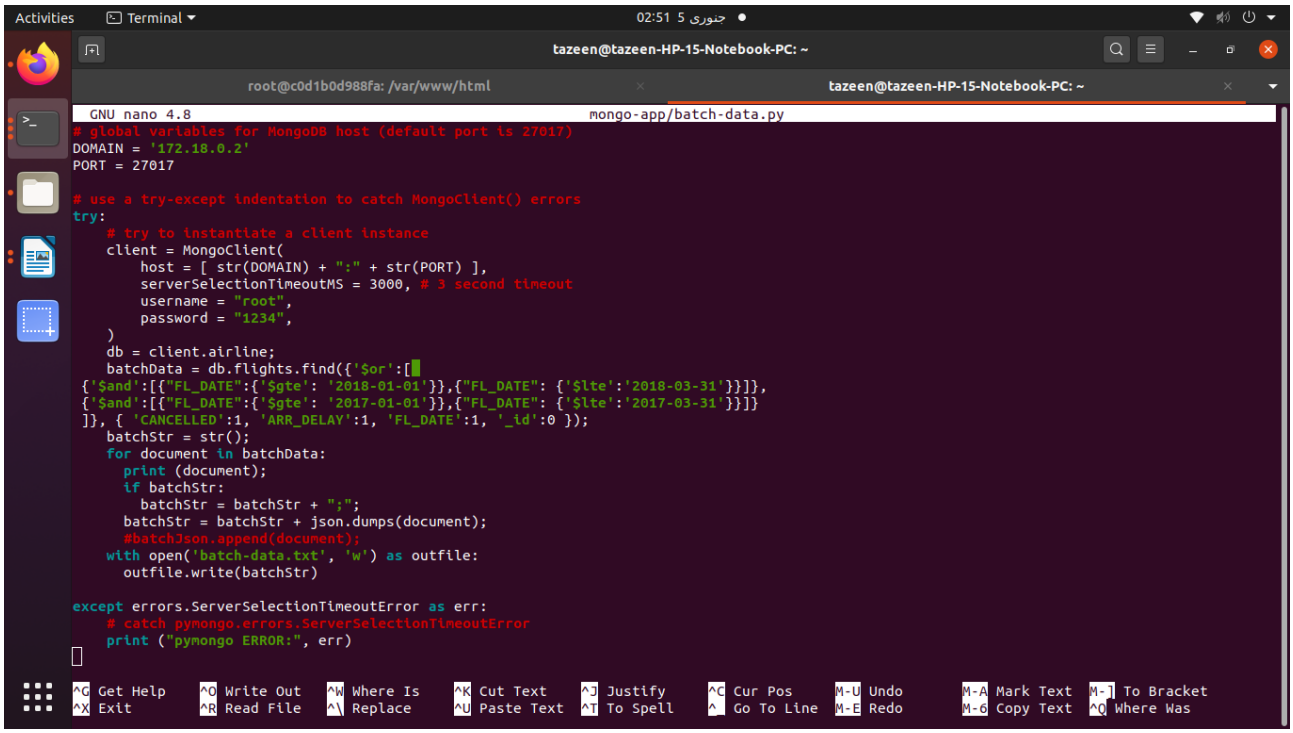
.



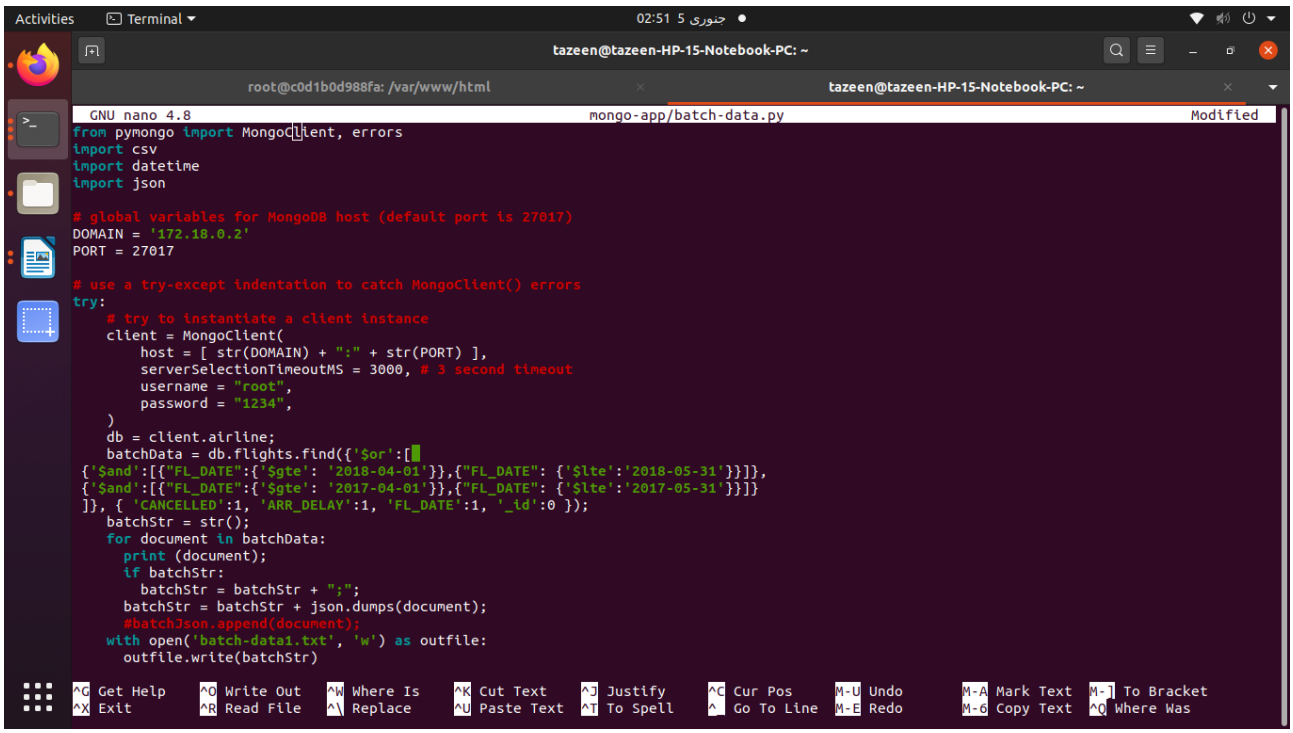


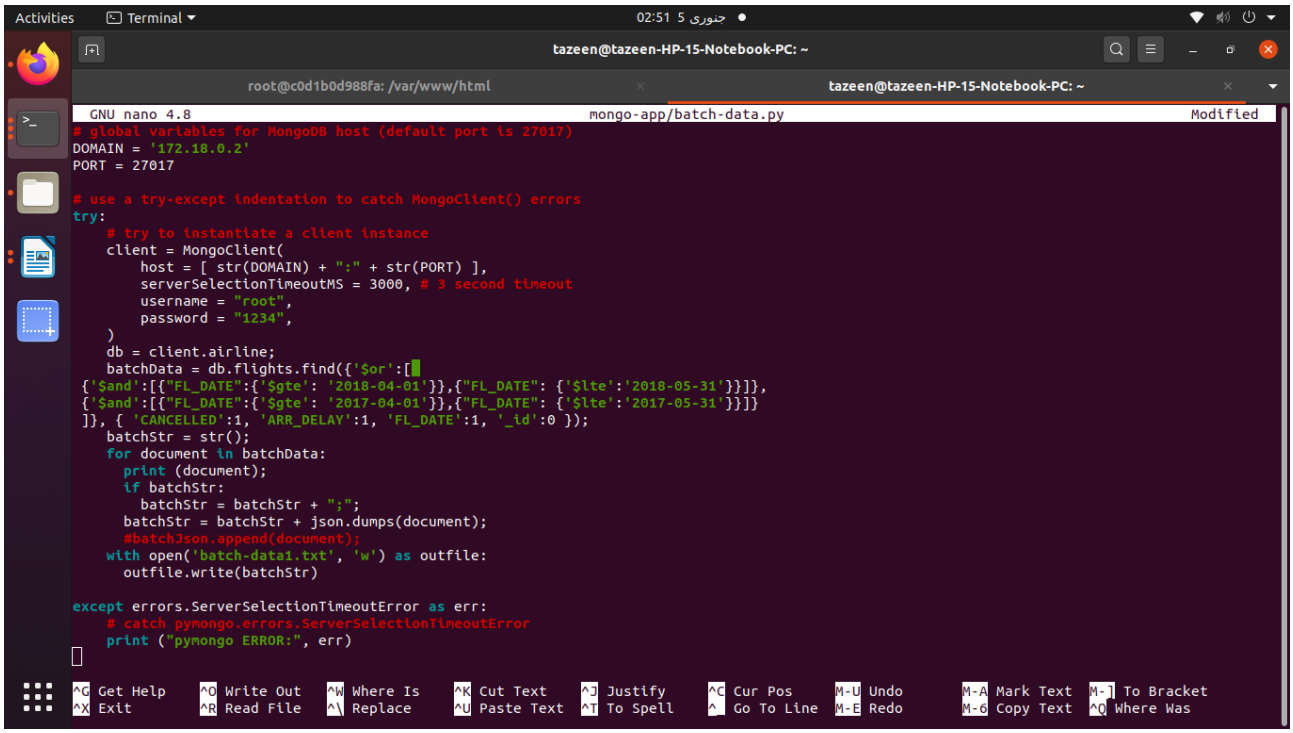
Creating multiple python scripts to export mongodb data for hadoop container. Because scripts take a lot of time in processing and stuck due to less resources. Each script creates text file data for specific months, total data is used from 1/1/2017 – 31/5/2017 and 1/1/2018 – 31/5/2018. The screenshots for batch-data.py is added below, on it's execution mongodb data is exported in text files shared using same bridge network.



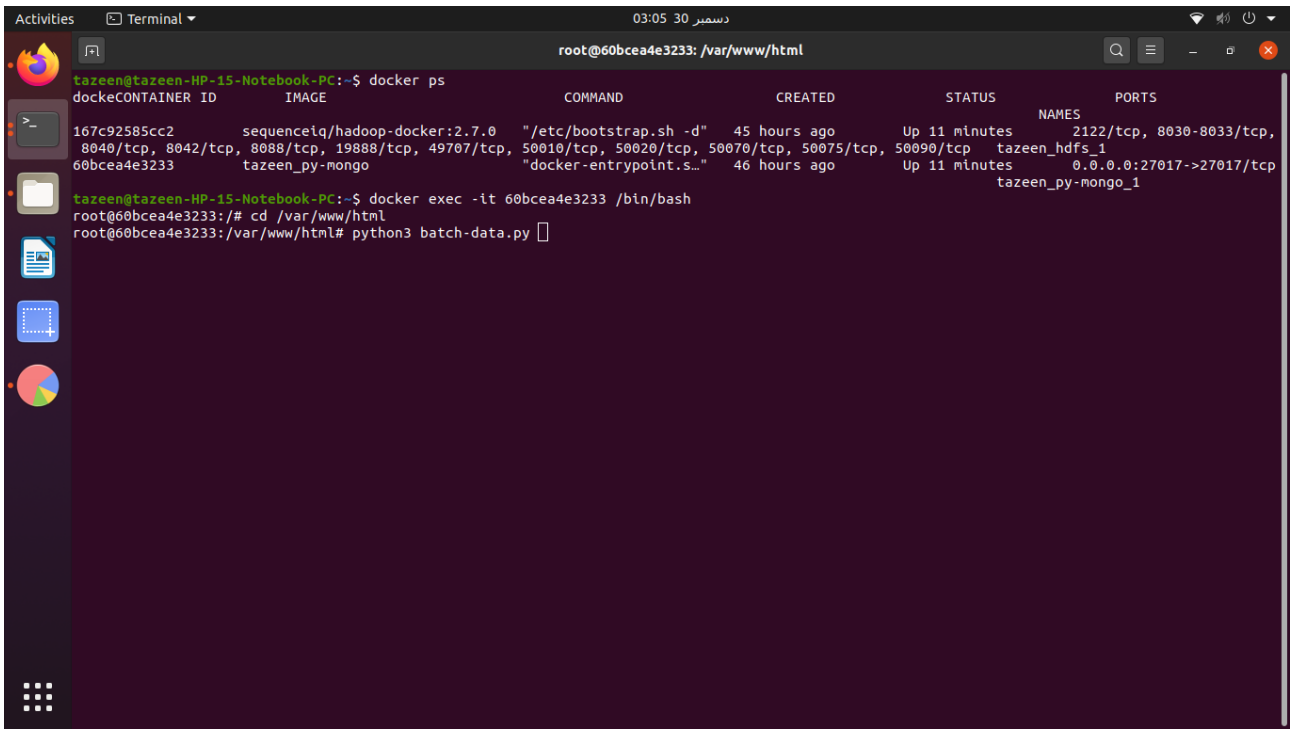


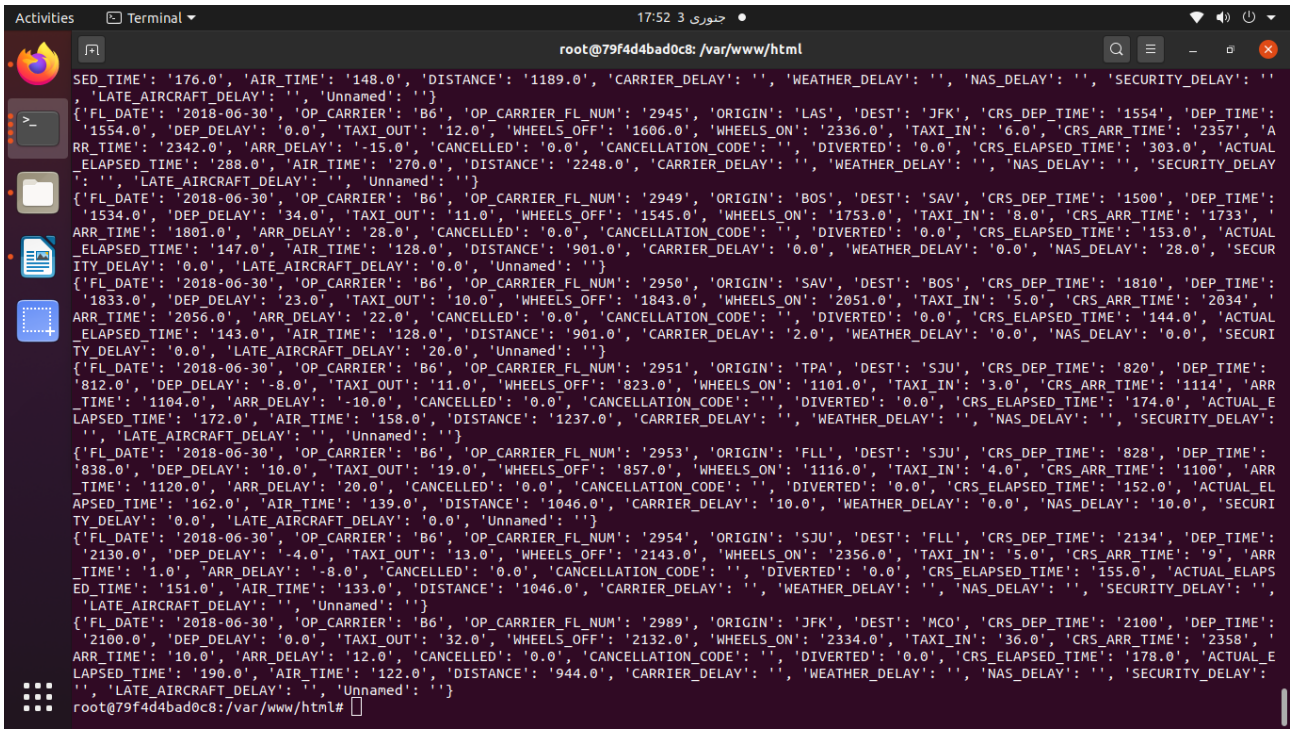
After execution, same file is updated for different data.





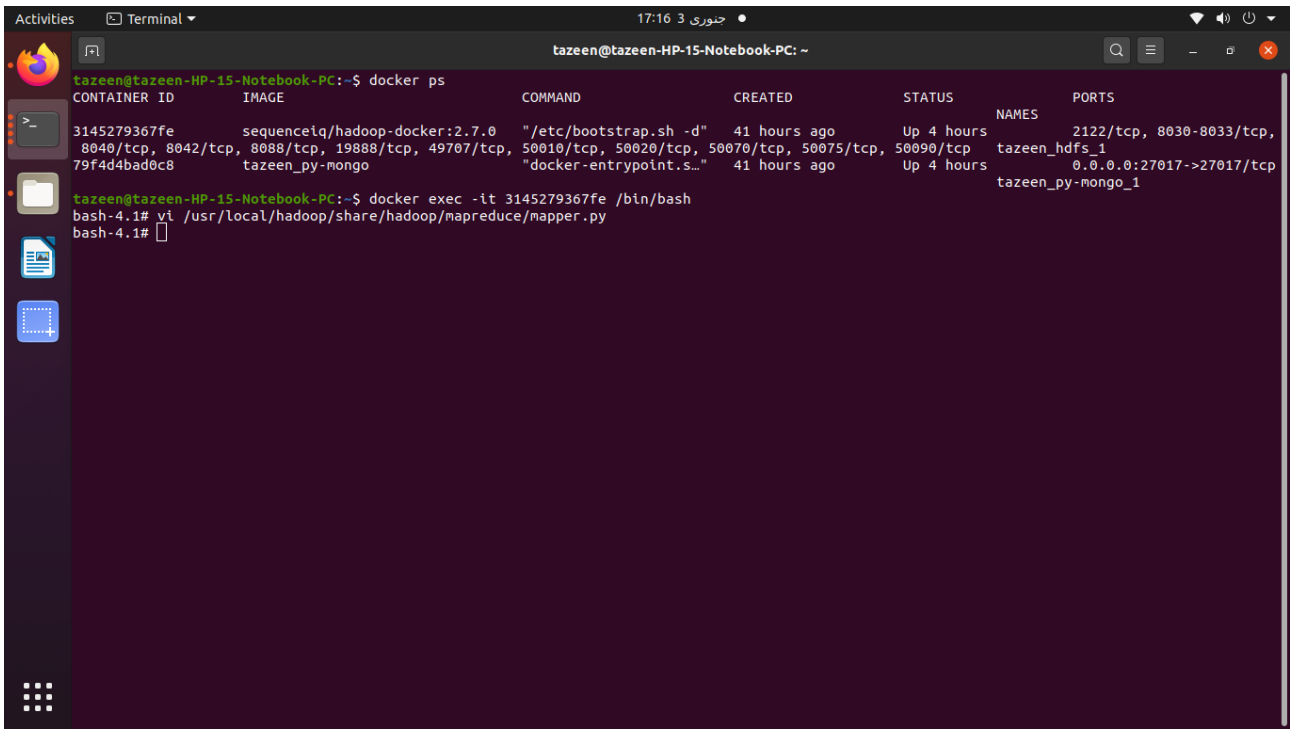
Executing the scripts by the following command

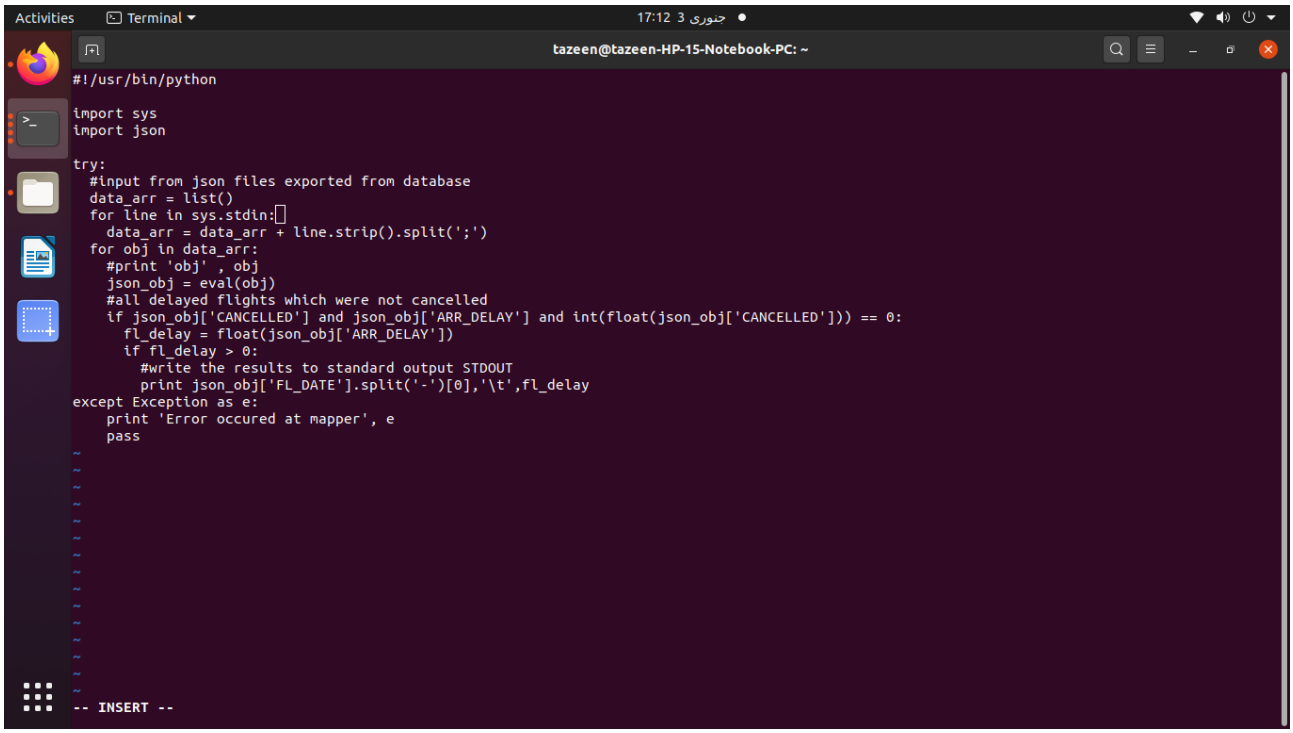


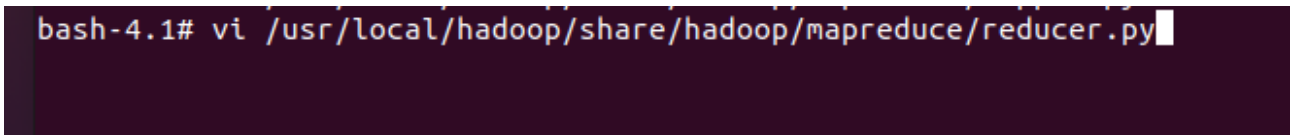


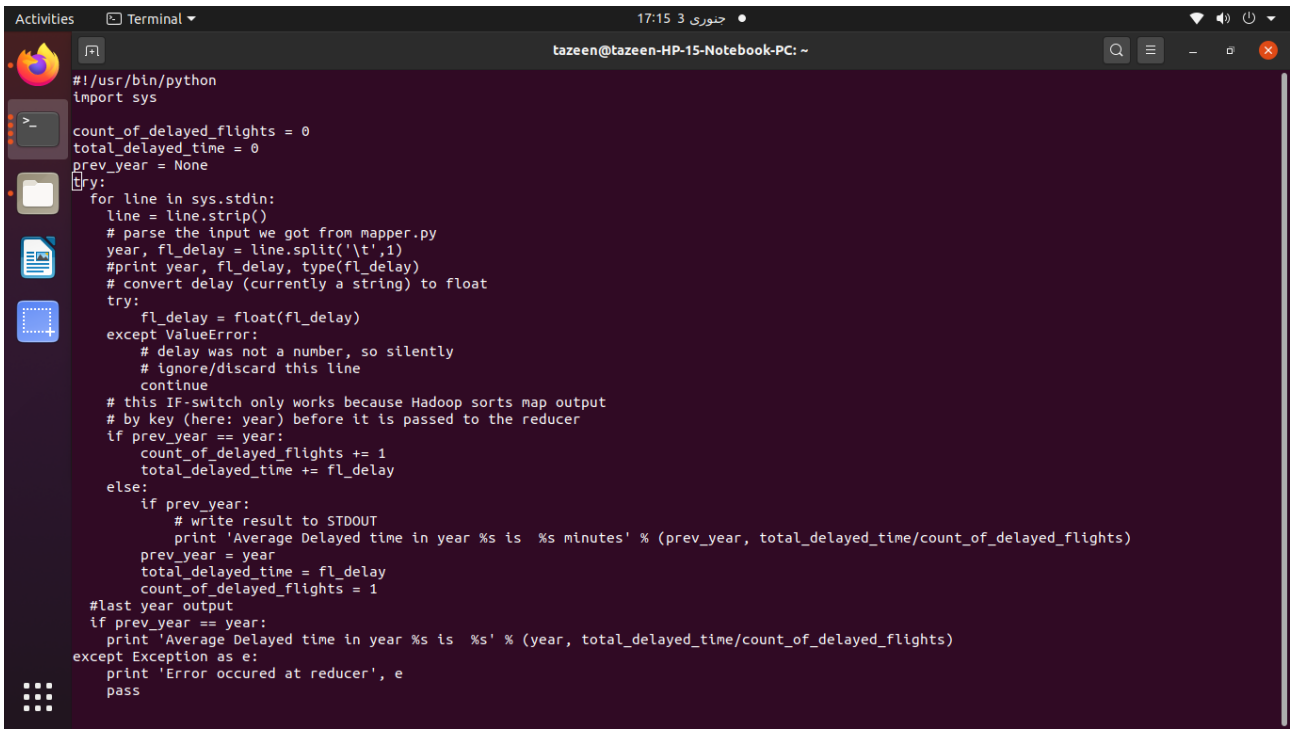
After executing the batch-data.py script with different data, two files batch-data.txt and batch-data1.txt are created which will be used in HDFS.

Now taking the above created files shared on the same bridge network as input files for hadoop and executing a map/reduce job. The map/reduce job calculates the average delay time minutes for each year. The data consists of canceled and delayed flights. Firstly, removing the canceled flights to find the correct delay time. Secondly, only considering the positive delay time as negative represents the flights were departed late. The map and reduce job files are shown below.

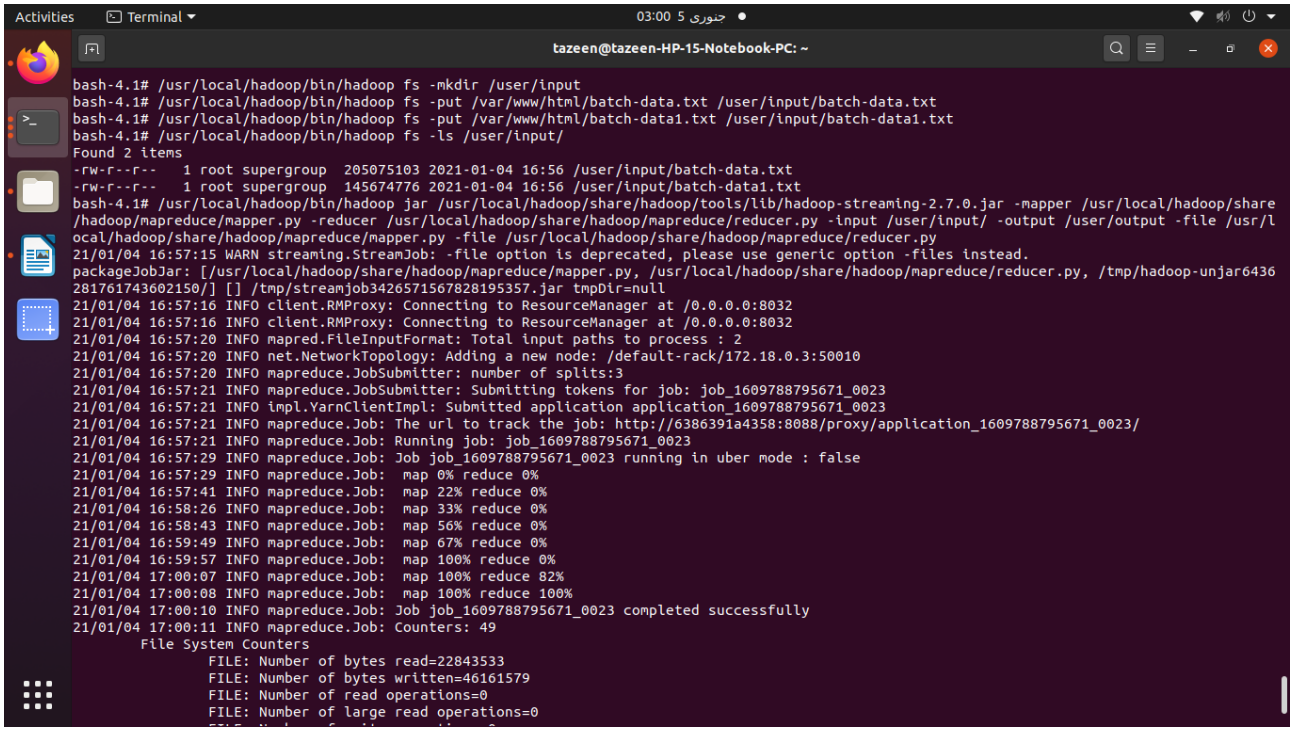

Activities
Terminal
17:16 3 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
tazeen@tazeen-HP-15-Notebook-PC:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
3145279367fe sequenceiq/hadoop-docker:2.7.0 "/etc/bootstrap.sh -d" 41 hours ago Up 4 hours 2122/tcp, 8030-8033/tcp, 8040/tcp, 8042/tcp, 8088/tcp, 19888/tcp, 49707/tcp, 50010/tcp, 50020/tcp, 50070/tcp, 50075/tcp, 50090/tcp tazeen_hdfs_1
79f4d4bad0c8 tazeen_py-mongo "docker-entrypoint.s…" 41 hours ago Up 4 hours 0.0.0.0:27017->27017/tcp tazeen_py-mongo_1
tazeen@tazeen-HP-15-Notebook-PC:~$ docker exec -it 3145279367fe /bin/bash
bash-4.1# vi /usr/local/hadoop/share/hadoop/mapreduce/mapper.py
bash-4.1#


Activities
Terminal
17:12 3 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
#!/usr/bin/python
import sys
import json
try:
#input from json files exported from database
data_arr = list()
for line in sys.stdin:
data_arr = data_arr + line.strip().split(';')
for obj in data_arr:
#print 'obj' , obj
json_obj = eval(obj)
#all delayed flights which were not cancelled
if json_obj['CANCELLED'] and json_obj['ARR_DELAY'] and int(float(json_obj['CANCELLED'])) == 0:
fl_delay = float(json_obj['ARR_DELAY'])
if fl_delay > 0:
#write the results to standard output STDOUT
print json_obj['FL_DATE'].split('-')[0],'\t',fl_delay
except Exception as e:
print 'Error occured at mapper', e
pass
-- INSERT --

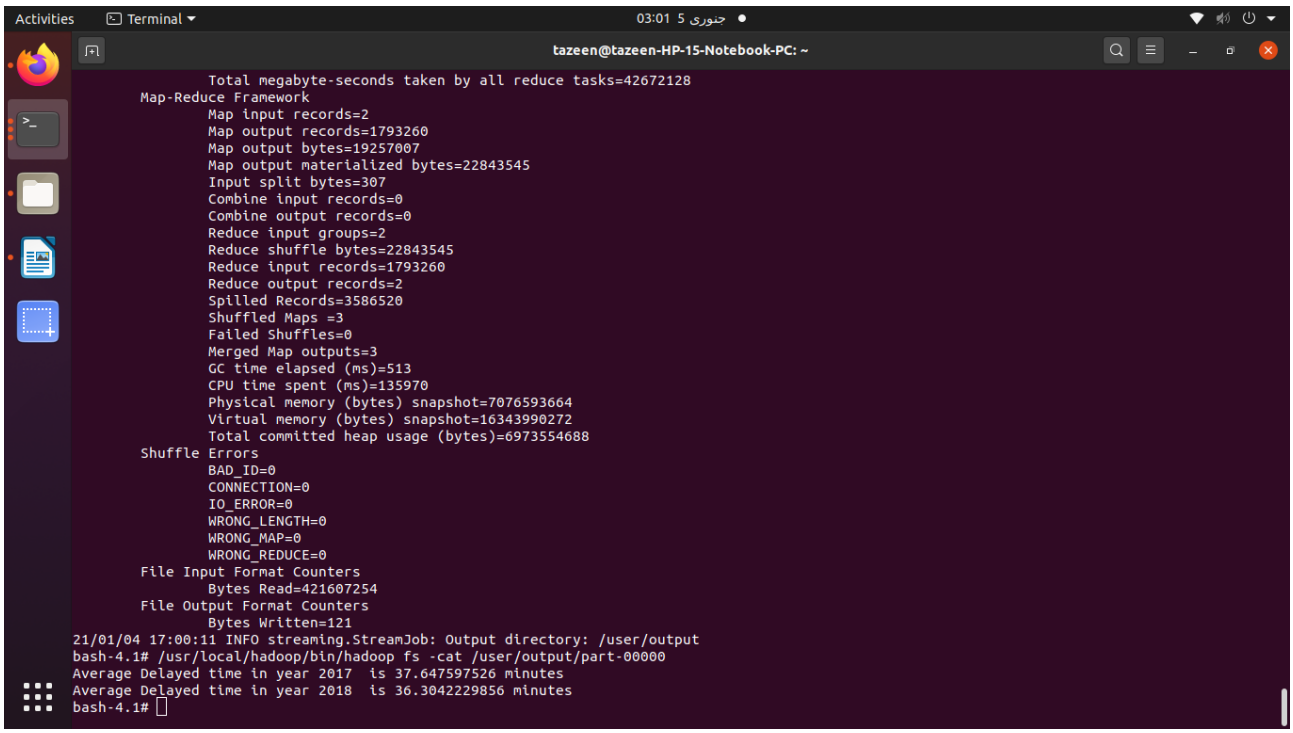

bash-4.1# vi /usr/local/hadoop/share/hadoop/mapreduce/reducer.py



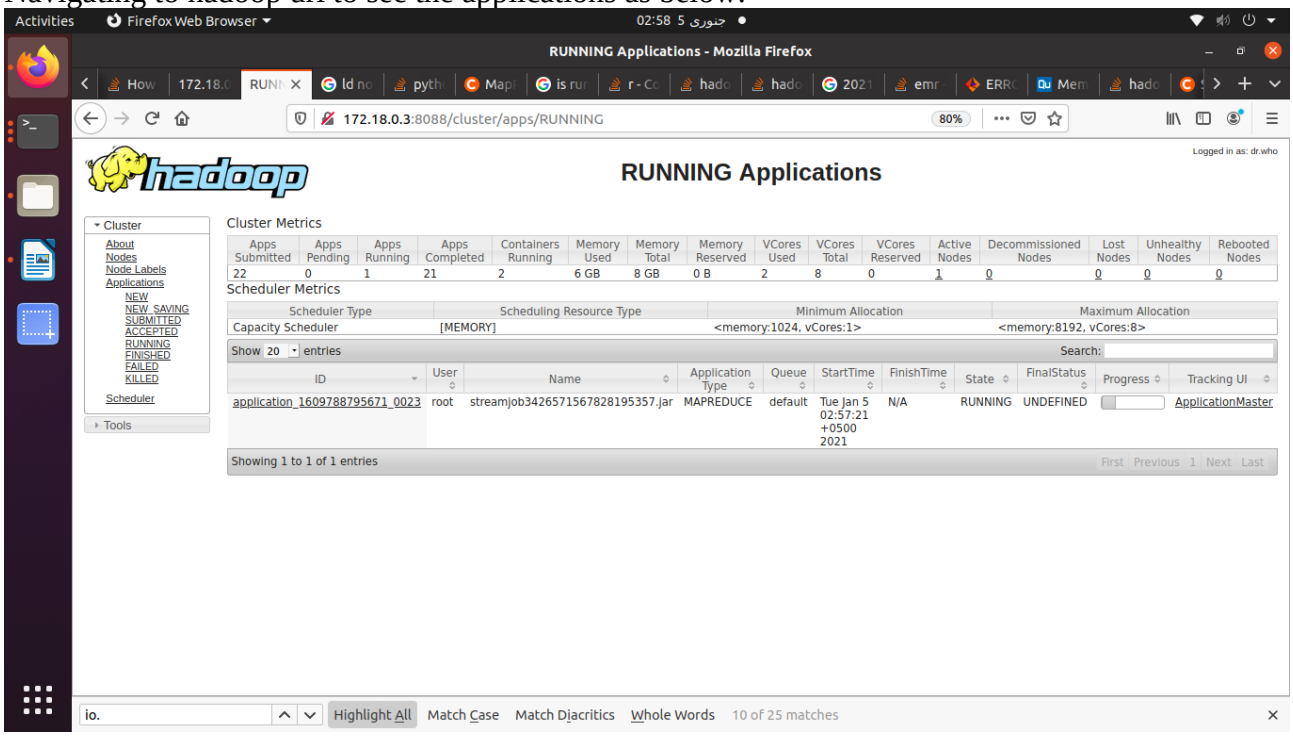
Now putting these batch data files (batch-data.txt and batch-data1.txt) in HDFS. A new folder is created as /user/input which will be for hadoop input data and output will be saved in /user/output.

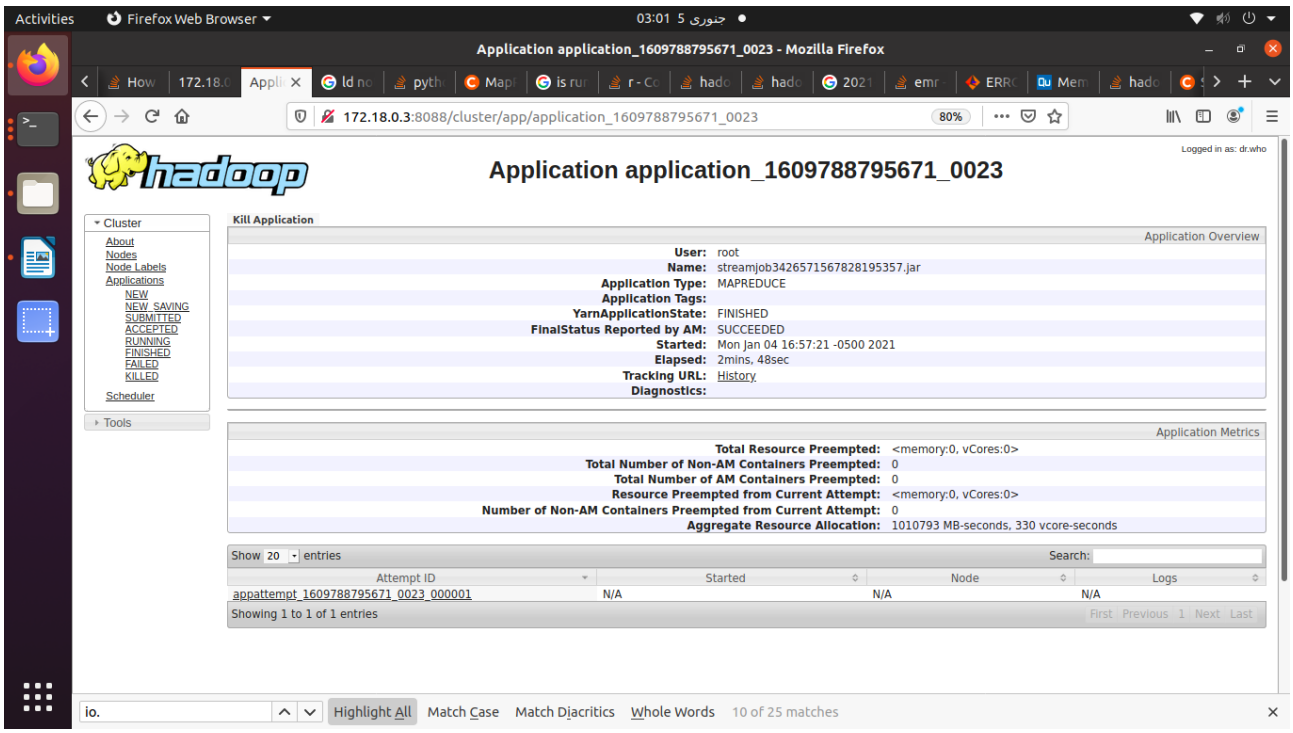


The job is successfully executed and output is saved in /user/output. The output can be viewed as ‘cat /user/output/part-00000.txt’ which contains the output on success.



Navigating to hadoop url to see the applications as below:



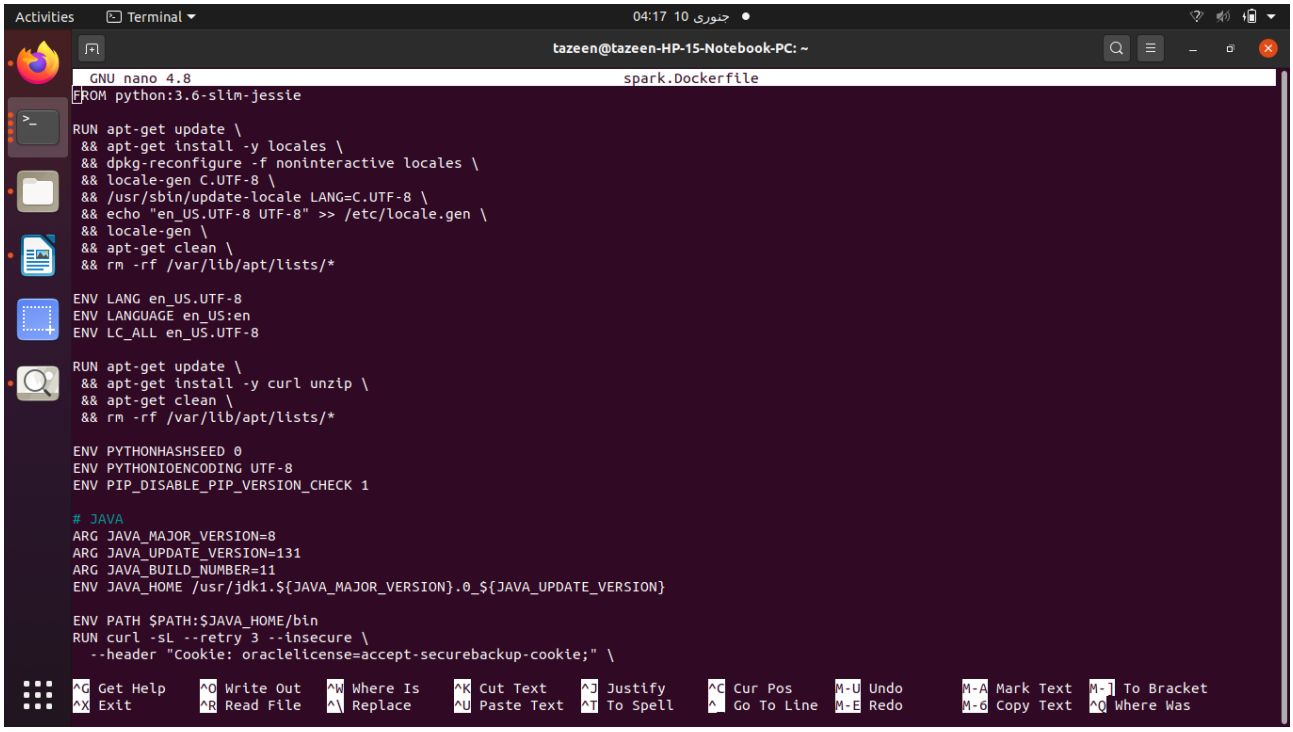


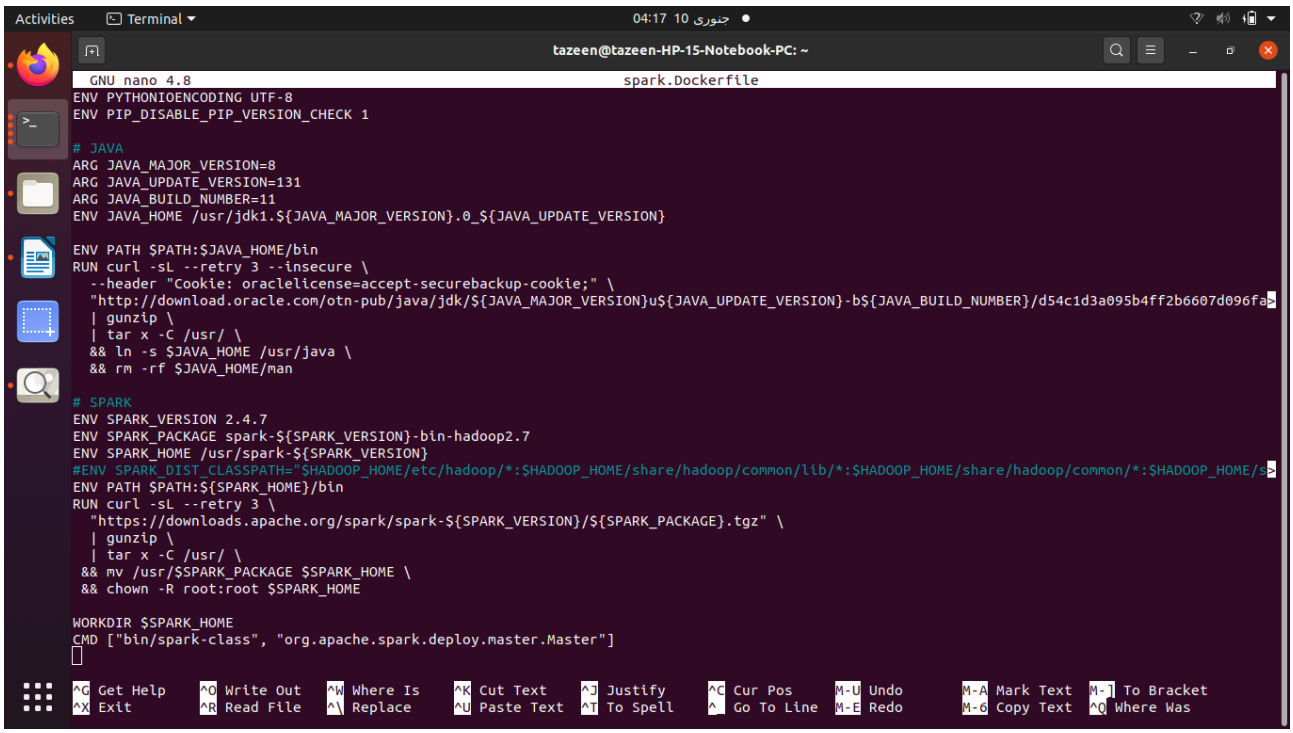
3- Streaming Layer

Now starting the spark job with redis as streaming layer. The data will be read from mongodb and streams will be created in redis which will be used in spark for processing.

The dockerfile for spark is created as following:

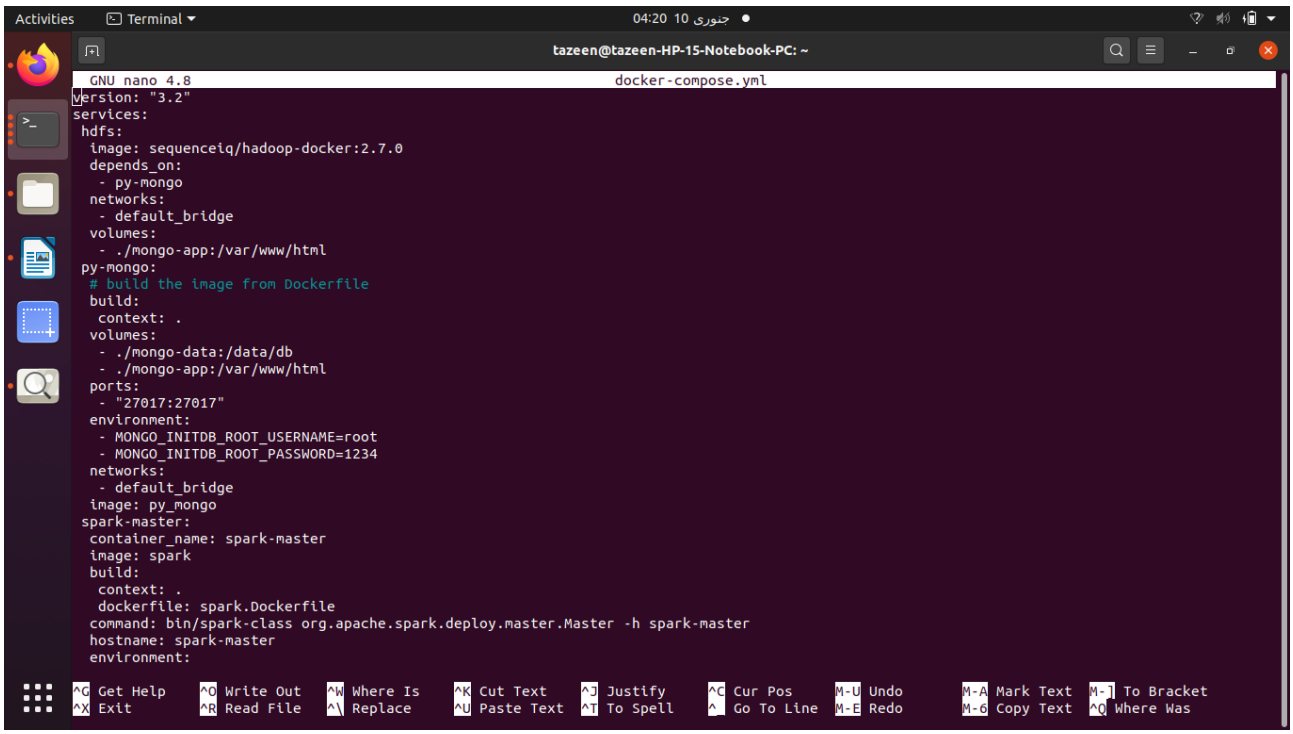
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano spark.Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```

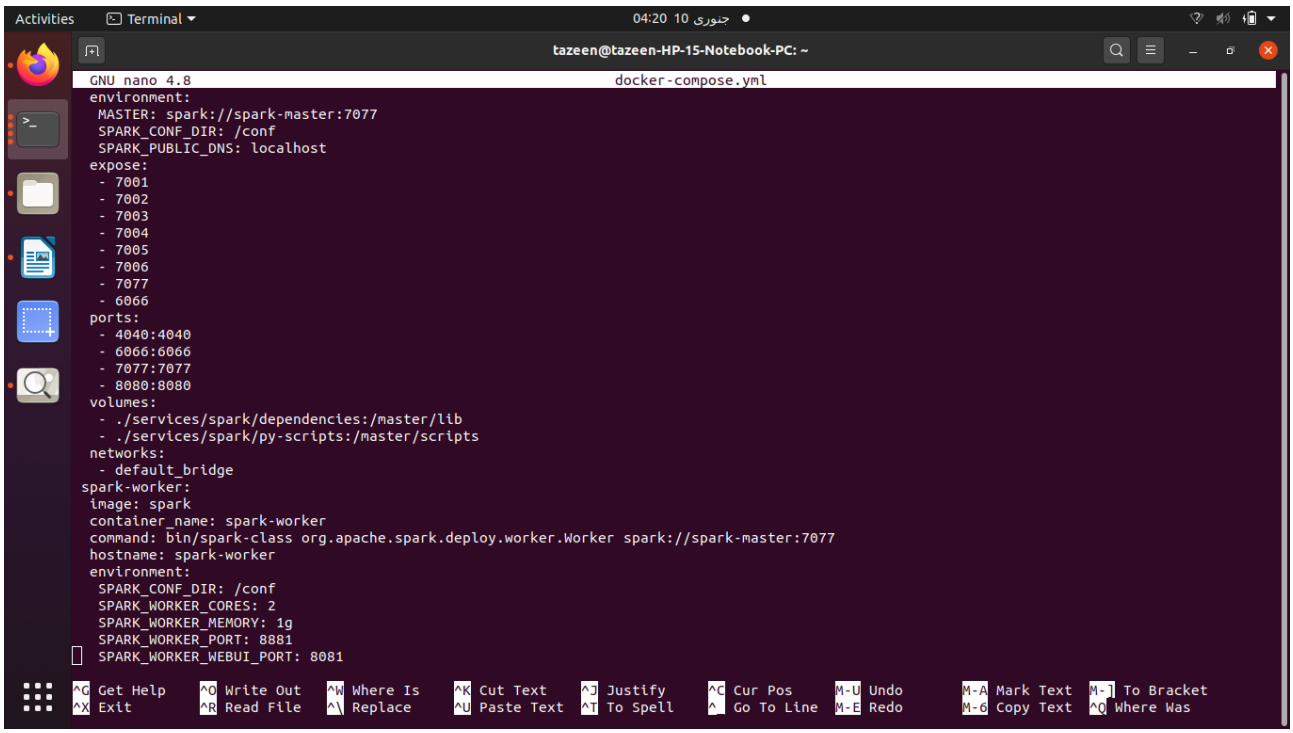


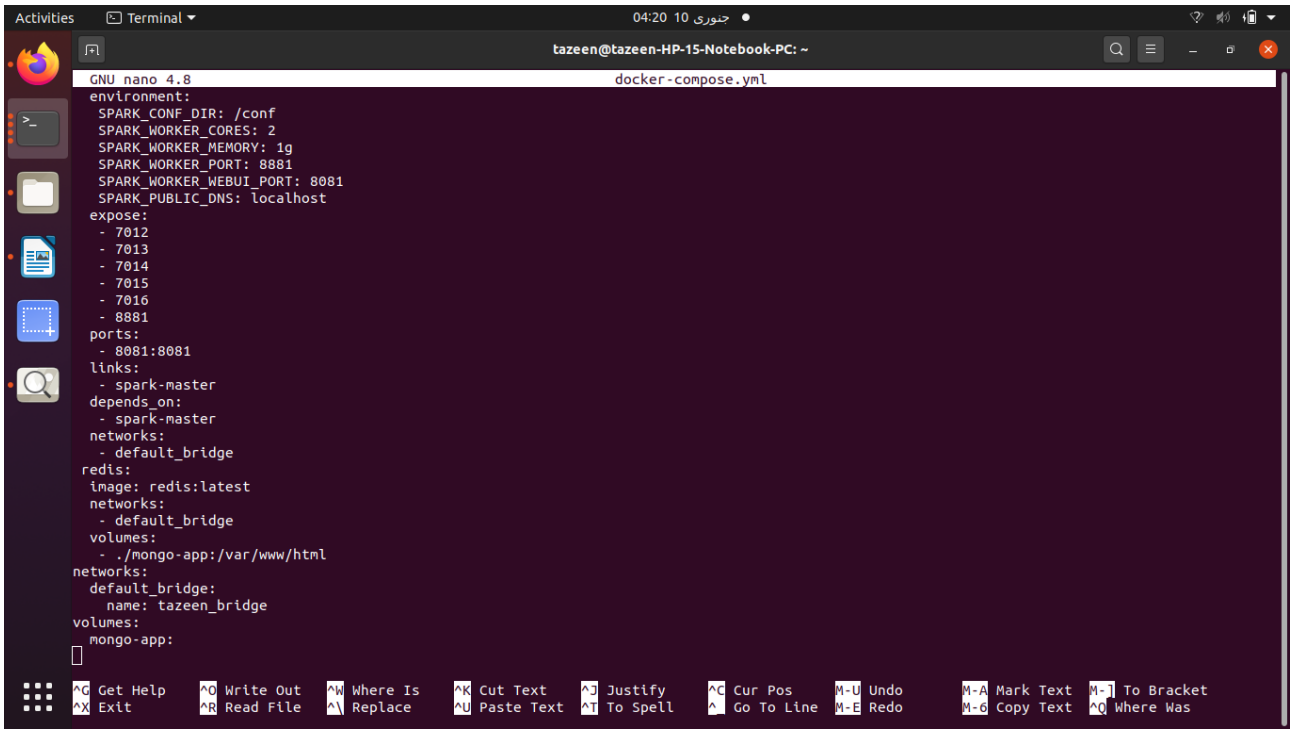


Updating the docker-compose.yml file for streaming layer. Spark master and worker docker are created as separate dockers and another docker is created for redis.

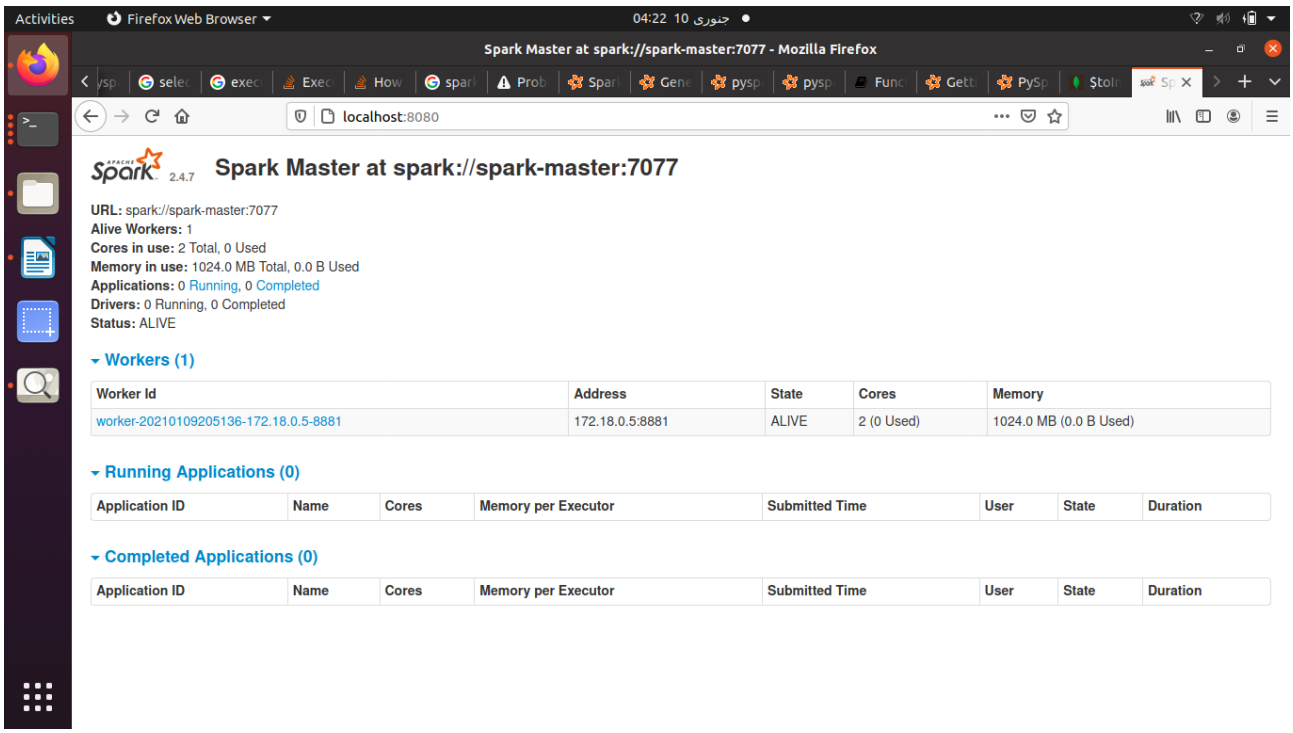
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano spark.Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```

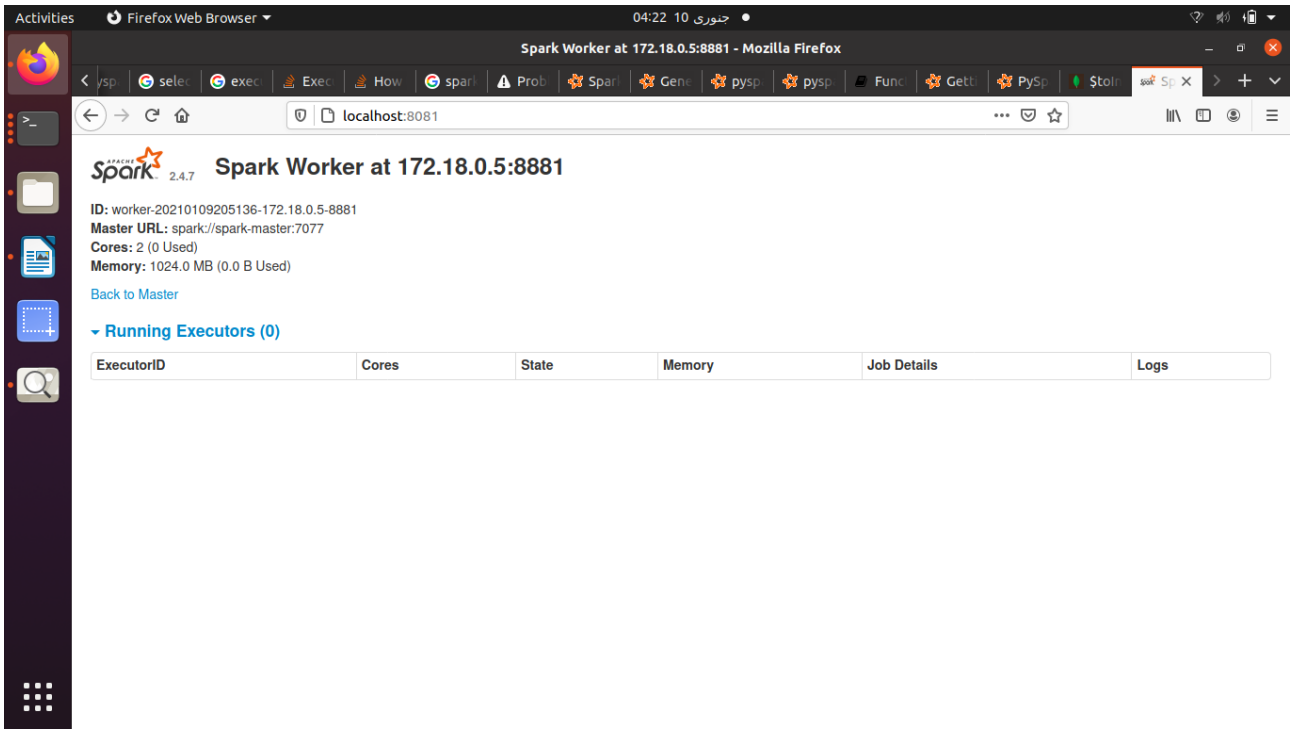




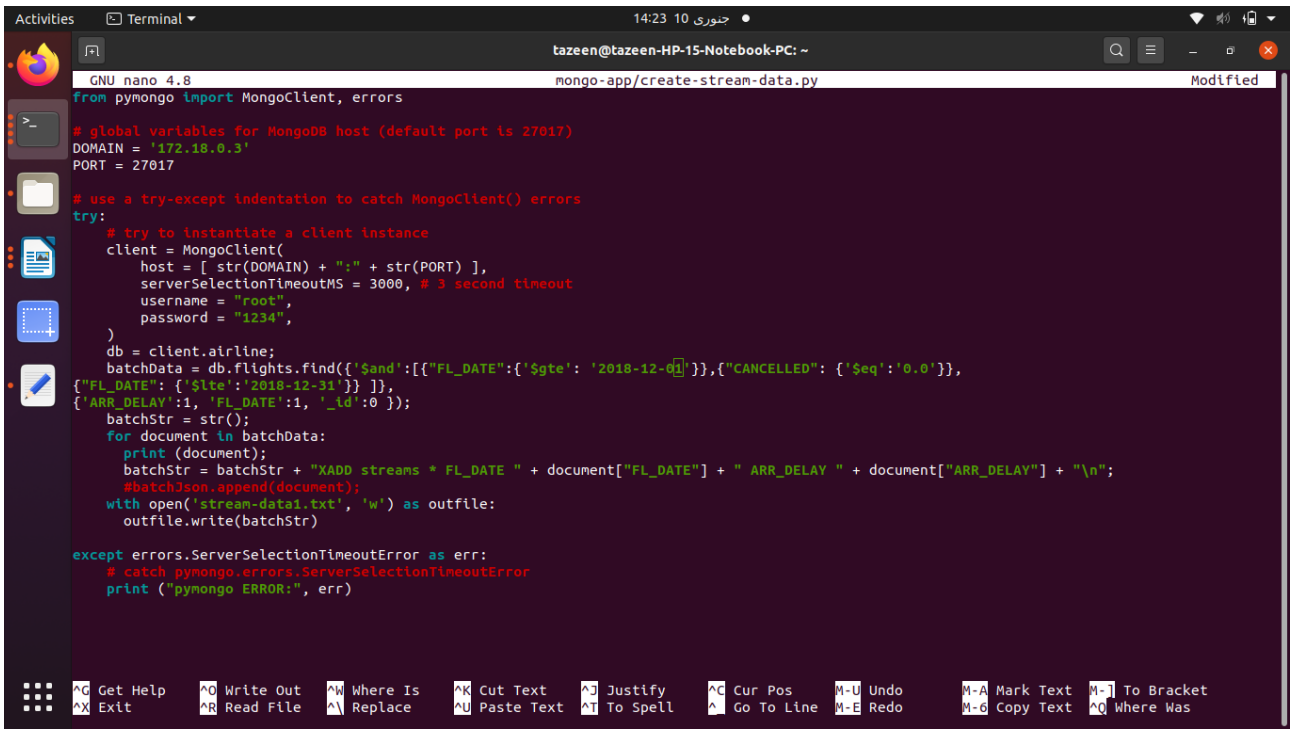


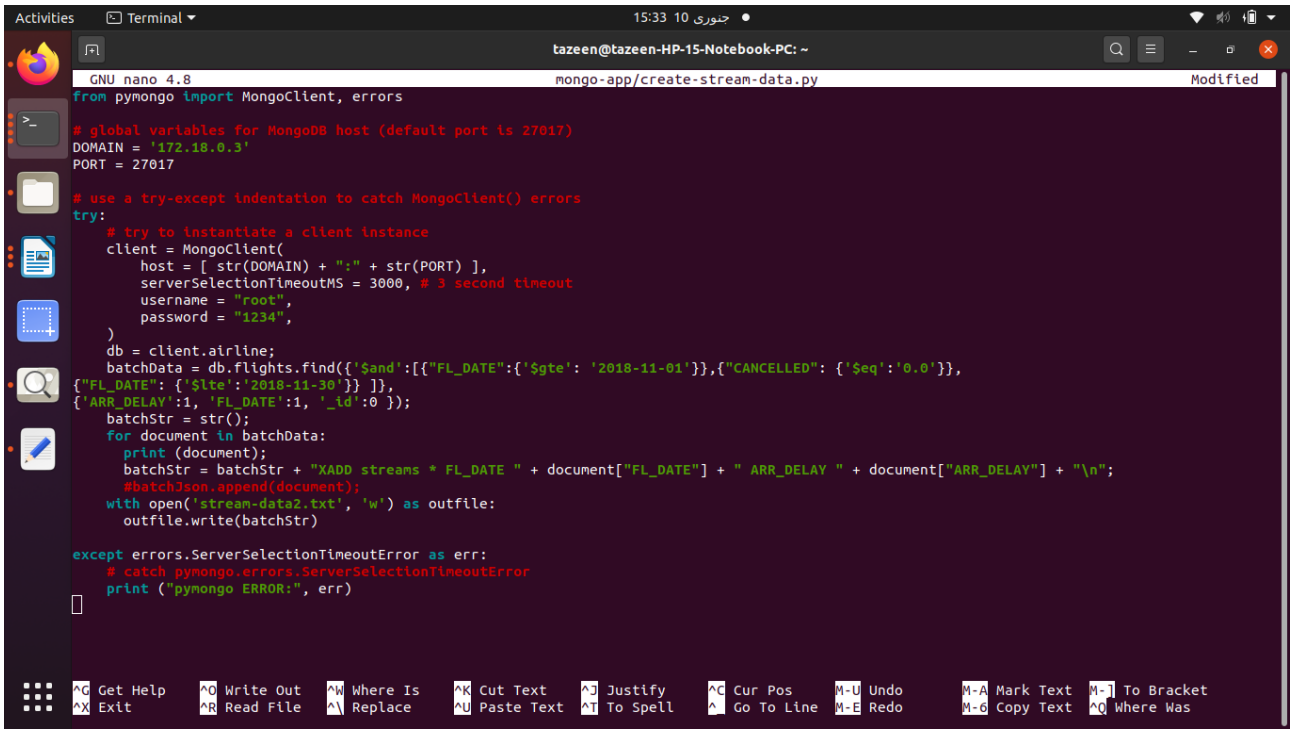
Navigate to localhost:8080 to see spark master is running.



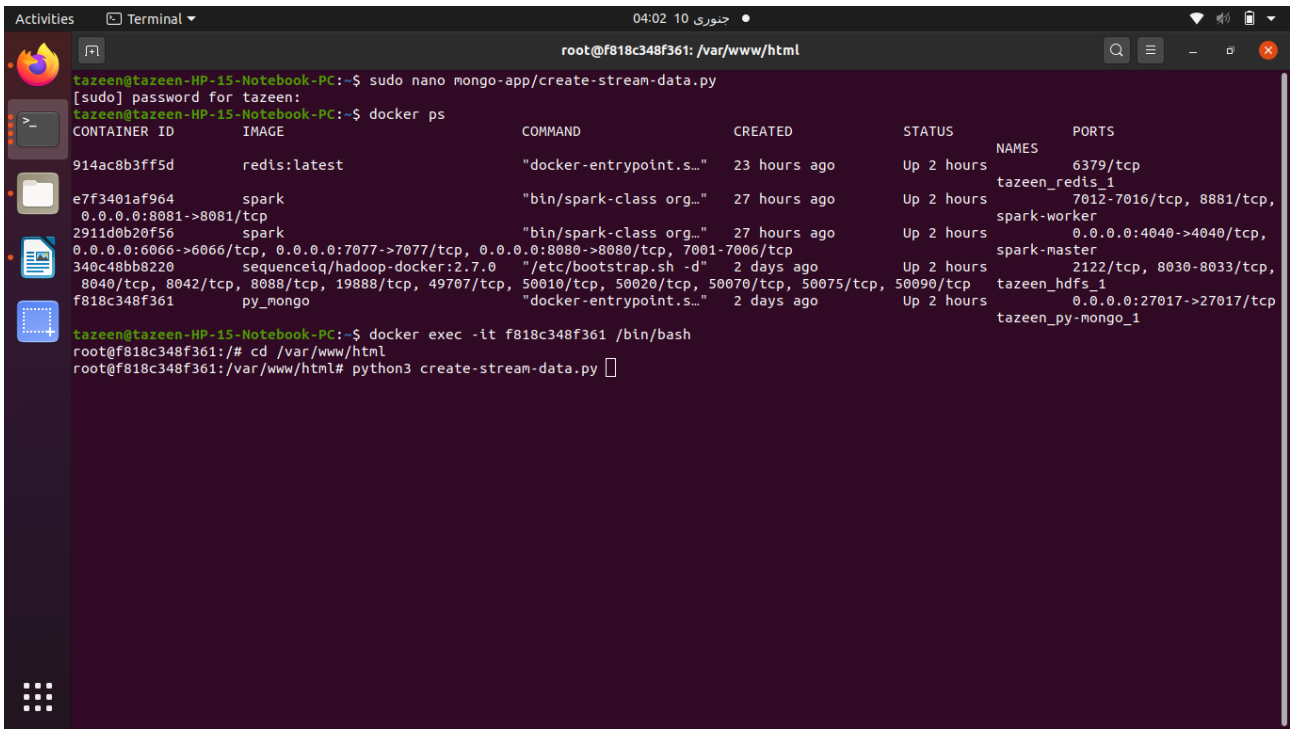


To create streaming data, a text file is generated with redis streams from mongodb database. The data for November and December 2018. The mongodb script is saved as mongo-app/create-stream-data.py to generate data is below. I will only find the data for the flights which are not canceled, therefore, column ‘CANCELLED’ should be 0.



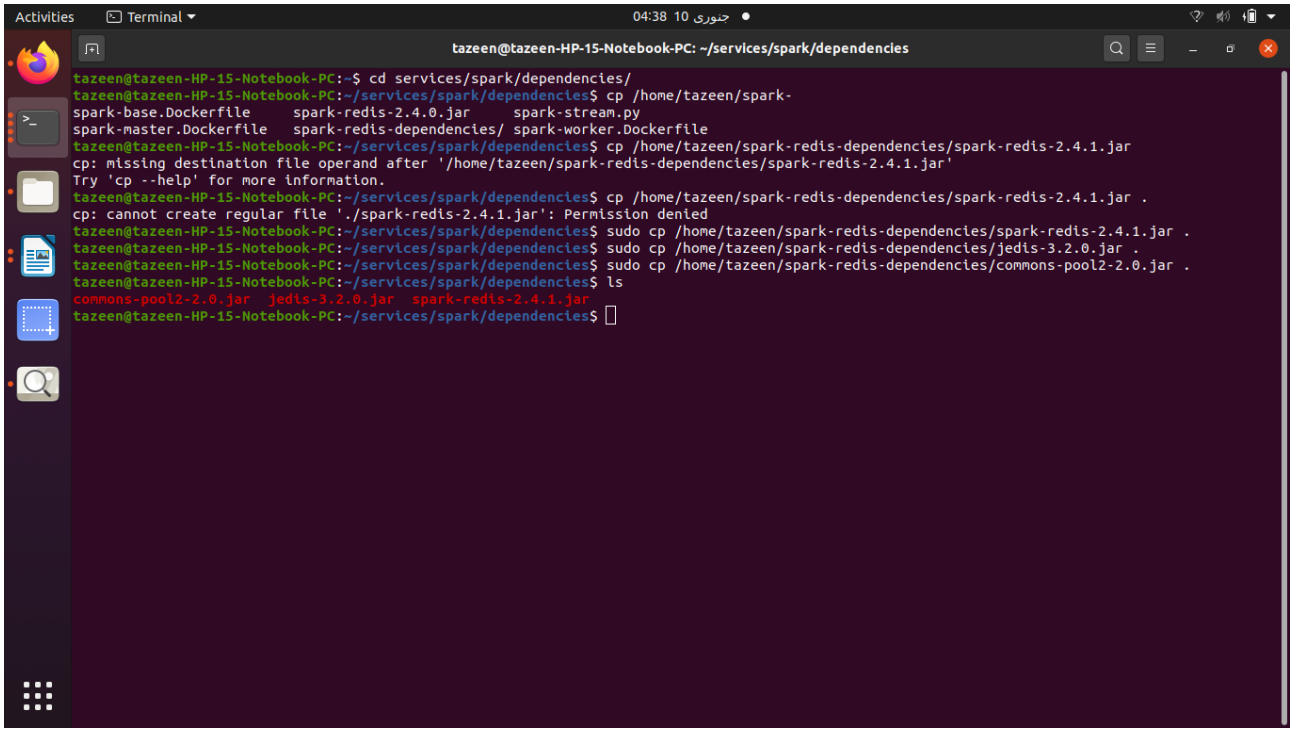


Now executing the script in mongo db docker container. It will create two stream data files as tream-data1.txt and stream-data2.txt. These files will be used by redis docker container to generate streams.

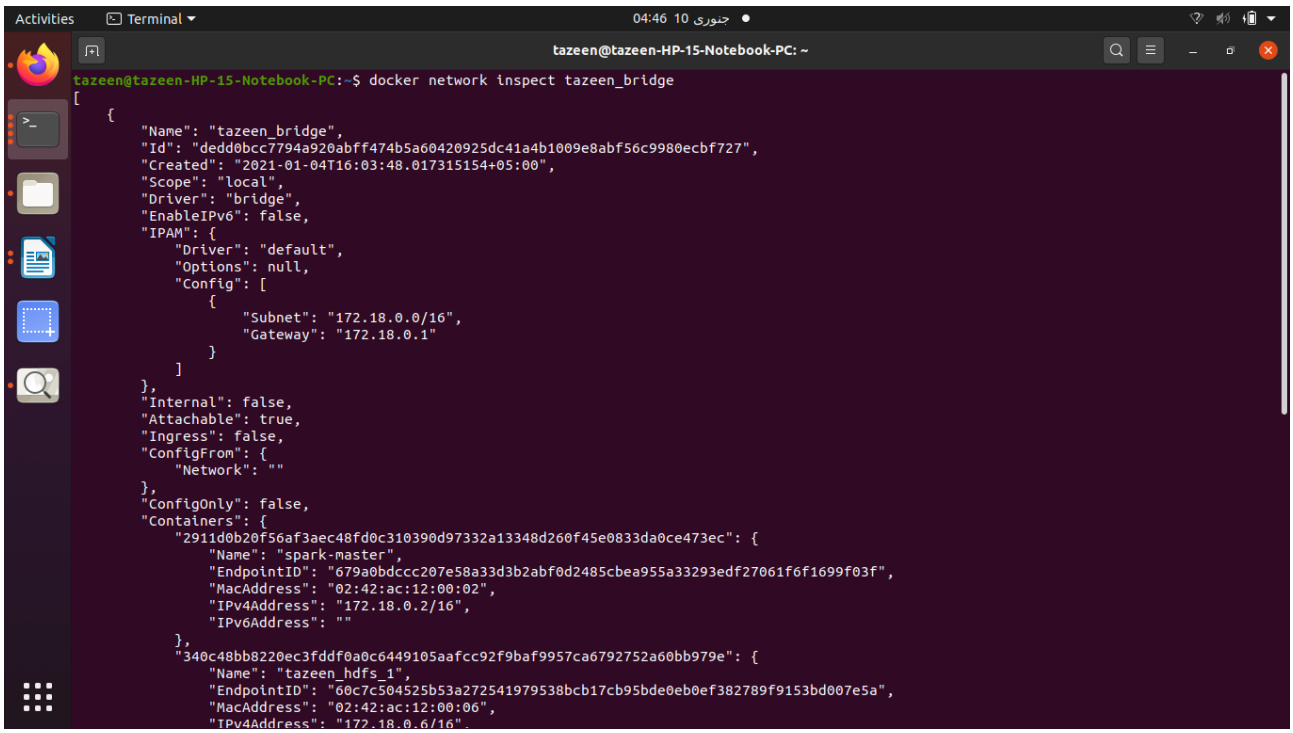


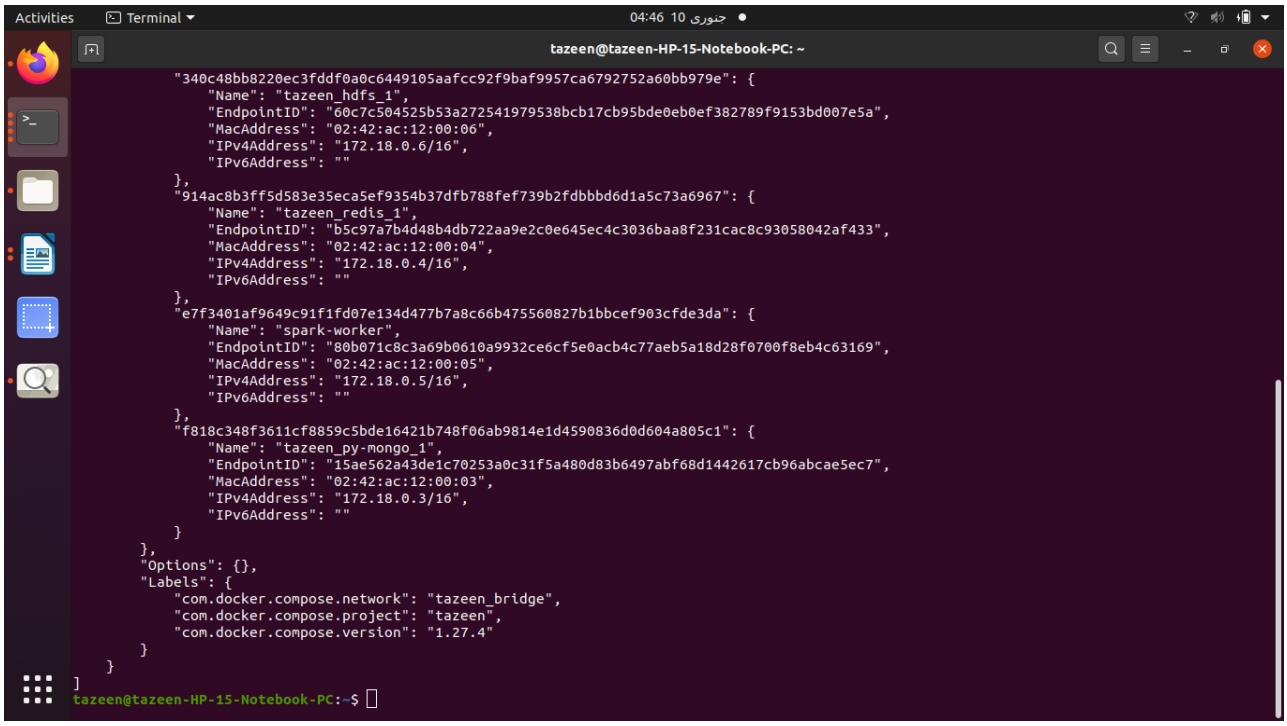
Let's execute the script to execute streaming data from redis to spark master. The dependencies required for spark is mounted on /master/lib folder in spark master. These dependencies consists of jar files required to read redis streams in pyspark.

Copying jar files to ./services/spark/dependencies folder which is mounted as /master/lib in spark master.

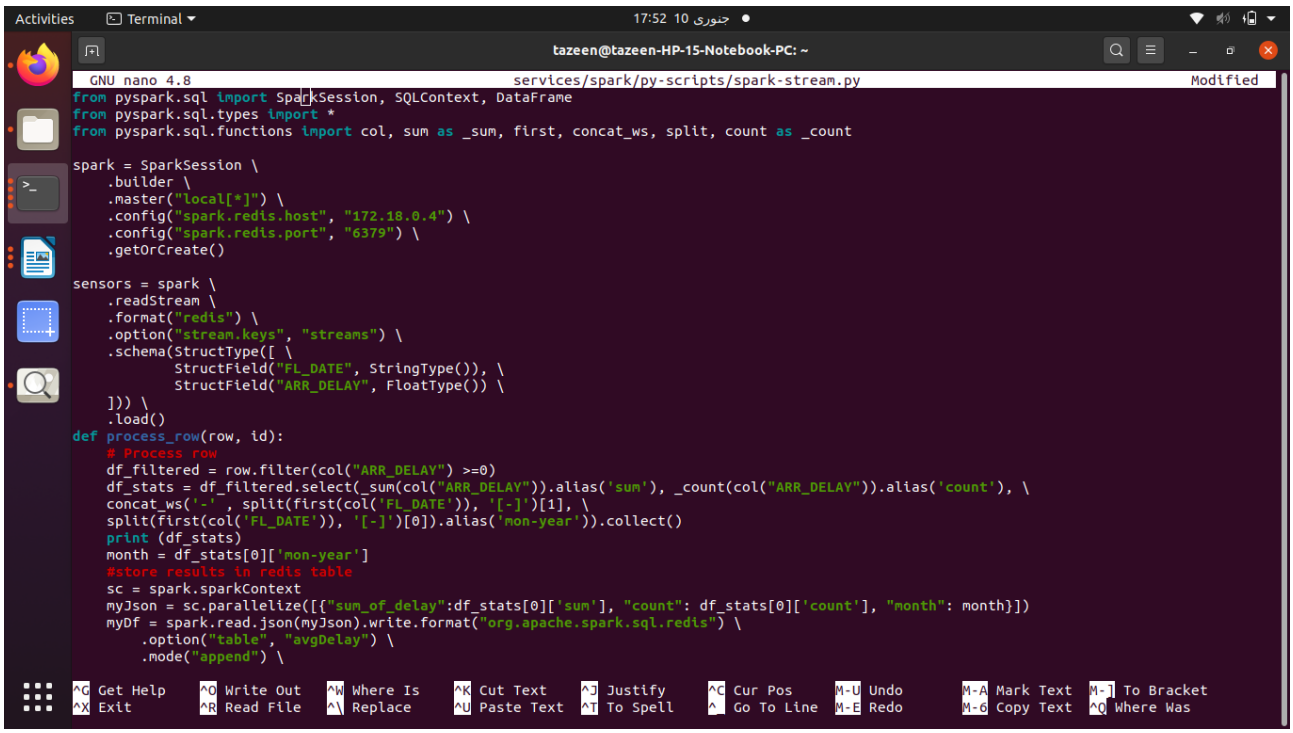


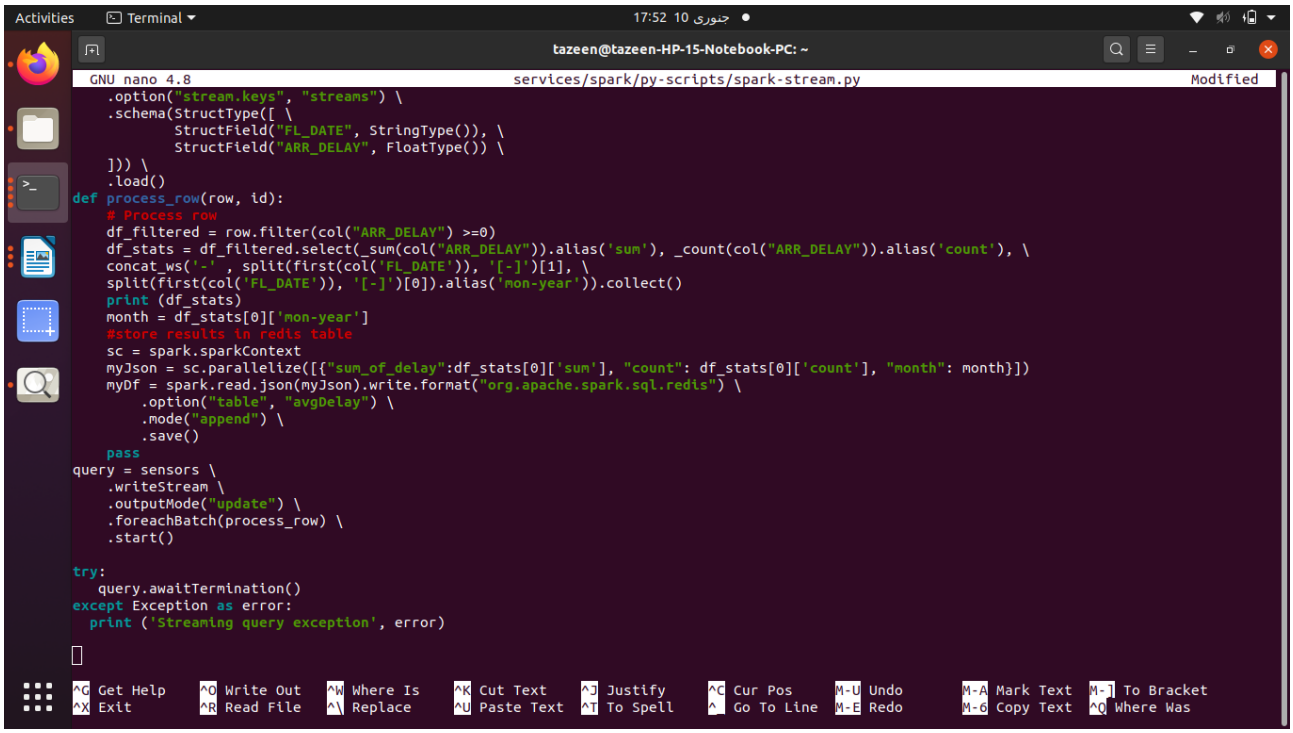
The python script to execute streaming reads streaming data from redis host, it's host ip can be seen from bridge network. For redis-docker ip is : 172.18.0.4



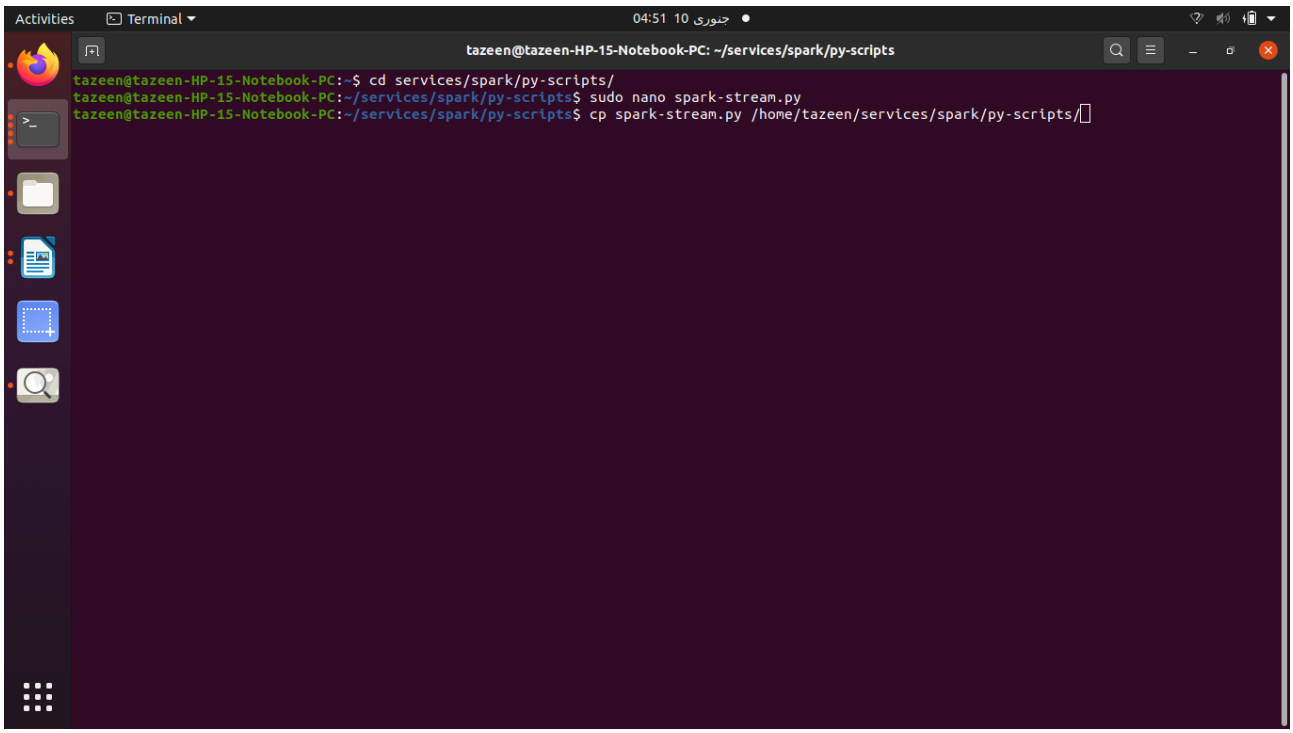


The python script is summing the delay time for positive flight delays, the number of rows provided in each stream and the month of flight date. The script is following:

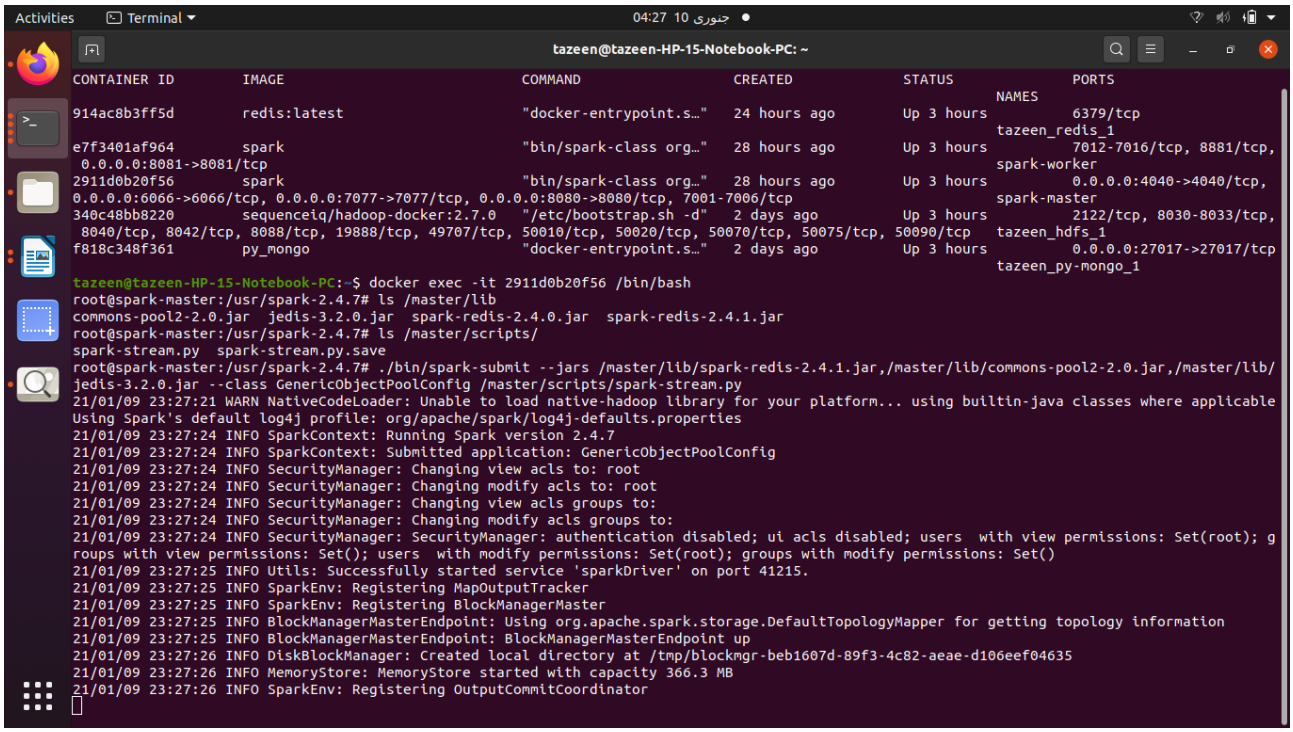




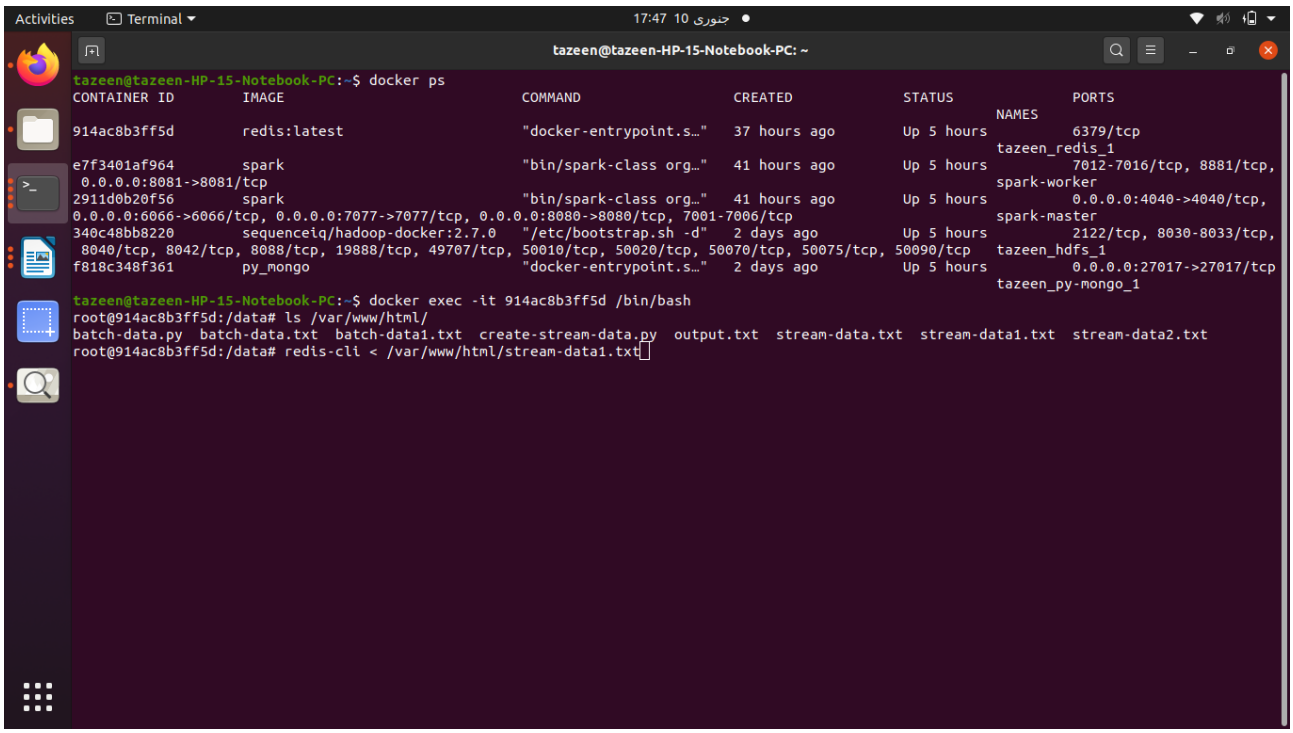
Now copying the script in ./services/spark/py-scripts/ folders which is mounted on /master/scripts folder in spark master.

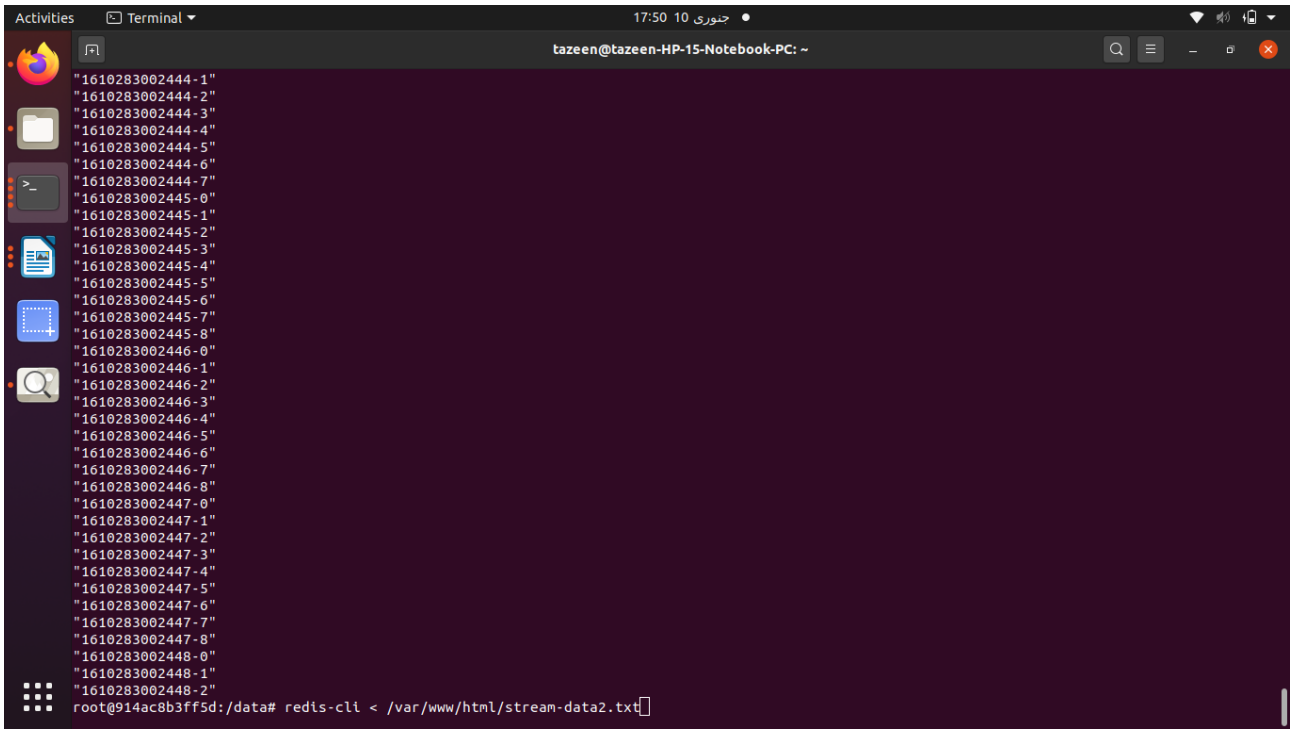


Connecting to the spark master terminal and executing the script for streaming.

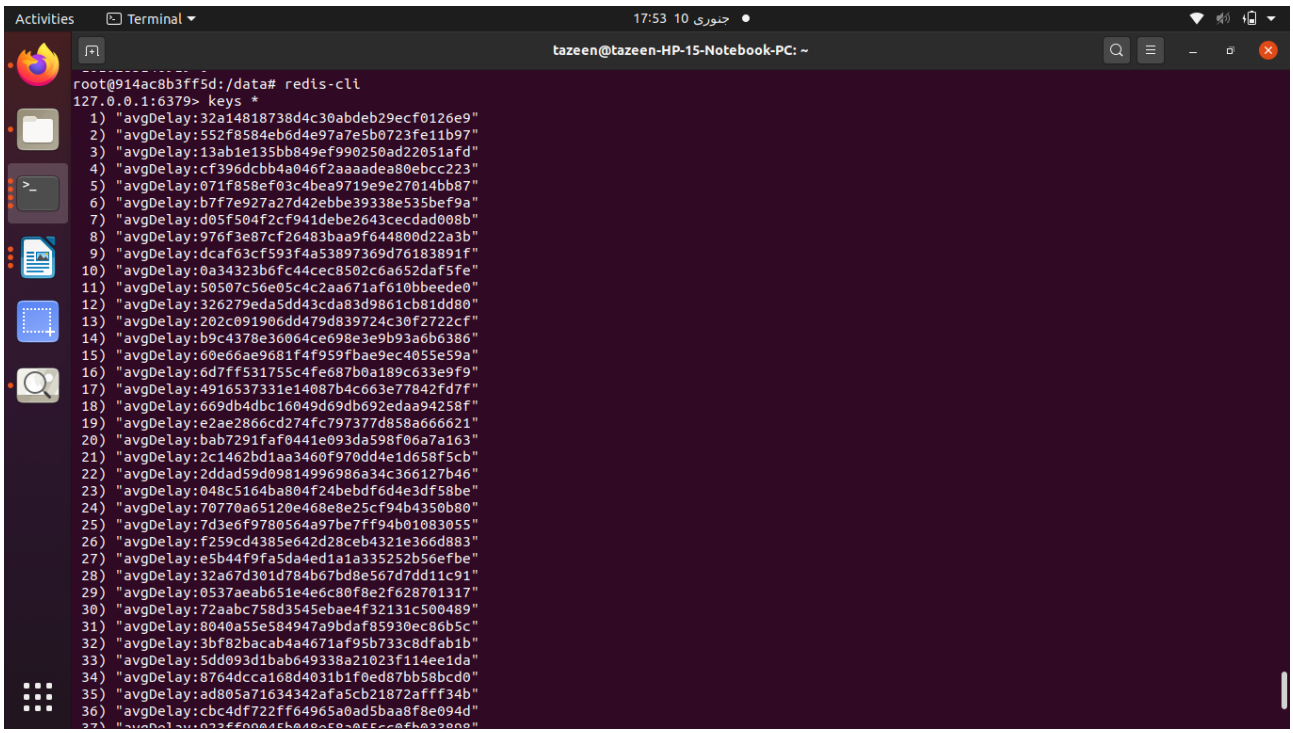


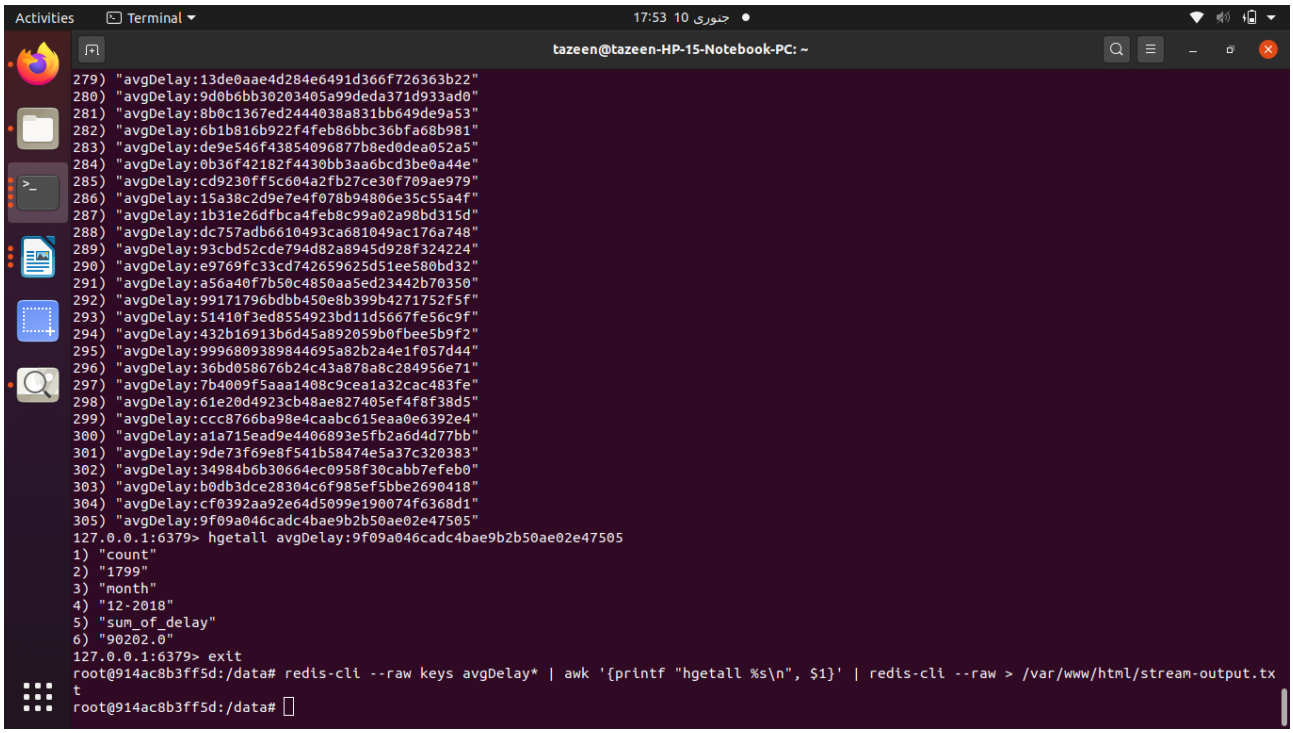
Now connecting to the redis docker container to send streams to spark. It consists of stream-data1.txt and stream-data2.txt as the redis streams for November 2018 and December 2018. The data is mounted as /var/www/html in redis docker.

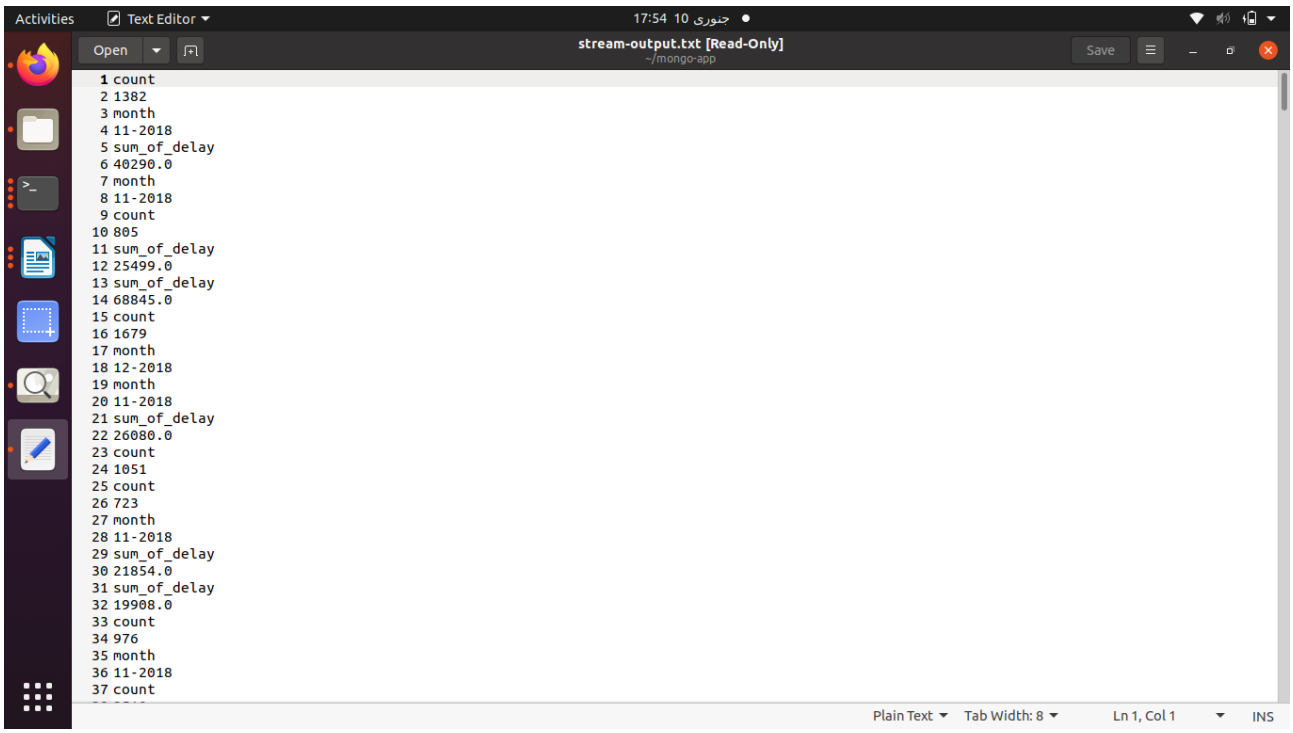




Using the cli to check the putput saved in redis table. Keys * lists all the keys created in redis. All keys starting with aygAverage are created by streaming. So saving all these results in stream-output.txt file. The output file will be saved in mongo-app folder in disk as /var/www/html is mounted for it.

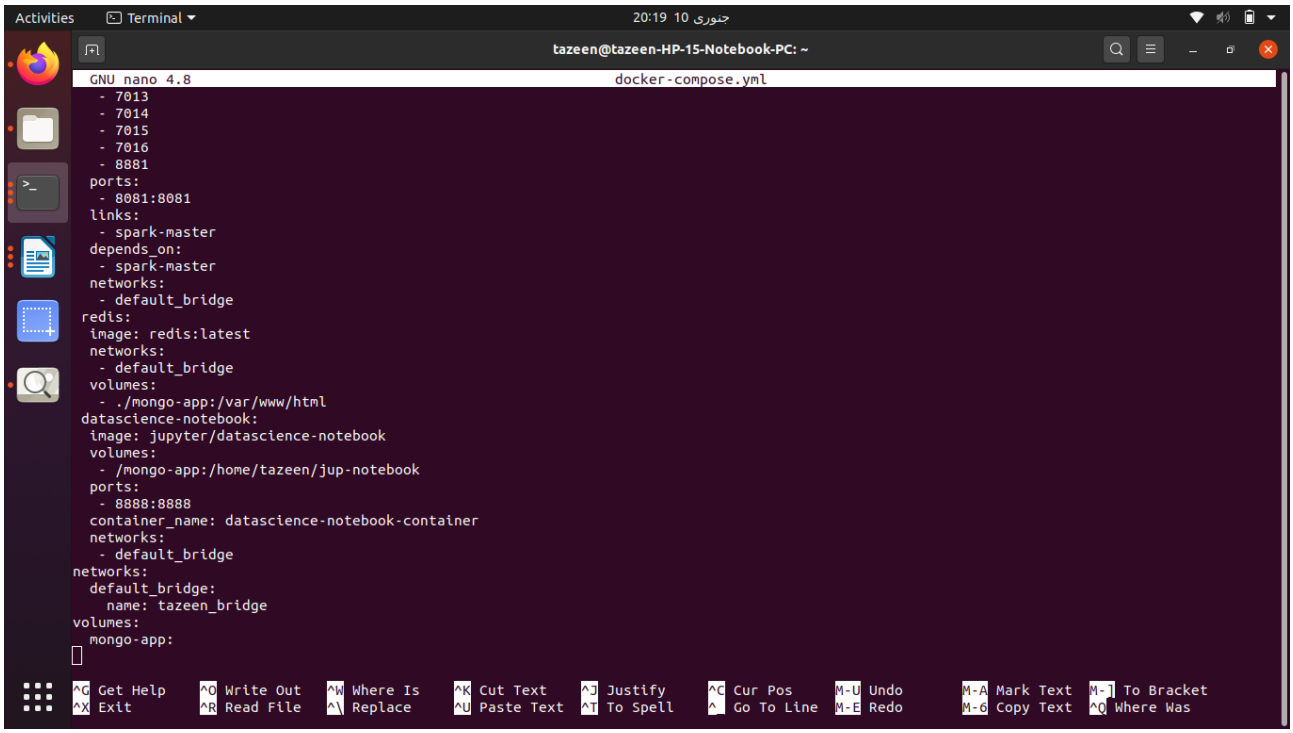

Activities
Terminal
17:53 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
root@914ac8b3ff5d:/data# redis-cli
127.0.0.1:6379> keys *
1) "avgDelay:32a14818738d4c30abdeb29ecf0126e9"
2) "avgDelay:552f8584eb6d4e97a7e5b0723fe11b97"
3) "avgDelay:13ab1e135bb849ef990250ad22051afd"
4) "avgDelay:cf396dcbb4a046f2aaaadea80ebcc223"
5) "avgDelay:071f858ef03c4bea9719e9e27014bb87"
6) "avgDelay:b7f7e927a27d42ebbe39338e535bef9a"
7) "avgDelay:d05f504f2cf941debe2643cecdad008b"
8) "avgDelay:976f3e87cf26483baa9f644800d22a3b"
9) "avgDelay:dcaf63cf593f4a53897369d76183891f"
10) "avgDelay:0a34323b6fc44cec8502c6a652daf5fe"
11) "avgDelay:50507c56e05c4c2aa671af610bbeede0"
12) "avgDelay:326279eda5dd43cda83d9861cb81dd80"
13) "avgDelay:202c091906dd479d839724c30f2722cf"
14) "avgDelay:b9c4378e36064ce698e3e9b93a6b6386"
15) "avgDelay:60e66ae9681f4f959fbae9ec4055e59a"
16) "avgDelay:6d7ff531755c4fe687b0a189c633e9f9"
17) "avgDelay:4916537331e14087b4c663e77842fd7f"
18) "avgDelay:669db4dbc16049d69db692edaa94258f"
19) "avgDelay:e2ae2866cd274fc797377d858a666621"
20) "avgDelay:bab7291faf0441e093da598f06a7a163"
21) "avgDelay:2c1462bd1aa3460f970dd4e1d658f5cb"
22) "avgDelay:2ddad59d09814996986a34c366127b46"
23) "avgDelay:048c5164ba804f24bebdf6d4e3df58be"
24) "avgDelay:70770a65120e468e8e25cf94b4350b80"
25) "avgDelay:7d3e6f9780564a97be7ff94b01083055"
26) "avgDelay:f259cd4385e642d28ceb4321e366d883"
27) "avgDelay:e5b44f9fa5da4ed1a1a335252b56efbe"
28) "avgDelay:32a67d301d784b67bd8e567d7dd11c91"
29) "avgDelay:0537aeab651e4e6c80f8e2f628701317"
30) "avgDelay:72aabc758d3545ebae4f32131c500489"
31) "avgDelay:8040a55e584947a9bdaf85930ec86b5c"
32) "avgDelay:3bf82bacab4a4671af95b733c8dfab1b"
33) "avgDelay:5dd093d1bab649338a21023f114ee1da"
34) "avgDelay:8764dcca168d4031b1f0ed87bb58bcd0"
35) "avgDelay:ad805a71634342afa5cb21872afff34b"
36) "avgDelay:cbc4df722ff64965a0ad5baa8f8e094d"


Activities
Terminal
17:53 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
279) "avgDelay:13de0aae4d284e6491d366f726363b22"
280) "avgDelay:9d0b6bb30203405a99deda371d933ad0"
281) "avgDelay:8b0c1367ed2444038a831bb649de9a53"
282) "avgDelay:6b1b816b922f4feb86bbc36bfa68b981"
283) "avgDelay:de9e546f43854096877b8ed0dea052a5"
284) "avgDelay:0b36f42182f4430bb3aa6bcd3be0a44e"
285) "avgDelay:cd9230ff5c604a2fb27ce30f709ae979"
286) "avgDelay:15a38c2d9e7e4f078b94806e35c55a4f"
287) "avgDelay:1b31e26dfbca4feb8c99a02a98bd315d"
288) "avgDelay:dc757adb6610493ca681049ac176a748"
289) "avgDelay:93cbd52cde794d82a8945d928f324224"
290) "avgDelay:e9769fc33cd742659625d51ee580bd32"
291) "avgDelay:a56a40f7b50c4850aa5ed23442b70350"
292) "avgDelay:99171796bdbb450e8b399b4271752f5f"
293) "avgDelay:51410f3ed8554923bd11d5667fe56c9f"
294) "avgDelay:432b16913b6d45a892059b0fbee5b9f2"
295) "avgDelay:9996809389844695a82b2a4e1f057d44"
296) "avgDelay:36bd058676b24c43a878a8c284956e71"
297) "avgDelay:7b4009f5aaa1408c9cea1a32cac483fe"
298) "avgDelay:61e20d4923cb48ae827405ef4f8f38d5"
299) "avgDelay:ccc8766ba98e4caabc615eaa0e6392e4"
300) "avgDelay:a1a715ead9e4406893e5fb2a6d4d77bb"
301) "avgDelay:9de73f69e8f541b58474e5a37c320383"
302) "avgDelay:34984b6b30664ec0958f30cabb7efeb0"
303) "avgDelay:b0db3dce28304c6f985ef5bbe2690418"
304) "avgDelay:cf0392aa92e64d5099e190074f6368d1"
305) "avgDelay:9f09a046cadc4bae9b2b50ae02e47505"
127.0.0.1:6379> hgetall avgDelay:9f09a046cadc4bae9b2b50ae02e47505
1) "count"
2) "1799"
3) "month"
4) "12-2018"
5) "sum_of_delay"
6) "90202.0"
127.0.0.1:6379> exit
root@914ac8b3ff5d:/data# redis-cli --raw keys avgDelay* | awk '{printf "hgetall %s\n", $1}' | redis-cli --raw > /var/www/html/stream-output.tx
t
root@914ac8b3ff5d:/data#

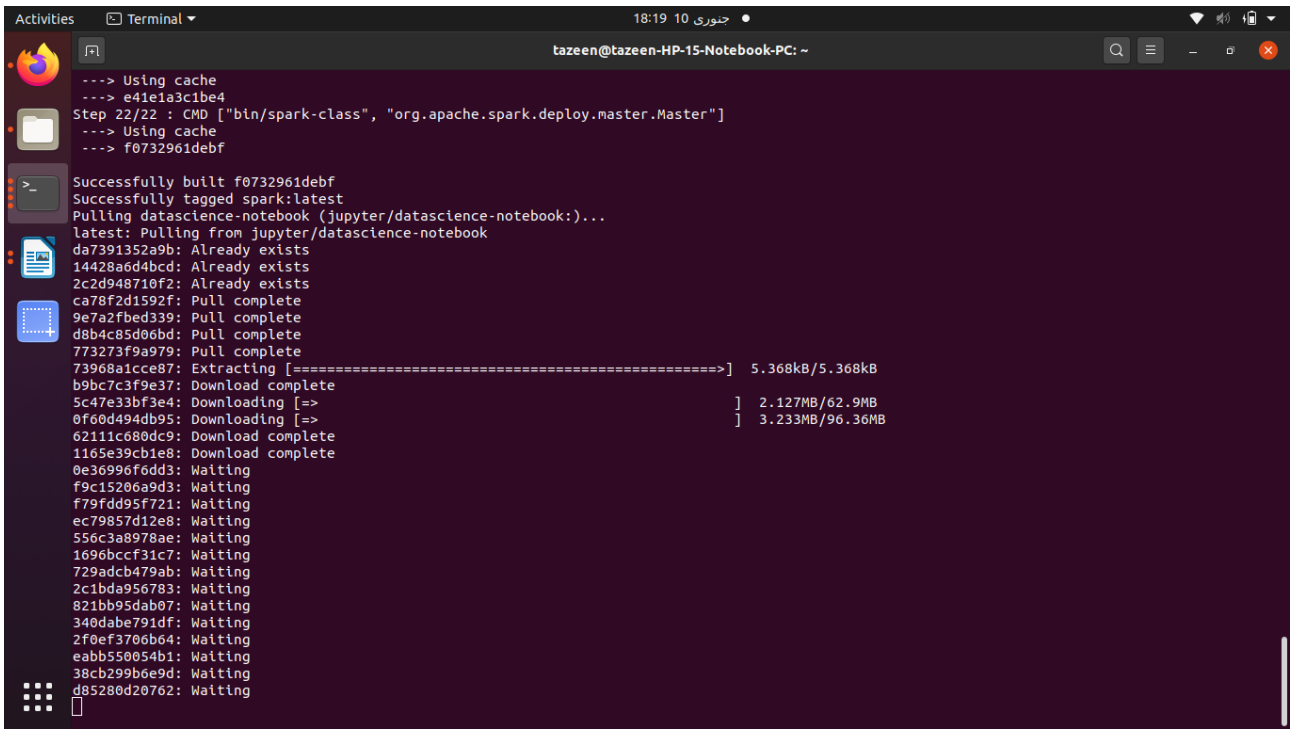


4. Serving Layer

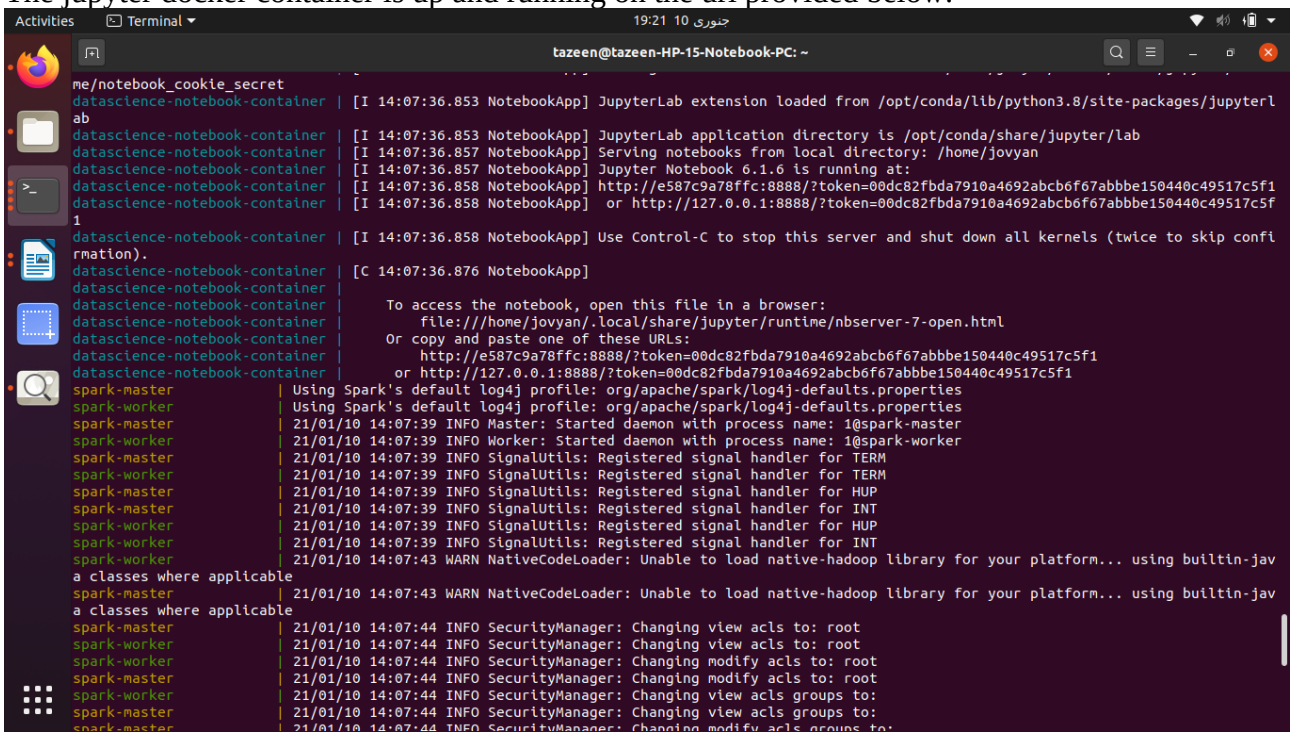
The last step is to create the serving layer which will show the results from batch and stream layers. To create a serving layer, jupyter notebook image is used in container. The updated docker-compose.yml is adds the jupyter image as below:



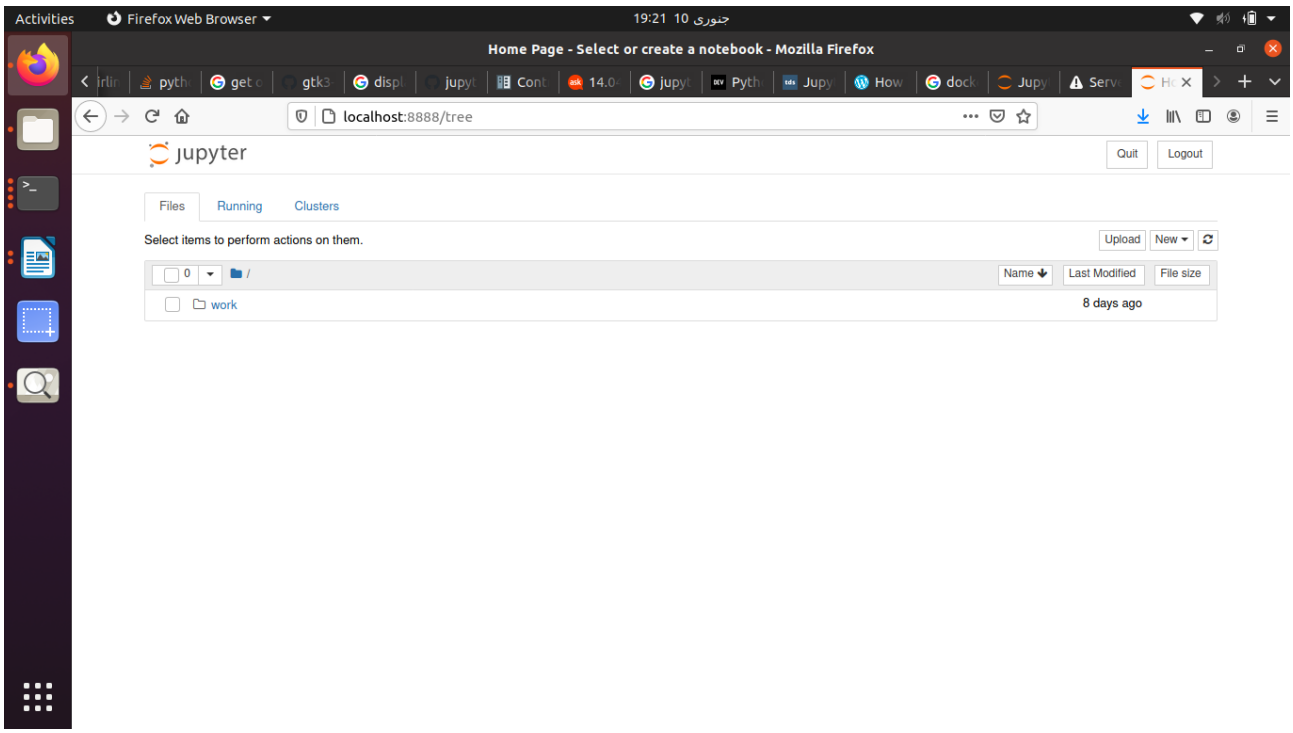
Now executing docker-compose –build



The jupyter docker container is up and running on the url provided below:

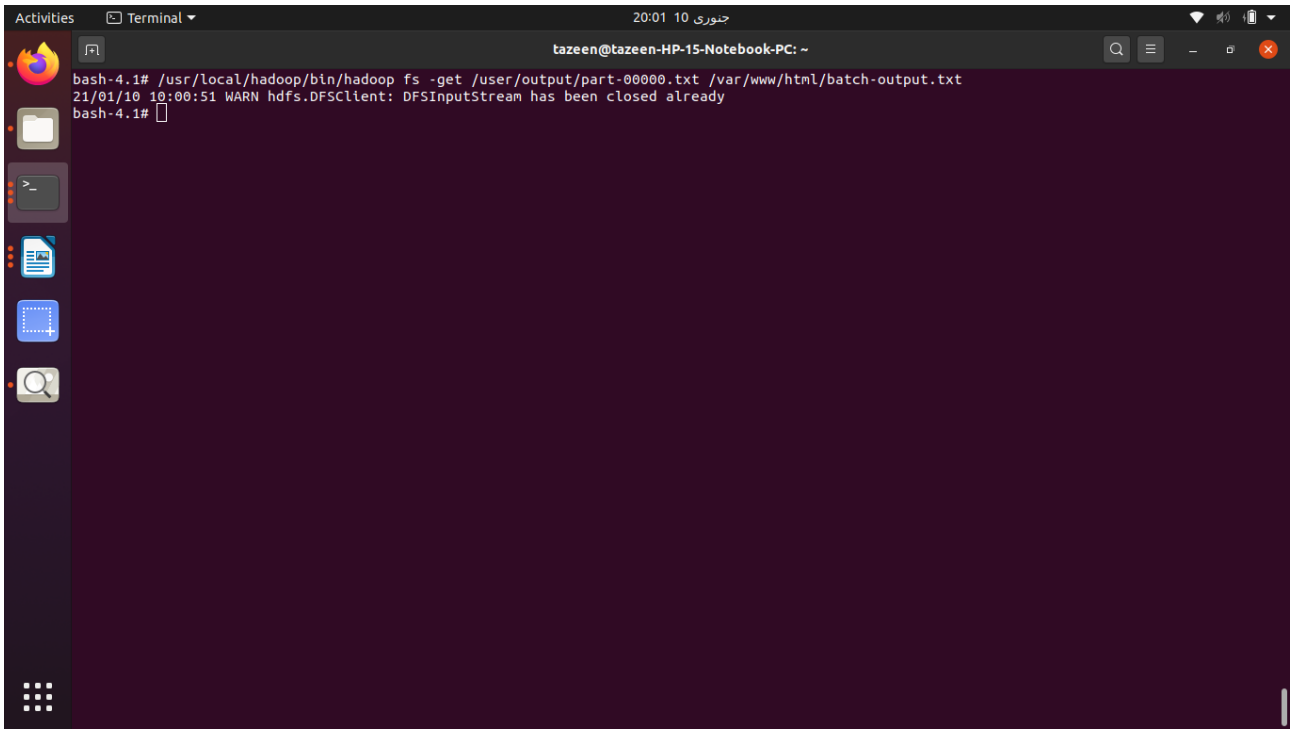


Navigating to the jupyter notebook on the url provided.

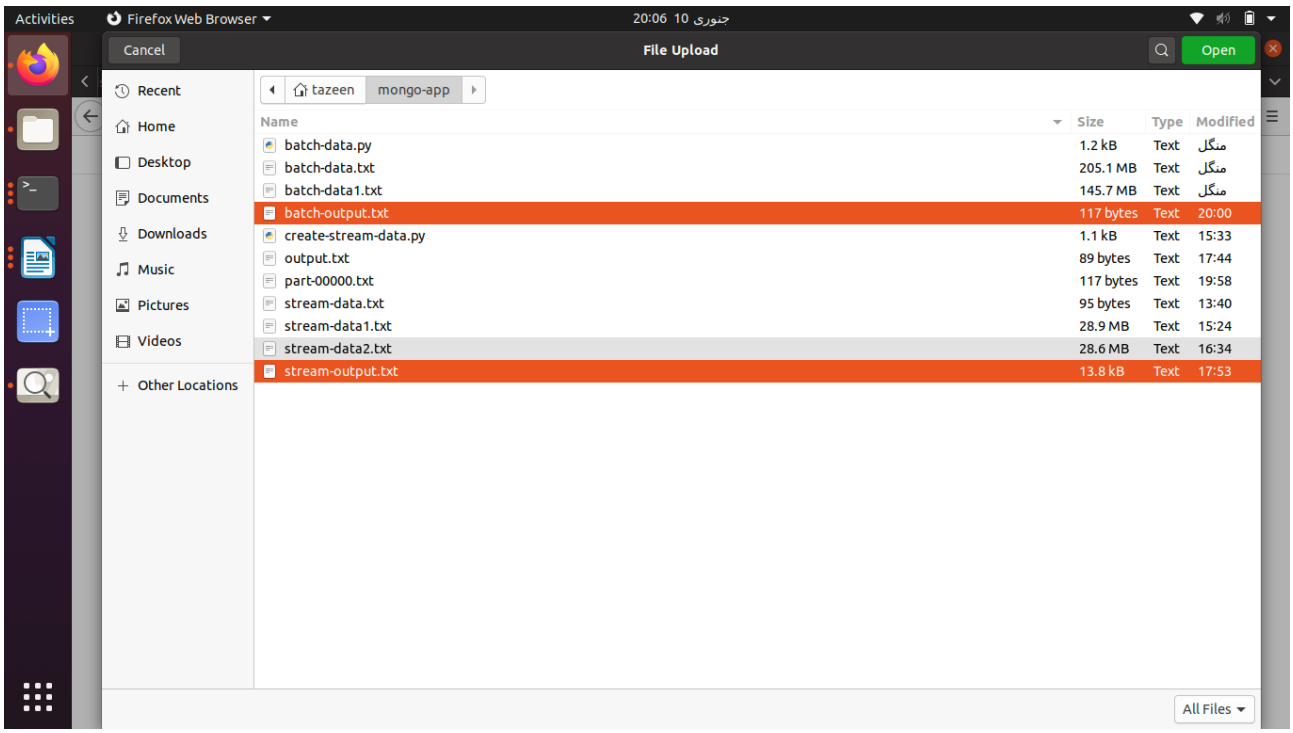


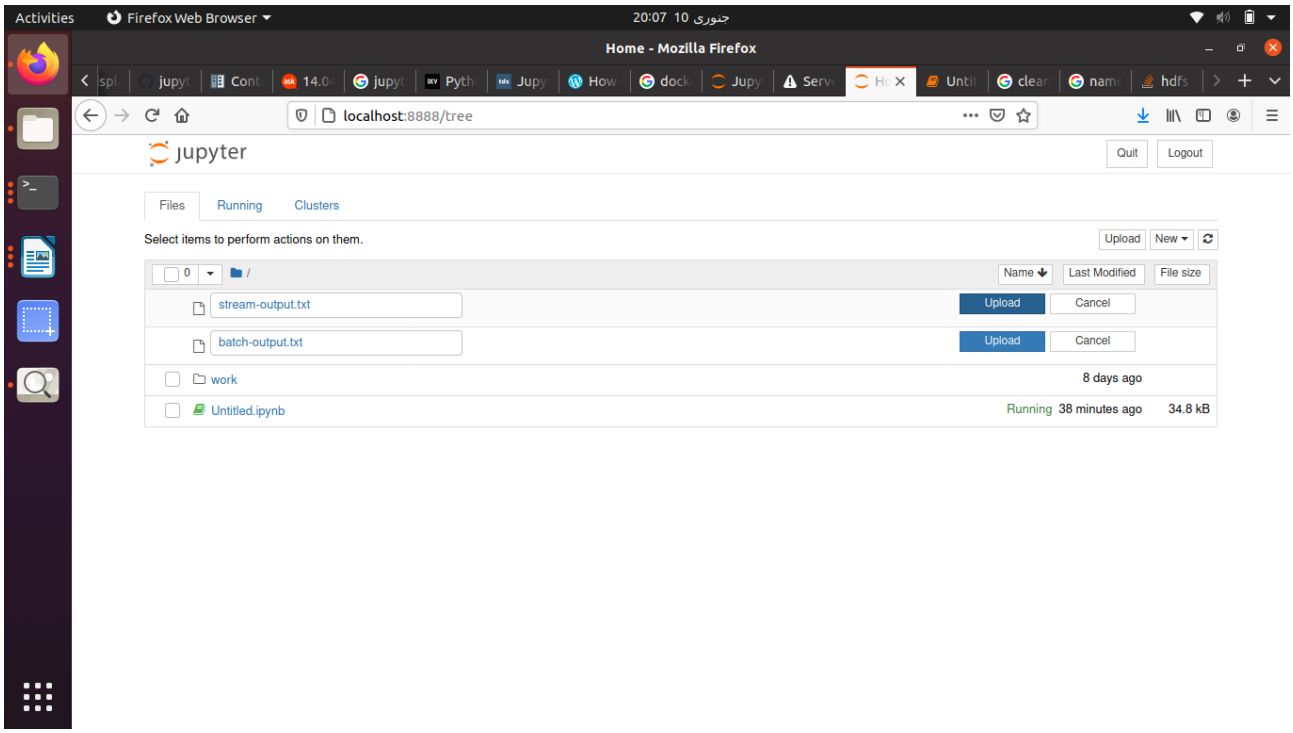
Now uploading output files in jupyter. These files are save in mongo-app folder which is mounted in the jupyter container.

First saving HDFS batch output file in mongo-app folder by:

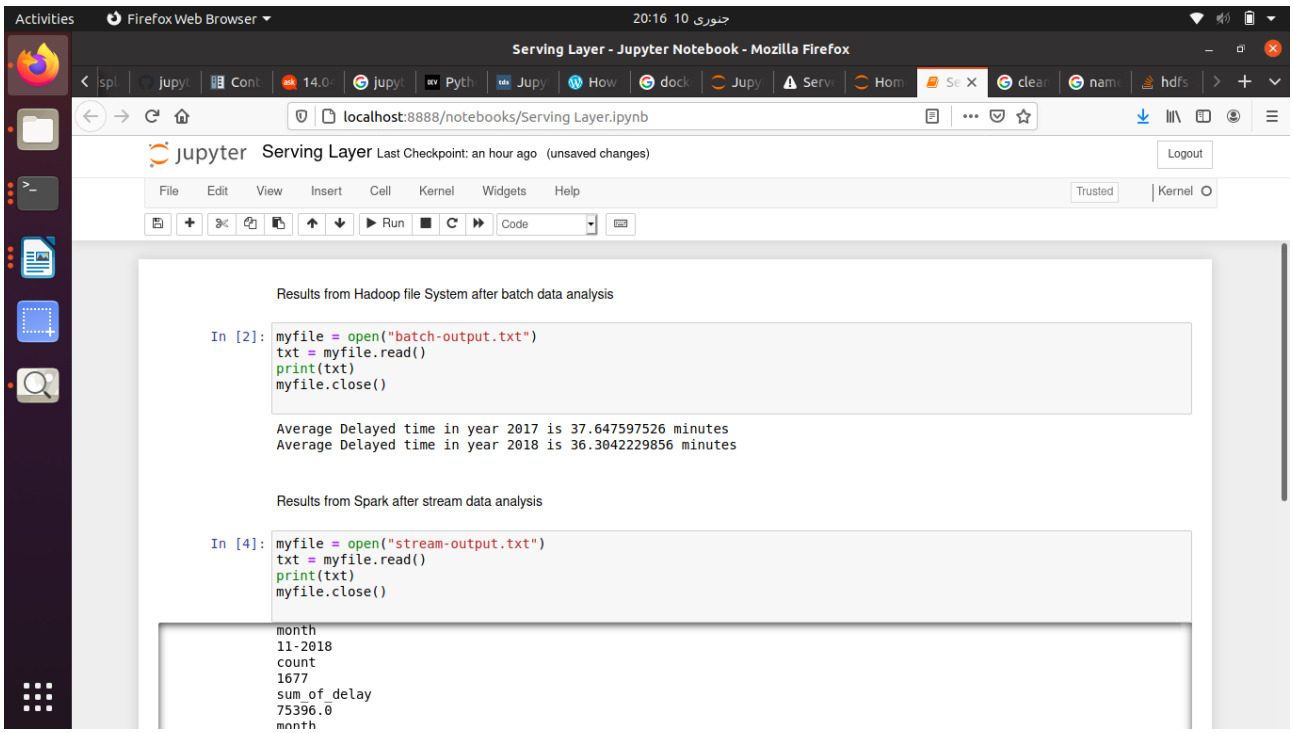


The stream-output file in already saved in mongo-app folder as previously shown. Now uploading these files in jupyter notebook.





Finally creating new notebook to display results from the batch and stream layers.


Activities
Firefox Web Browser
20:16 10 جنوري
Serving Layer - Jupyter Notebook - Mozilla Firefox
localhost:8888/notebooks/Serving Layer.ipynb
jupyter Serving Layer Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Kernel
Run
Code
Results from Hadoop file System after batch data analysis
In [2]:
myfile = open("batch-output.txt")
txt = myfile.read()
print(txt)
myfile.close()
Average Delayed time in year 2017 is 37.647597526 minutes
Average Delayed time in year 2018 is 36.3042229856 minutes
Results from Spark after stream data analysis
In [4]:
myfile = open("stream-output.txt")
txt = myfile.read()
print(txt)
myfile.close()
month
11-2018
count
1677
sum_of_delay
75396.0
month