Lambda Architecture on Airline Data:

1. **Understanding of Data:**

   I am using airline cancellation and delay data over different years. The dataset contains details of the flights which were canceled or delayed. Also, it maintains the arrival and destination time delay in minutes. It can be downloaded from Kaggle (Airline Delay and Cancellation Data).

   EDA on data using jupyter notebook:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

EDA on 2017 airline data

```
In [2]: #reading the csv file
        df1 = pd.read_csv('2017.csv');
        print (df1.shape)
        df1.head(10)
```

(5674621, 28)

Out[2]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAPSEI
0	2017-01-01	AA	1	JFK	LAX	800	831.0	31.0	25.0	856.0	...	
1	2017-01-01	AA	2	LAX	JFK	900	934.0	34.0	34.0	1008.0	...	
2	2017-01-01	AA	4	LAX	JFK	1130	1221.0	51.0	20.0	1241.0	...	
3	2017-01-01	AA	5	DFW	HNL	1135	1252.0	77.0	19.0	1311.0	...	
4	2017-01-01	AA	6	OGG	DFW	1855	1855.0	0.0	16.0	1911.0	...	
5	2017-01-01	AA	7	DFW	OGG	940	1619.0	399.0	12.0	1631.0	...	

```
In [3]: df1.tail(10)
```

Out[3]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAP
5674611	2017-12-31	UA	2415	EWR	PDX	825	850.0	25.0	14.0	904.0	...	
5674612	2017-12-31	UA	2417	PDX	EWR	2240	2238.0	-2.0	10.0	2248.0	...	
5674613	2017-12-31	UA	2418	RIC	DEN	1601	1600.0	-1.0	14.0	1614.0	...	

In [4]:
```
# Total number of canceled flights
df1.CANCELLED.sum()
```

Out[4]:
```
82693.0
```

In [5]:
```
# Let's explore column CANCELLED
df1.CANCELLED.unique()
```

Out[5]:
```
array([0., 1.])
```

In [6]:
```
# From above we see it's binary: 0 or 1, Let's see how it looks like
canceled = df1[(df1['CANCELLED'] > 0)]
```

In [7]:
```
canceled.head(3)
```

Out[7]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAPSED_
80	2017-01-01	AA	106	PHX	JFK	957	NaN	NaN	NaN	NaN	...	
151	2017-01-01	AA	229	DFW	KOA	1030	NaN	NaN	NaN	NaN	...	
152	2017-01-01	AA	230	KOA	DFW	1840	NaN	NaN	NaN	NaN	...	

3 rows × 28 columns

In [8]:
```
# Arrival delay data (in minutes)
df1.ARR_DELAY.head()
```

Out[8]:
```
0    27.0
1    42.0
2    42.0
3    97.0
4    42.0
Name: ARR_DELAY, dtype: float64
```

In [9]:
```
# Check if FL_DATE is DateTime type
type(df1['FL_DATE'])
```

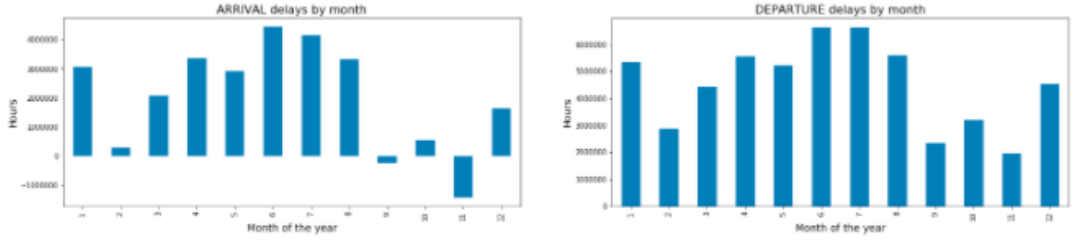
Out[9]:
```
pandas.core.series.Series
```

In [10]:
```
# Convert string to DateTime
pd.to_datetime(df1.FL_DATE)
```

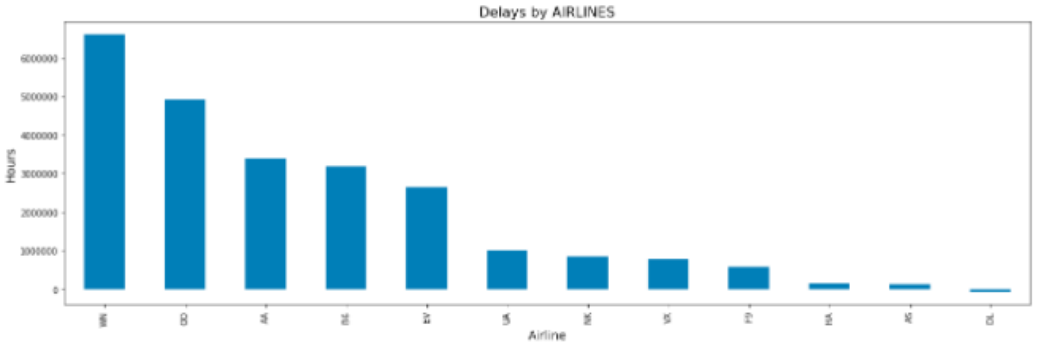
Out[10]:
```
0         2017-01-01
1         2017-01-01
2         2017-01-01
3         2017-01-01
4         2017-01-01
             ...
5674616   2017-12-31
5674617   2017-12-31
5674618   2017-12-31
5674619   2017-12-31
5674620   2017-12-31
Name: FL_DATE, Length: 5674621, dtype: datetime64[ns]
```

In [11]:
```
# Month variable
df1['FL_DATE_month'] = pd.to_datetime(df1['FL_DATE']).dt.month
```

In [12]:
```
# Arrival and departure delays by month of the year
plt.figure(figsize=(25, 12)).subplots_adjust(hspace = 0.5)

plt.subplot(2, 2 ,1)
df1.groupby('FL_DATE_month').ARR_DELAY.sum().plot.bar().set_title('ARRIVAL delays by month')
plt.title('ARRIVAL delays by month', fontsize=16)
plt.ylabel('Hours', fontsize=14)
plt.xlabel('Month of the year', fontsize=14)

plt.subplot(2, 2 ,2)
df1.groupby('FL_DATE_month').DEP_DELAY.sum().plot.bar()
plt.title('DEPARTURE delays by month', fontsize=16)
plt.ylabel('Hours', fontsize=14)
plt.xlabel('Month of the year', fontsize=14)

plt.show()
```

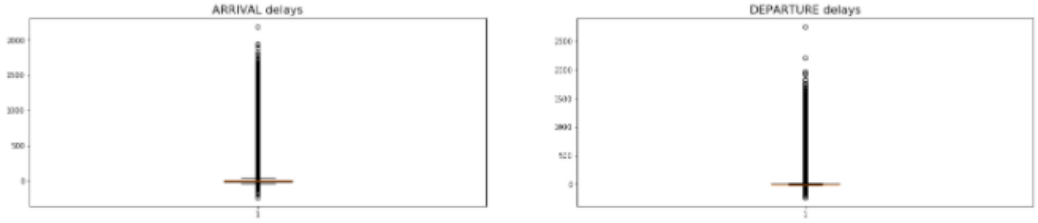


In [13]:
```
# Delays by airlines
plt.figure(figsize=(20, 6))
df1.groupby('OP_CARRIER').ARR_DELAY.sum().sort_values(ascending=False).plot.bar()
plt.title('Delays by AIRLINES', fontsize=16)
plt.xlabel('Airline', fontsize=14)
plt.ylabel('Hours', fontsize=14)
plt.show()
```

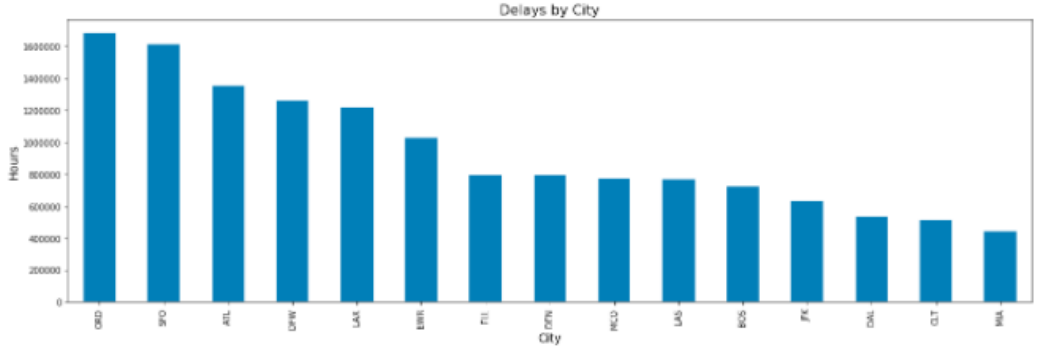


In [14]:
```
# Arrival and departure delays by month of the year
plt.figure(figsize=(25, 12)).subplots_adjust(hspace = 0.5)

plt.subplot(2, 2 ,1)
plt.boxplot(df1.ARR_DELAY.dropna())
plt.title('ARRIVAL delays', fontsize=16)

plt.subplot(2, 2 ,2)
plt.boxplot(df1.DEP_DELAY.dropna())
plt.title('DEPARTURE delays', fontsize=16)

plt.show()
```


```
plt.subplot(2, 2 ,2)
plt.boxplot(df1.DEP_DELAY.dropna())
plt.title('DEPARTURE delays', fontsize=16)

plt.show()
```

In [15]:
```
# Delays by City
city_by_delay = df1.groupby('ORIGIN').ARR_DELAY.sum().sort_values(ascending=False)
plt.figure(figsize=(20, 6))
city_by_delay[:15].plot.bar()
plt.title('Delays by City', fontsize=16)
plt.xlabel('City', fontsize=14)
plt.ylabel('Hours', fontsize=14)
plt.show()
```



EDA on 2018 airline data

In [16]:
```
#reading the csv file
df1 = pd.read_csv('2018.csv');
print (df1.shape)
df1.head(10)
```

```
(7213446, 28)
```

Out[16]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAPSEI
0	2018-01-01	UA	2429	EWR	DEN	1517	1512.0	-5.0	15.0	1527.0	...	
1	2018-01-01	UA	2427	LAS	SFO	1115	1107.0	-8.0	11.0	1118.0	...	
2	2018-01-01	UA	2426	SNA	DEN	1335	1330.0	-5.0	15.0	1345.0	...	
3	2018-01-01	UA	2425	RSW	ORD	1546	1552.0	6.0	19.0	1611.0	...	
4	2018-01-01	UA	2424	ORD	ALB	630	650.0	20.0	13.0	703.0	...	
5	2018-01-01	UA	2422	ORD	OMA	2241	2244.0	3.0	15.0	2259.0	...	

In [17]:
```
df1.tail(10)
```

Out[17]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAP
7213436	2018-12-31	AA	1812	PDX	PHX	1058	1100.0	2.0	17.0	1117.0	...	
7213437	2018-12-31	AA	1812	PHX	PDX	825	821.0	-4.0	15.0	836.0	...	
7213438	2018-12-31	AA	1813	CLT	ATL	2100	2100.0	0.0	12.0	2112.0	...	
7213439	2018-12-31	AA	1814	DFW	PHL	1955	2026.0	31.0	12.0	2038.0	...	
7213440	2018-12-31	AA	1815	CLT	DCA	1321	1320.0	-1.0	12.0	1332.0	...	
7213441	2018-12-31	AA	1815	DCA	CLT	1534	1530.0	-4.0	20.0	1550.0	...	
7213442	2018-12-31	AA	1816	CLT	DFW	1751	1757.0	6.0	18.0	1815.0	...	

```
df1.CANCELLED.sum()
```

Out[18]:
```
116584.0
```

In [19]:
```
# Let's explore column CANCELLED
df1.CANCELLED.unique()
```

Out[19]:
```
array([0., 1.])
```

In [20]:
```
# From above we see it's binary: 0 or 1, Let's see how it Looks like
canceled = df1[(df1['CANCELLED'] > 0)]
```

In [21]:
```
canceled.head(3)
```

Out[21]:

	FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	...	CRS_ELAPSED
178	2018-01-01	UA	2034	IAH	MFE	1440	NaN	NaN	NaN	NaN	...	
875	2018-01-01	UA	864	LAS	SFO	1744	NaN	NaN	NaN	NaN	...	
1244	2018-01-01	UA	488	MFE	IAH	1726	NaN	NaN	NaN	NaN	...	

3 rows × 28 columns

In [22]:
```
# Arrival delay data (in minutes)
df1.ARR_DELAY.head()
```

Out[22]:
```
0   -23.0
1   -24.0
2   -13.0
3    -2.0
4    14.0
Name: ARR_DELAY, dtype: float64
```

In [23]:
```
# Check if FL_DATE is DateTime type
type(df1['FL_DATE'])
```

Out[23]:
```
pandas.core.series.Series
```

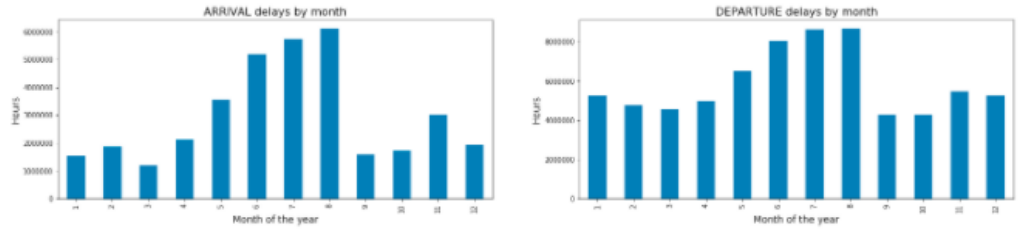
In [24]:
```
# Convert string to DateTime
pd.to_datetime(df1.FL_DATE)
```

```
pd.to_datetime(df1.FL_DATE)
```

Out[24]:
```
0          2018-01-01
1          2018-01-01
2          2018-01-01
3          2018-01-01
4          2018-01-01
              ...
7213441    2018-12-31
7213442    2018-12-31
7213443    2018-12-31
7213444    2018-12-31
7213445    2018-12-31
Name: FL_DATE, Length: 7213446, dtype: datetime64[ns]
```

In [25]:
```
# Month variable
df1['FL_DATE_month'] = pd.to_datetime(df1['FL_DATE']).dt.month
```

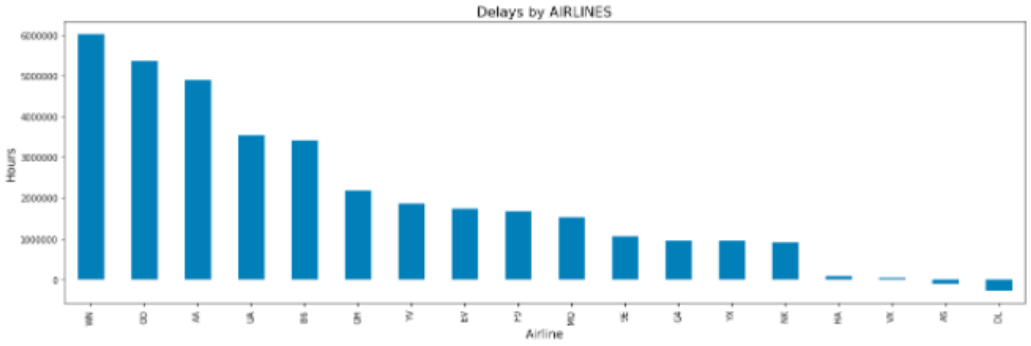
In [26]:
```
# Arrival and departure delays by month of the year
plt.figure(figsize=(25, 12)).subplots_adjust(hspace = 0.5)

plt.subplot(2, 2 ,1)
df1.groupby('FL_DATE_month').ARR_DELAY.sum().plot.bar().set_title('ARRIVAL delays by month')
plt.title('ARRIVAL delays by month', fontsize=16)
plt.ylabel('Hours', fontsize=14)
plt.xlabel('Month of the year', fontsize=14)

plt.subplot(2, 2 ,2)
df1.groupby('FL_DATE_month').DEP_DELAY.sum().plot.bar()
plt.title('DEPARTURE delays by month', fontsize=16)
plt.ylabel('Hours', fontsize=14)
plt.xlabel('Month of the year', fontsize=14)

plt.show()
```



Month of the year

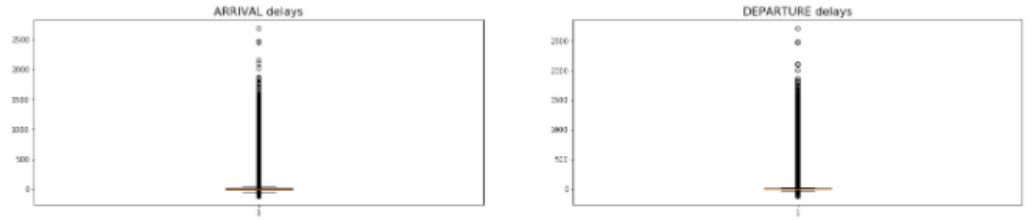
Month of the year

In [27]:
```
# Delays by airlines
plt.figure(figsize=(20, 6))
df1.groupby('OP_CARRIER').ARR_DELAY.sum().sort_values(ascending=False).plot.bar()
plt.title('Delays by AIRLINES', fontsize=16)
plt.xlabel('Airline', fontsize=14)
plt.ylabel('Hours', fontsize=14)
plt.show()
```

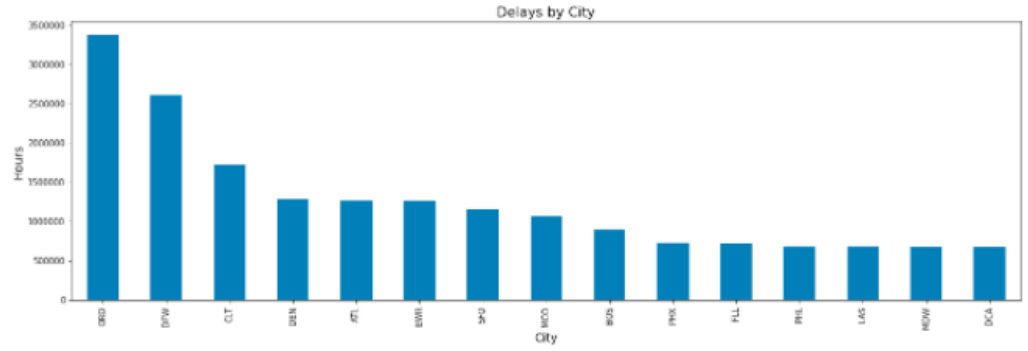


In [28]:
```
# Arrival and departure delays by month of the year
plt.figure(figsize=(25, 12)).subplots_adjust(hspace = 0.5)

plt.subplot(2, 2 ,1)
plt.boxplot(df1.ARR_DELAY.dropna())
plt.title('ARRIVAL delays', fontsize=16)

plt.subplot(2, 2 ,2)
plt.boxplot(df1.DEP_DELAY.dropna())
plt.title('DEPARTURE delays', fontsize=16)

plt.show()
```

```
plt.subplot(2, 2 ,2)
plt.boxplot(df1.DEP_DELAY.dropna())
plt.title('DEPARTURE delays', fontsize=16)

plt.show()
```




```
In [29]: # Delays by City
city_by_delay = df1.groupby('ORIGIN').ARR_DELAY.sum().sort_values(ascending=False)
plt.figure(figsize=(20, 6))
city_by_delay[:15].plot.bar()
plt.title('Delays by City', fontsize=16)
plt.xlabel('City', fontsize=14)
plt.ylabel('Hours', fontsize=14)
plt.show()
```

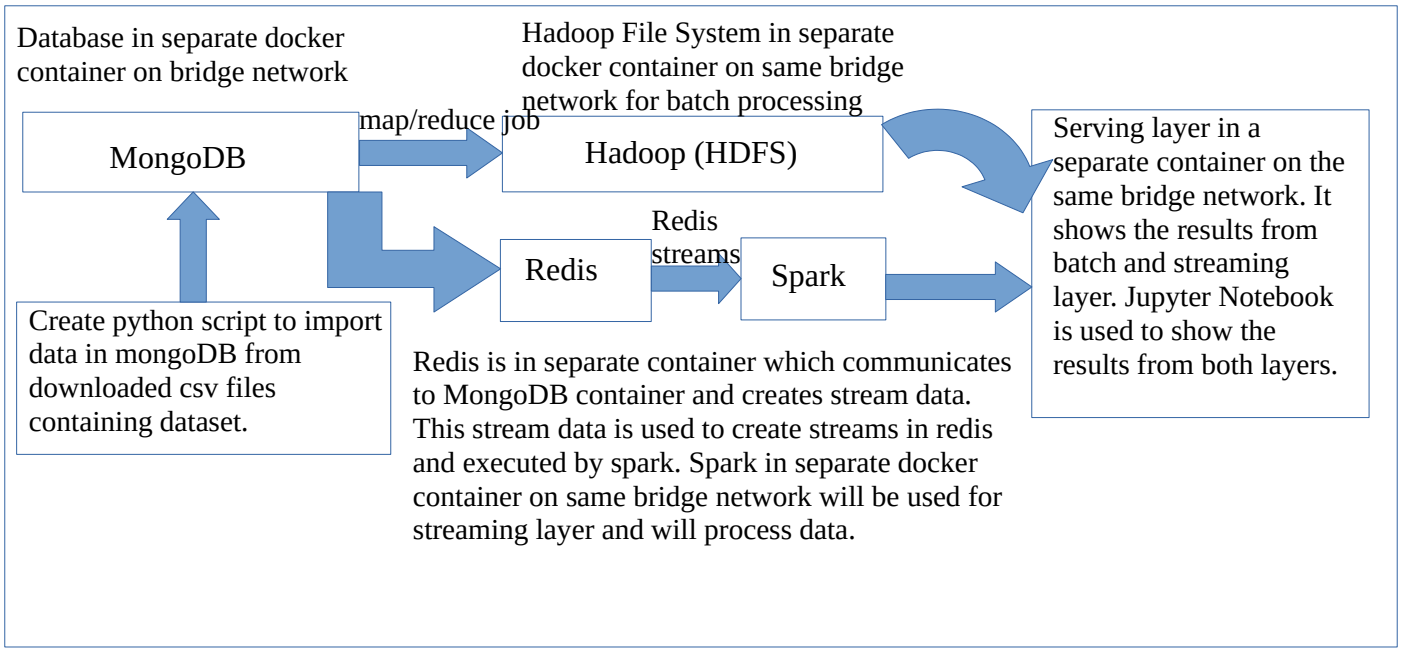


2. Domain Knowledge

For the batch processing, I am using airline data for years 2017 and 2018. The data exists in tabular form which is around 1.2 GB. I will be executing a hadoop map/reduce job to calculate the average delay time in flights over the years. Firstly, the delay time can be analyzed by filtering the canceled flights. That is; removing the rows with ‘CANCELED’ column as 1. Next, a delayed flight can be defined as arriving late to the destination so the ‘ARR_DELAY’ column value will be positive and greater than 0.
For streaming layer, spark will be used which will take data streams from redis. It will execute a script to calculate average flights delay in a month. Each data stream from redis will be for each month and spark will read it execute the script. The output is save to redis table which consists of month and the average delay. The script is executed with the above analysis that a delay flight will have ‘CANCELLED’ value 0 and positive delay values are considered for average.
The serving layer will show the results from batch and streaming layer.

3. **Methodology**

Architecture Diagram:



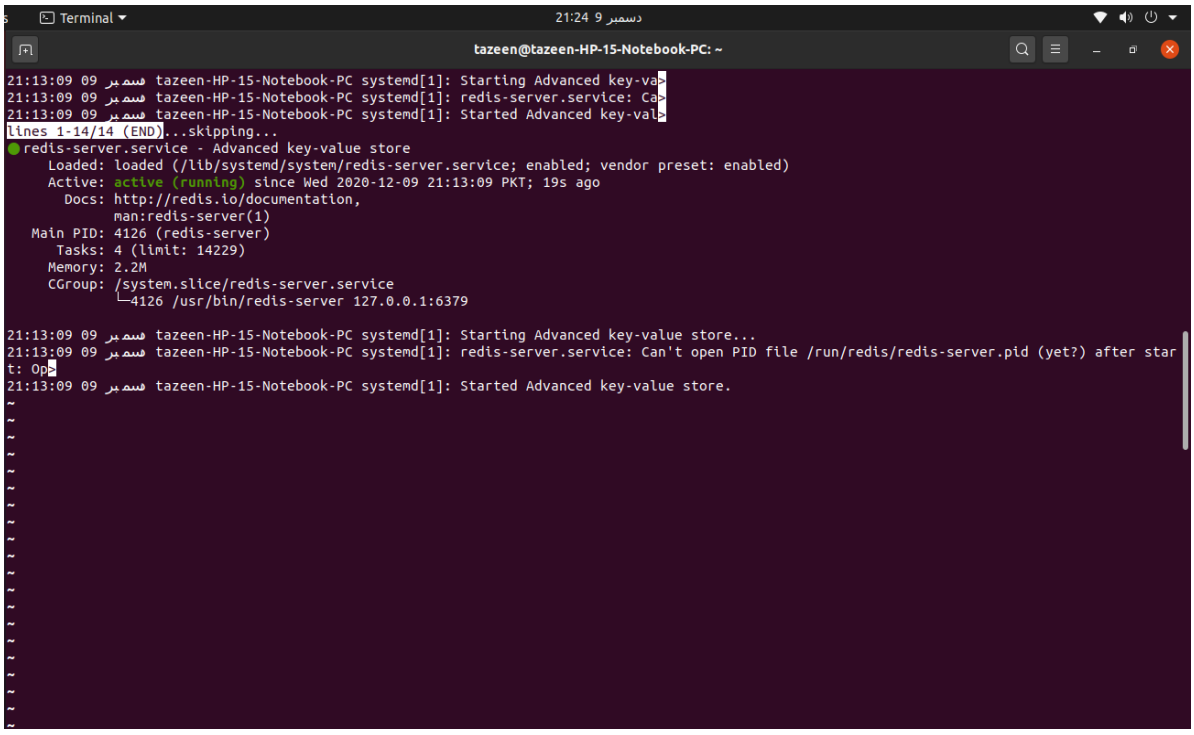
4. **Technical Details**

Initially I worked on Redis lab commands which are:
Redis is an open-source in-memory key-value data store. It can be used as a database, cache and, message broker, and supports various data structures such as Strings, Hashes, Lists, Sets, and more.

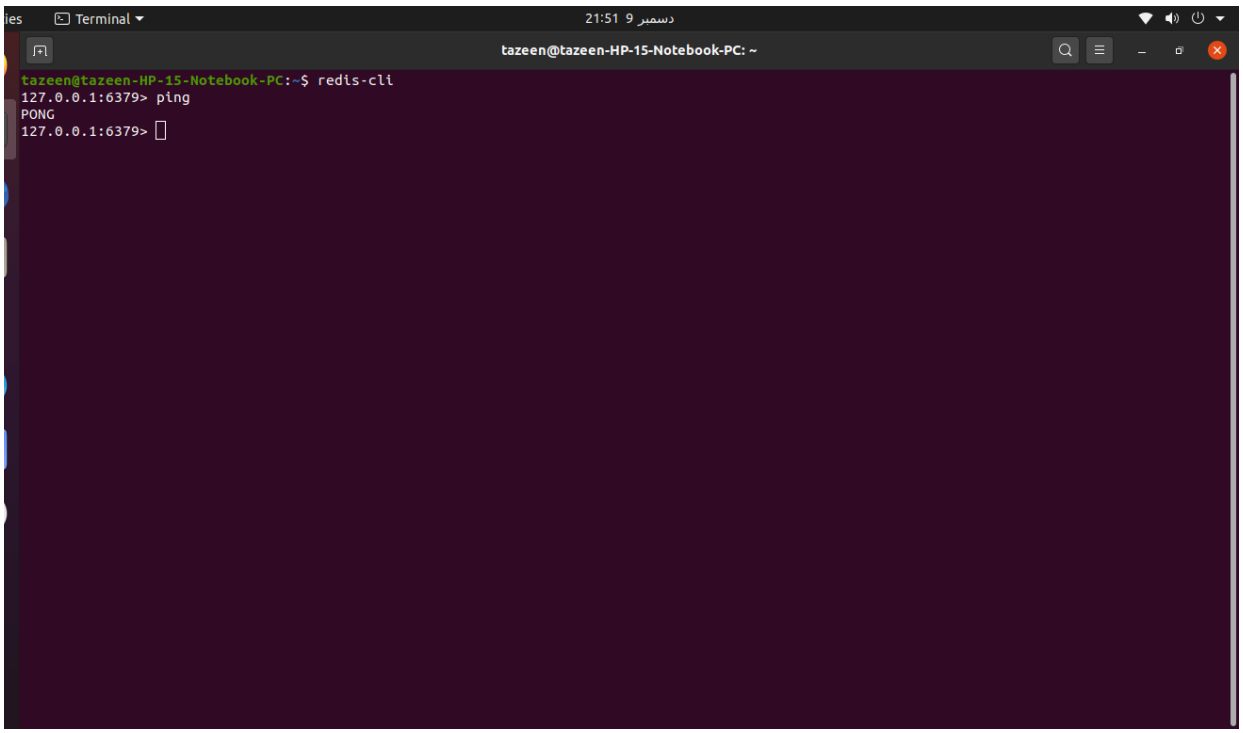
Installation in Ubuntu:

```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo apt update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://pk.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://security.ubuntu.com/ubuntu focal-security InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
115 packages can be upgraded. Run 'apt list --upgradable' to see them.
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo apt install redis-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libhiredis0.14 libjemalloc2 liblua5.1-0 lua-bitop lua-cjson redis-tools
Suggested packages:
  ruby-redis
The following NEW packages will be installed:
  libhiredis0.14 libjemalloc2 liblua5.1-0 lua-bitop lua-cjson redis-server
  redis-tools
0 upgraded, 7 newly installed, 0 to remove and 115 not upgraded.
Need to get 915 kB of archives.
After this operation, 4,077 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 libhiredis0.14 amd64 0.14.0-6 [30.2 kB]
Get:2 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 libjemalloc2 amd64 5.2.1-1ubuntu1 [235 kB]
Get:3 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 liblua5.1-0 amd64 5.1.5-8.1build4 [99.9 kB]
Get:4 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 lua-bitop amd64 1.0.2-5 [6,680 B]
Get:5 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 lua-cjson amd64 2.1.0+dfsg-2.1 [17.4 kB]
Get:6 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 redis-tools amd64 5:5.0.7-2 [489 kB]
Get:7 http://pk.archive.ubuntu.com/ubuntu focal/universe amd64 redis-server amd64 5:5.0.7-2 [37.3 kB]
Fetched 915 kB in 3s (331 kB/s)
Selecting previously unselected package libhiredis0.14:amd64.
(Reading database ... 178989 files and directories currently installed.)
Preparing to unpack .../0-libhiredis0.14_0.14.0-6_amd64.deb ...
Unpacking libhiredis0.14:amd64 (0.14.0-6) ...
Selecting previously unselected package libjemalloc2:amd64.
Preparing to unpack .../1-libjemalloc2_5.2.1-1ubuntu1_amd64.deb ...
Unpacking libjemalloc2:amd64 (5.2.1-1ubuntu1) ...
Selecting previously unselected package liblua5.1-0:amd64.
```

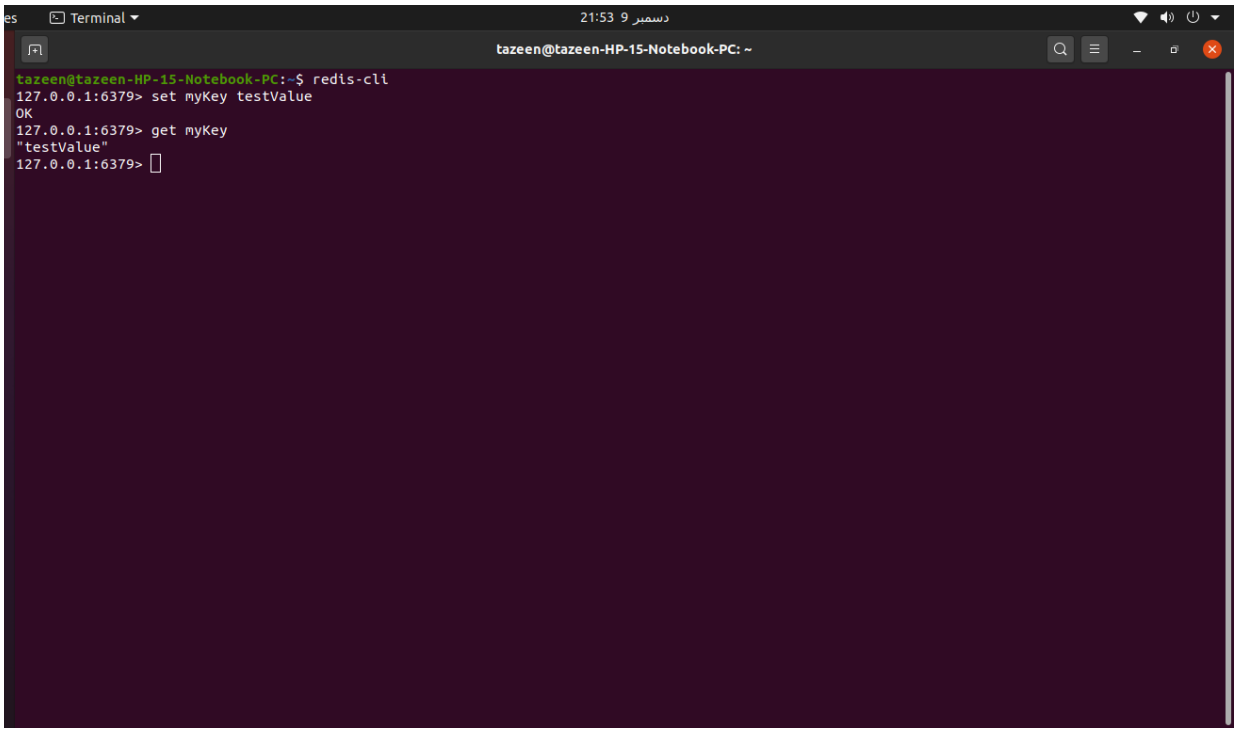
```
Selecting previously unselected package liblua5.1-0:amd64.
Preparing to unpack .../2-liblua5.1-0_5.1.5-8.1build4_amd64.deb ...
Unpacking liblua5.1-0:amd64 (5.1.5-8.1build4) ...
Selecting previously unselected package lua-bitop:amd64.
Preparing to unpack .../3-lua-bitop_1.0.2-5_amd64.deb ...
Unpacking lua-bitop:amd64 (1.0.2-5) ...
Selecting previously unselected package lua-cjson:amd64.
Preparing to unpack .../4-lua-cjson_2.1.0+dfsg-2.1_amd64.deb ...
Unpacking lua-cjson:amd64 (2.1.0+dfsg-2.1) ...
Selecting previously unselected package redis-tools.
Preparing to unpack .../5-redis-tools_5%3a5.0.7-2_amd64.deb ...
Unpacking redis-tools (5:5.0.7-2) ...
Selecting previously unselected package redis-server.
Preparing to unpack .../6-redis-server_5%3a5.0.7-2_amd64.deb ...
Unpacking redis-server (5:5.0.7-2) ...
Setting up libjemalloc2:amd64 (5.2.1-1ubuntu1) ...
Setting up lua-cjson:amd64 (2.1.0+dfsg-2.1) ...
Setting up lua-bitop:amd64 (1.0.2-5) ...
Setting up liblua5.1-0:amd64 (5.1.5-8.1build4) ...
Setting up libhiredis0.14:amd64 (0.14.0-6) ...
Setting up redis-tools (5:5.0.7-2) ...
Setting up redis-server (5:5.0.7-2) ...
Created symlink /etc/systemd/system/redis.service → /lib/systemd/system/redis-se
rver.service.
Created symlink /etc/systemd/system/multi-user.target.wants/redis-server.service
 → /lib/systemd/system/redis-server.service.
Processing triggers for systemd (245.4-4ubuntu3.2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo systemctl status redis-server
● redis-server.service - Advanced key-value store
     Loaded: loaded (/lib/systemd/system/redis-server.service; enabled; vendor >
     Active: active (running) since Wed 2020-12-09 21:13:09 PKT; 19s ago
       Docs: http://redis.io/documentation,
             man:redis-server(1)
   Main PID: 4126 (redis-server)
      Tasks: 4 (limit: 14229)
     Memory: 2.2M
     CGroup: /system.slice/redis-server.service
```



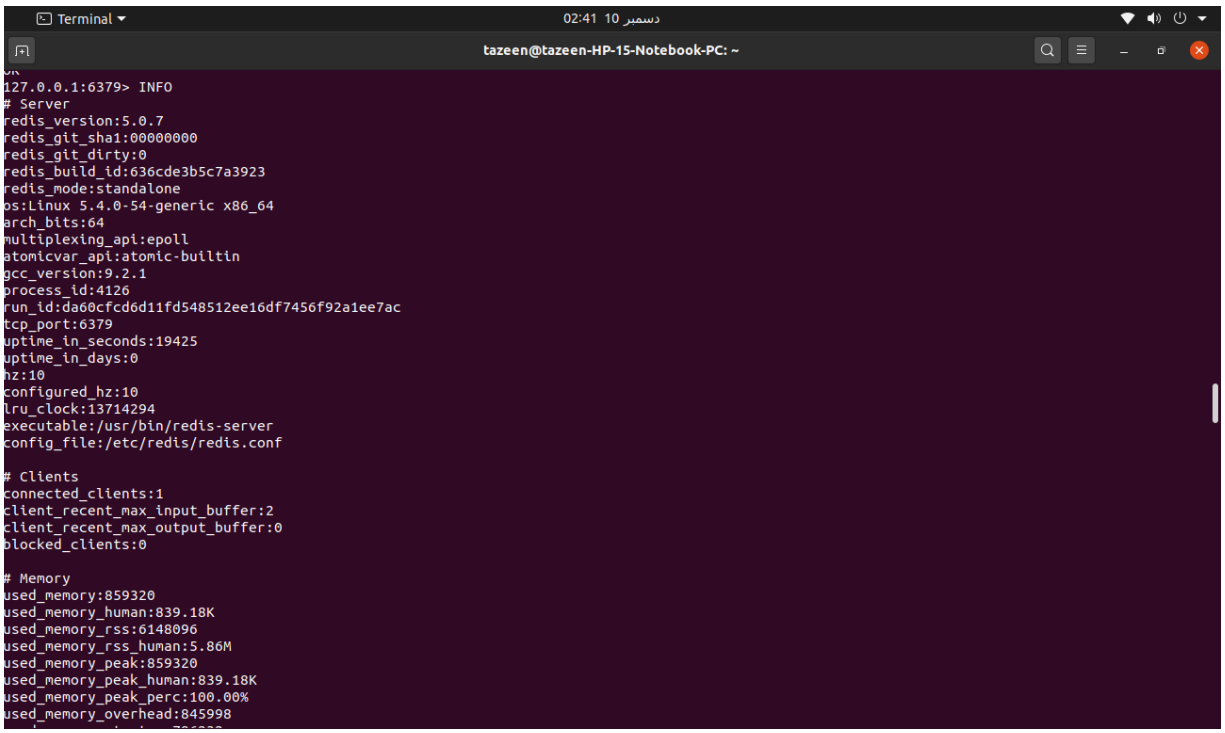
Redis cli: Check if redis is running

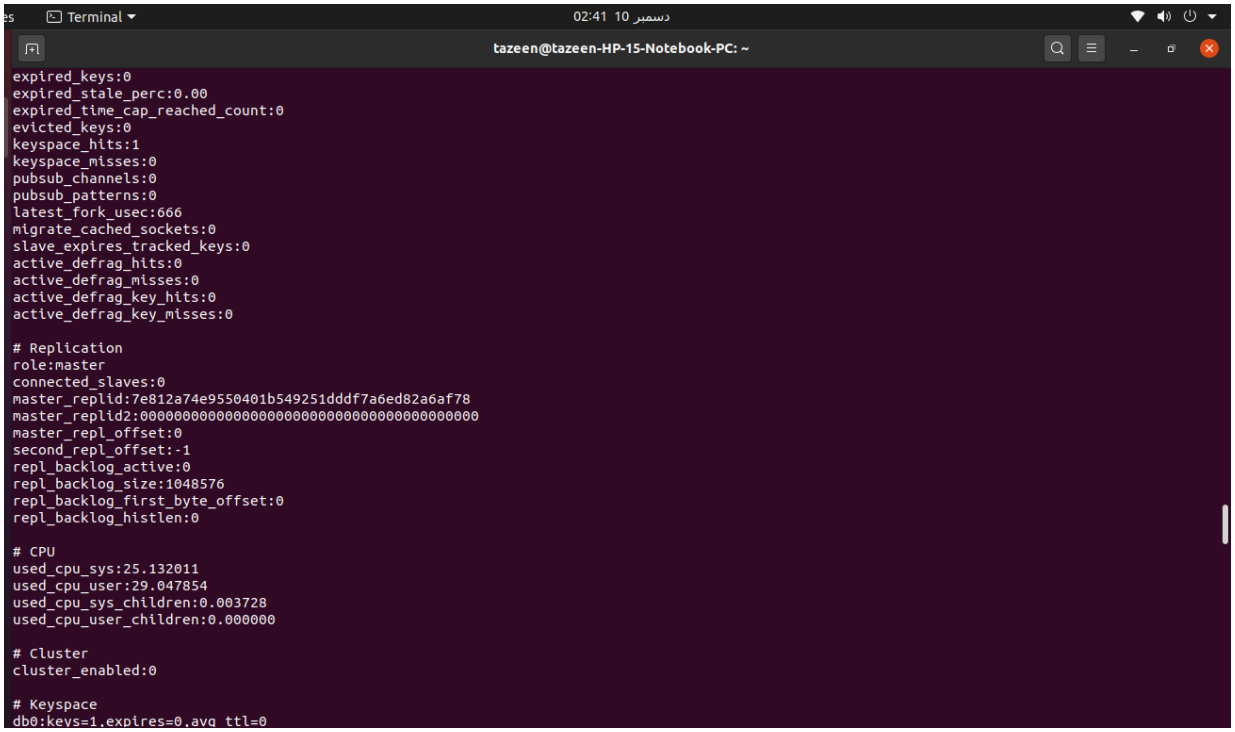


Get and Set key value:



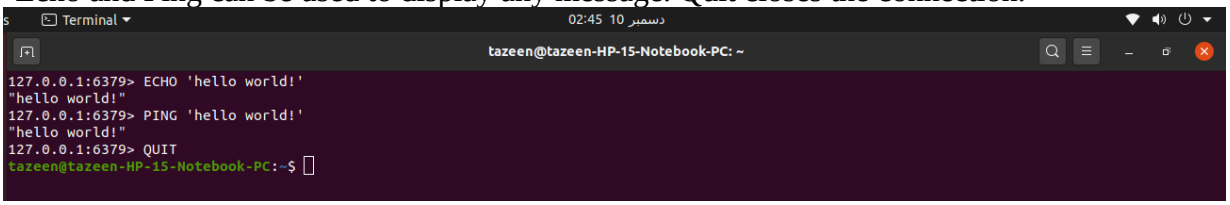
Commands to information about client connected:





List all databases and select any database by index. All the databases are listed with indexes such as; db0 which this database is on 0 index.

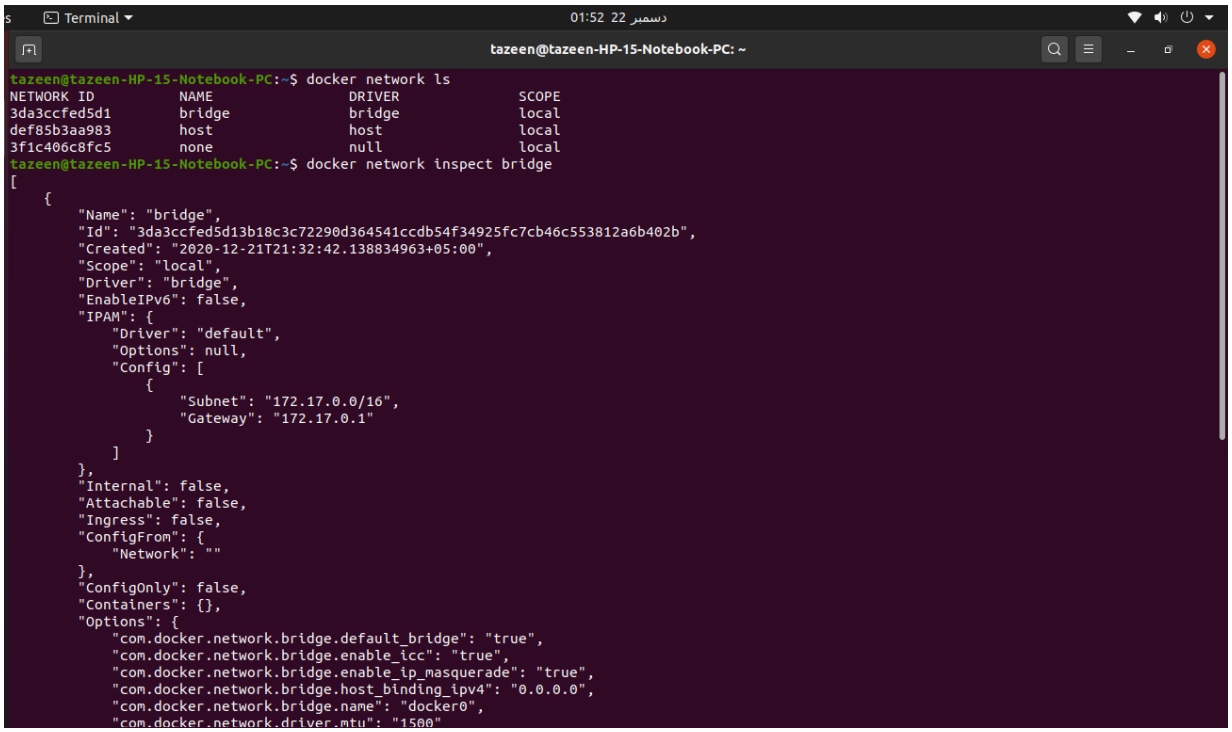
Echo and Ping can be used to display any message. Quit closes the connection.

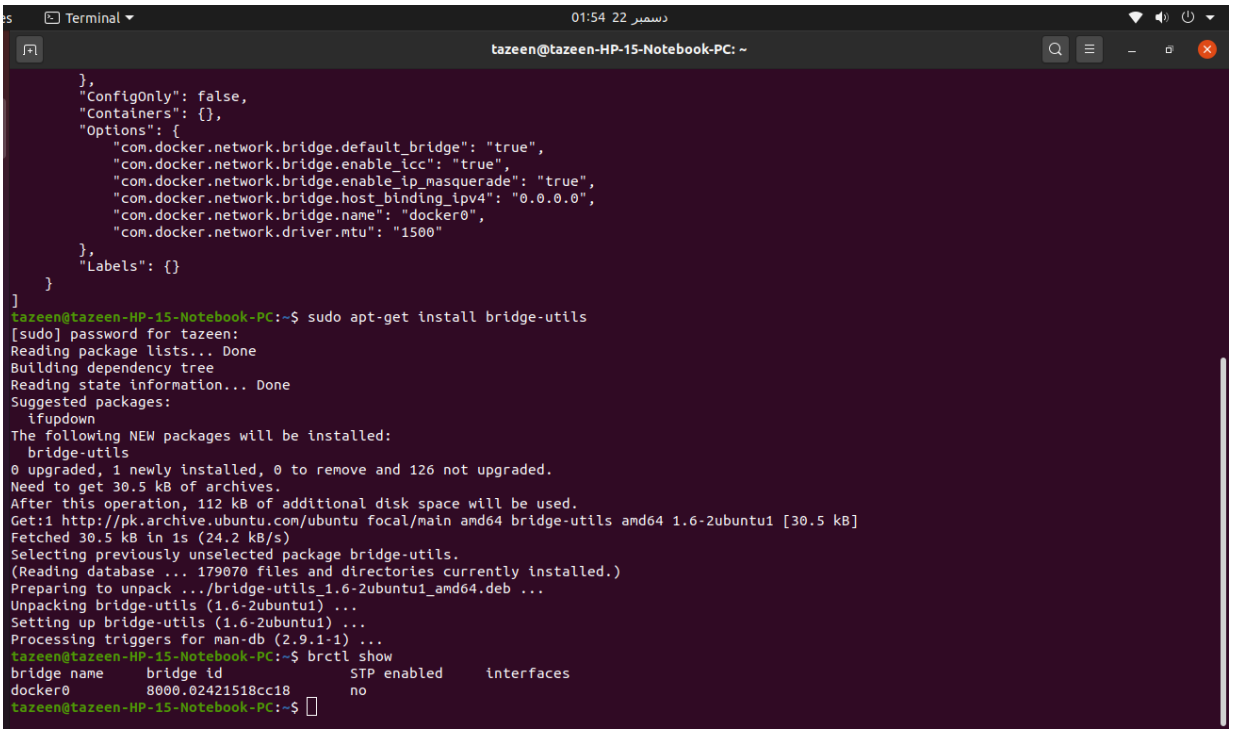


Lambda Architecture:

I- Database (mongodb layer)

Creating docker-compose.yml file for mongo db container



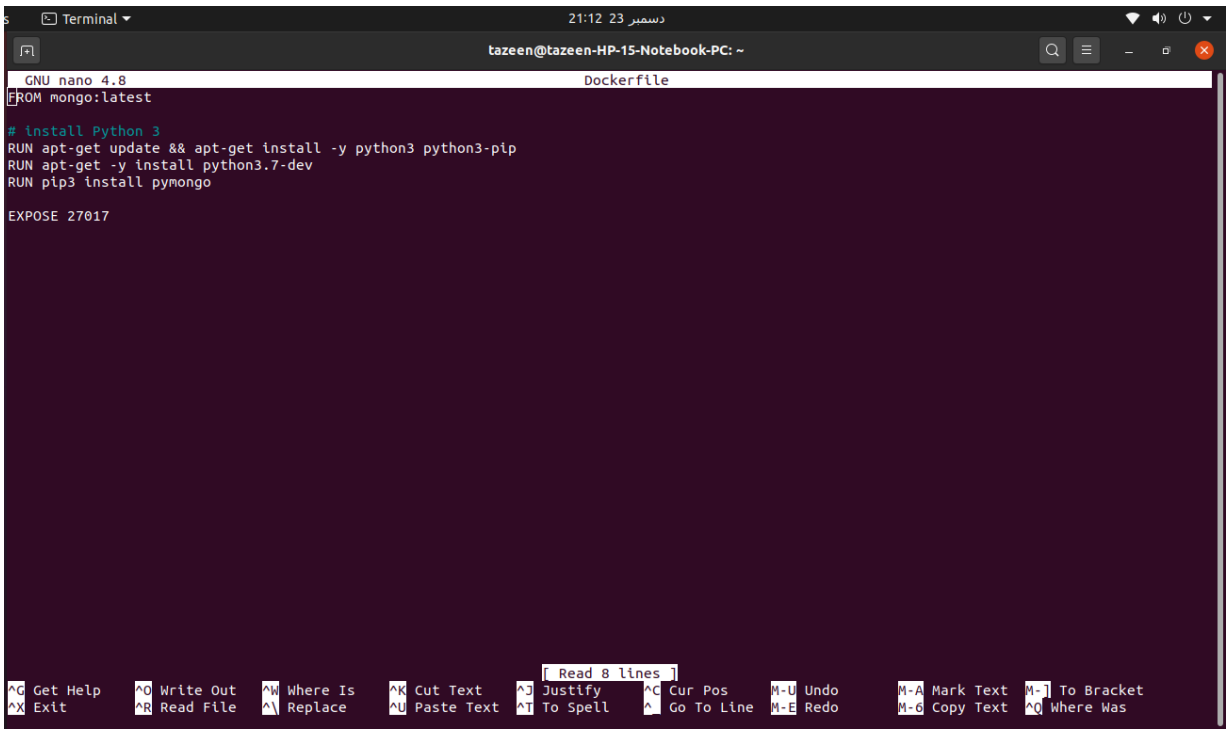


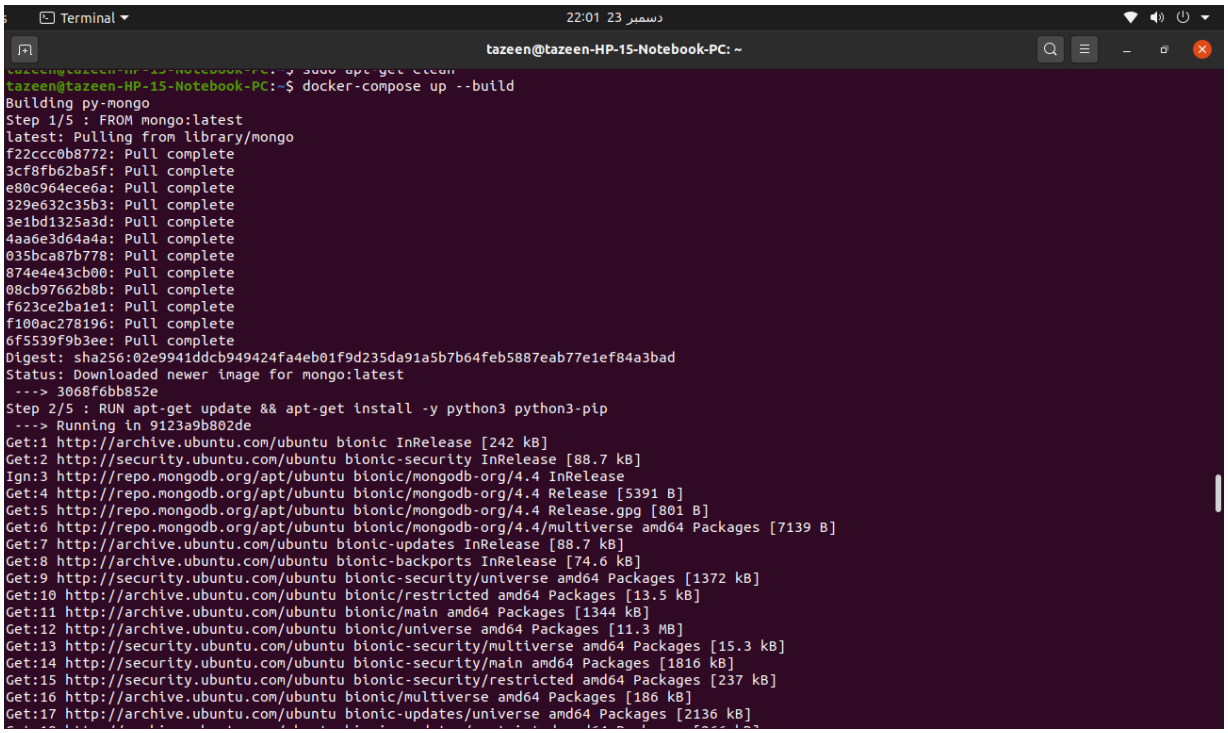
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname
-m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   651  100   651    0     0   1493      0 --:--:-- --:--:-- --:--:--  1496
100 11.6M  100 11.6M    0     0   317k      0  0:00:37  0:00:37 --:--:--  315k
tazeen@tazeen-HP-15-Notebook-PC:~$
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo chmod +x /usr/local/bin/docker-compose
tazeen@tazeen-HP-15-Notebook-PC:~$ docker-compose --version
docker-compose version 1.27.4, build 40524192
```

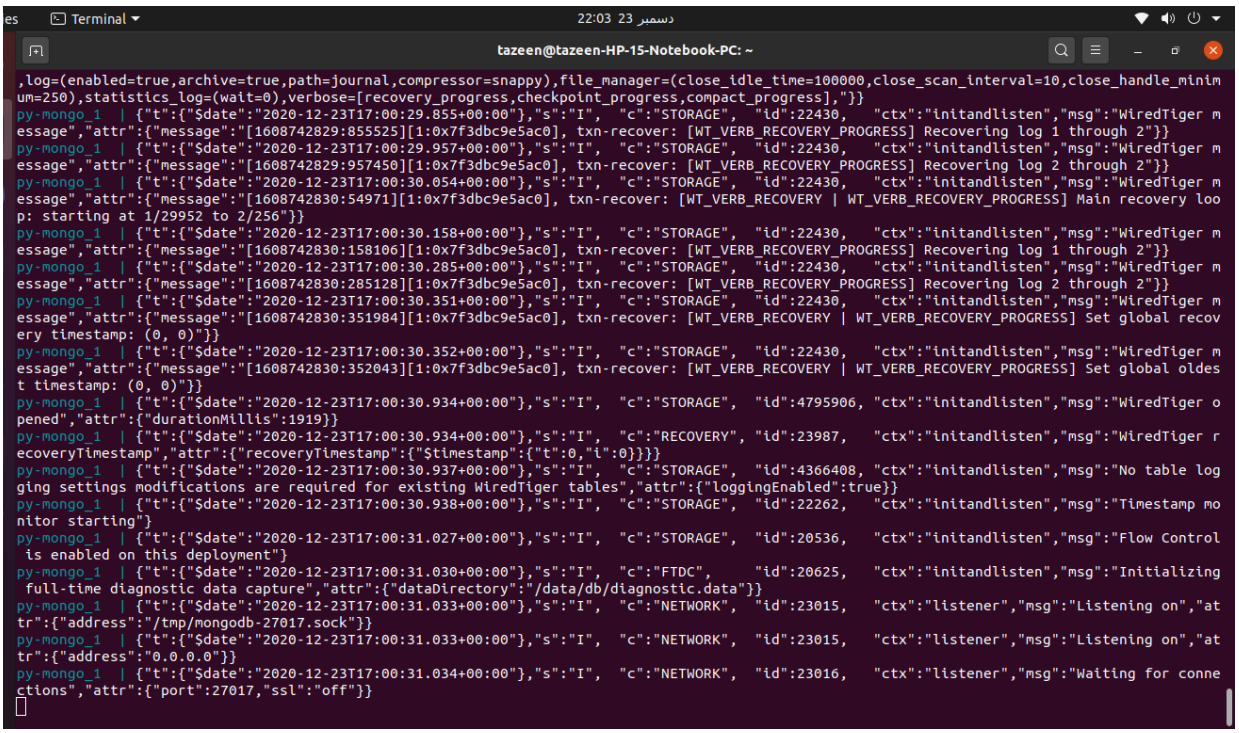
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano docker-compose.yml
[sudo] password for tazeen:
```

```
Terminal                                    21:11 23 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
  GNU nano 4.8                              docker-compose.yml
version: "3.2"
services:
 py-mongo:
  # build the image from Dockerfile
  build:
   context: ${PWD}
  volumes:
   - $PWD/mongo-data:/data/db
   - $PWD/mongo-app:/var/www/html
  ports:
   - "27017:27017"
  environment:
   - MONGO_INITDB_ROOT_USERNAME=root
   - MONGO_INITDB_ROOT_PASSWORD=1234

                          [ Read 14 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
```

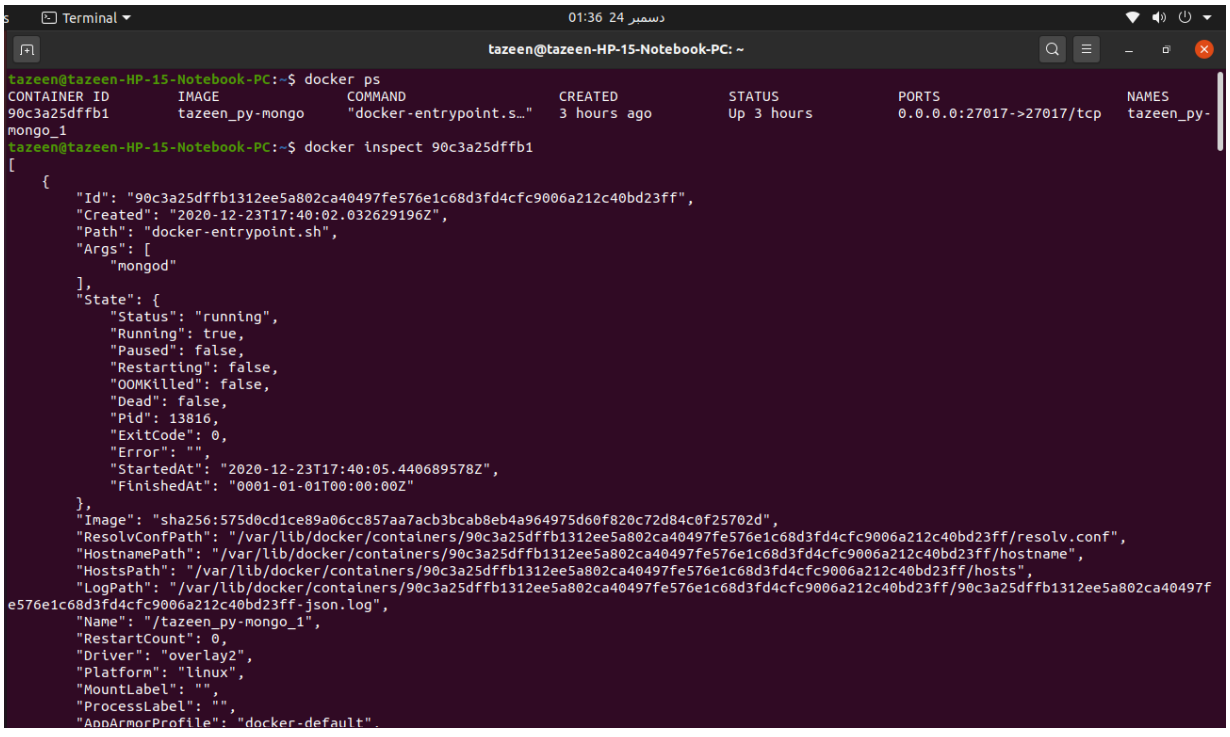
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```

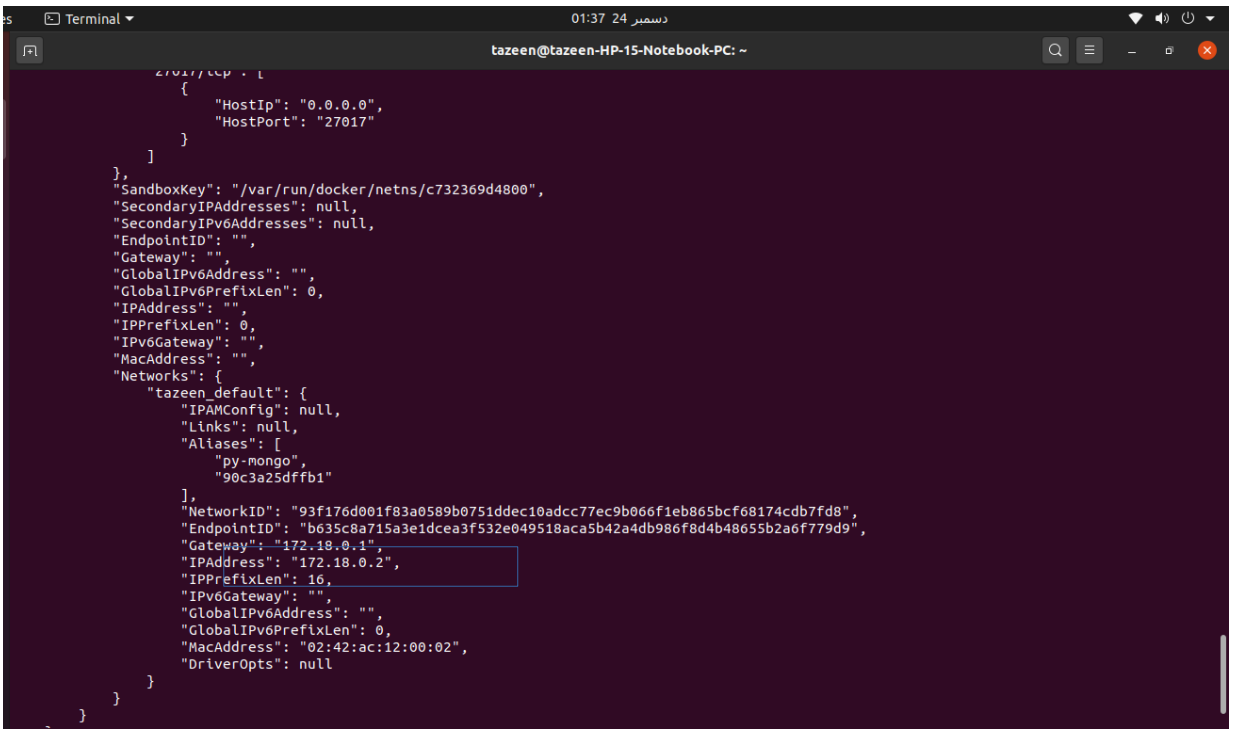

Terminal
21:12 23 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
GNU nano 4.8
Dockerfile
FROM mongo:latest
install Python 3
RUN apt-get update && apt-get install -y python3 python3-pip
RUN apt-get -y install python3.7-dev
RUN pip3 install pymongo
EXPOSE 27017
[Read 8 lines]
^G Get Help ^O Write Out ^W Where 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

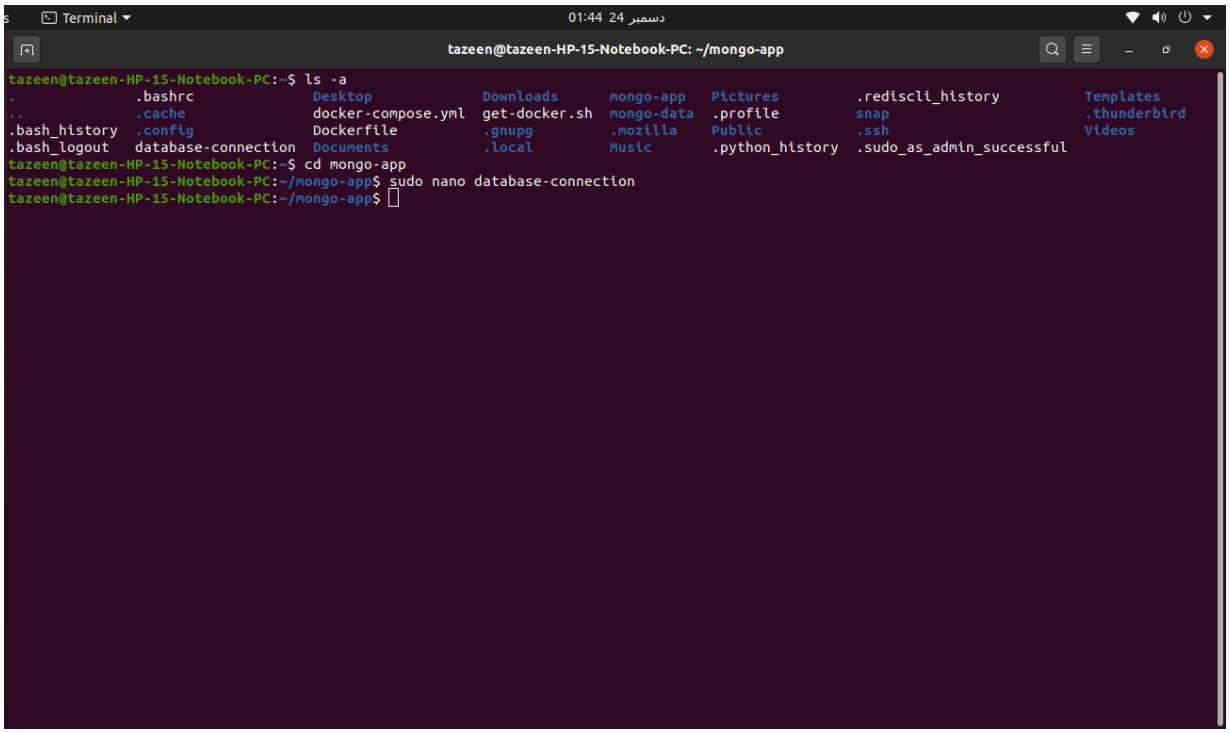

Terminal
22:01 23 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
tazeen@tazeen-HP-15-Notebook-PC:~$ docker-compose up --build
Building py-mongo
Step 1/5 : FROM mongo:latest
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ece6a: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b8b: Pull complete
f623ce2ba1e1: Pull complete
f100ac278196: Pull complete
6f5539f9b3ee: Pull complete
Digest: sha256:02e9941ddcb949424fa4eb01f9d235da91a5b7b64feb5887eab77e1ef84a3bad
Status: Downloaded newer image for mongo:latest
---> 3068f6bb852e
Step 2/5 : RUN apt-get update && apt-get install -y python3 python3-pip
---> Running in 9123a9b802de
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Ign:3 http://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 InRelease
Get:4 http://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 Release [5391 B]
Get:5 http://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 Release.gpg [801 B]
Get:6 http://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4/multiverse amd64 Packages [7139 B]
Get:7 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:9 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1372 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:13 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [15.3 kB]
Get:14 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1816 kB]
Get:15 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [237 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2136 kB]

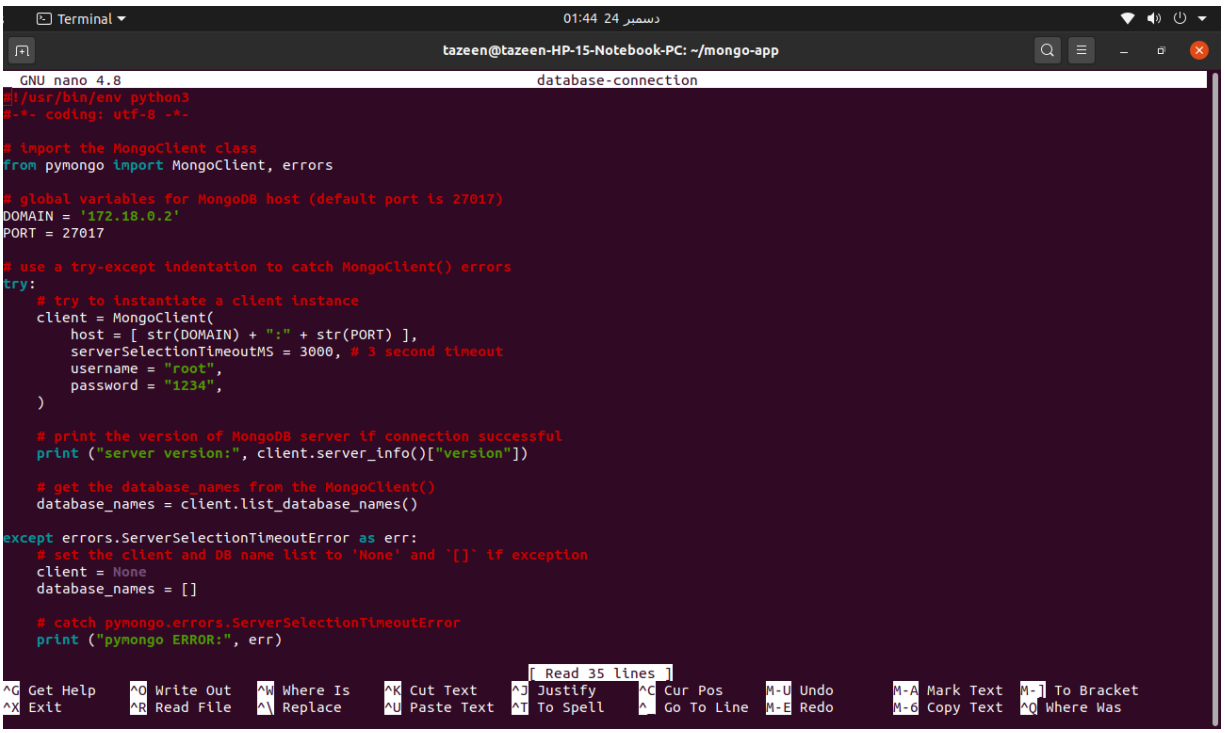


Get the docker container id and inspect it to see the IP address

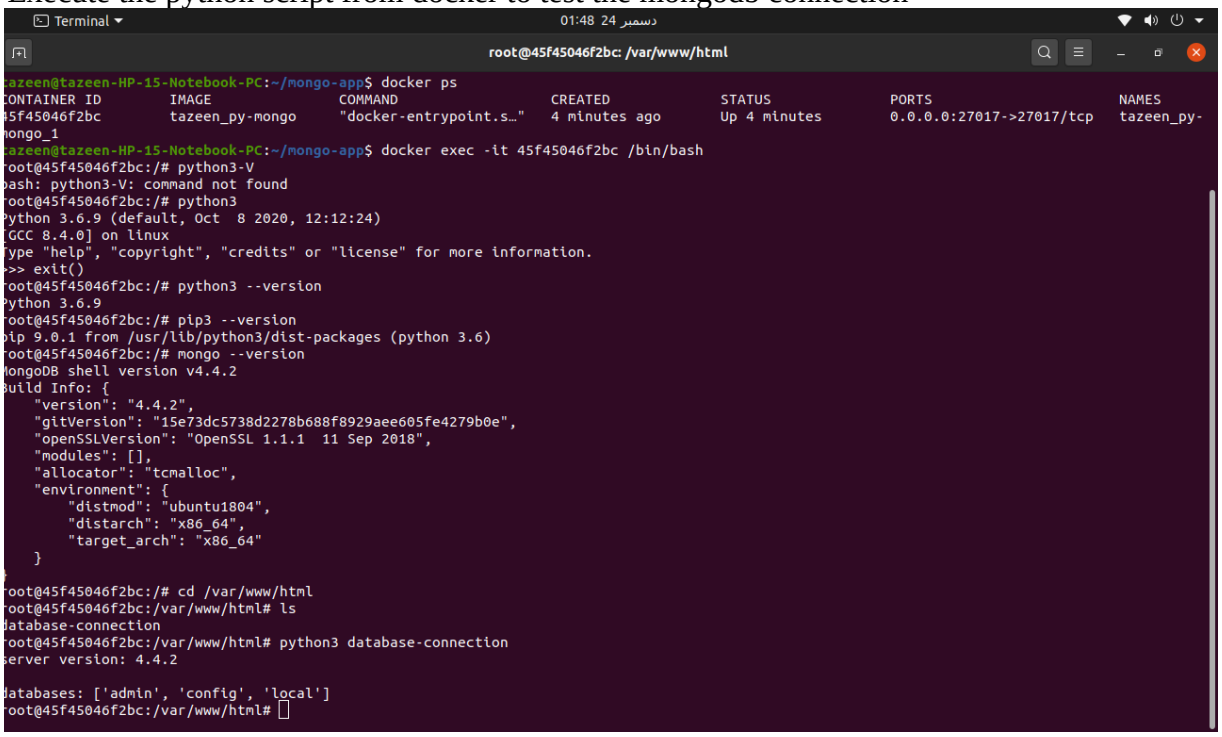



Terminal
01:37 24 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~
{
"HostIp": "0.0.0.0",
"HostPort": "27017"
}
]
},
"SandboxKey": "/var/run/docker/netns/c732369d4800",
"SecondaryIPAddresses": null,
"SecondaryIPv6Addresses": null,
"EndpointID": "",
"Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 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
}
}
}

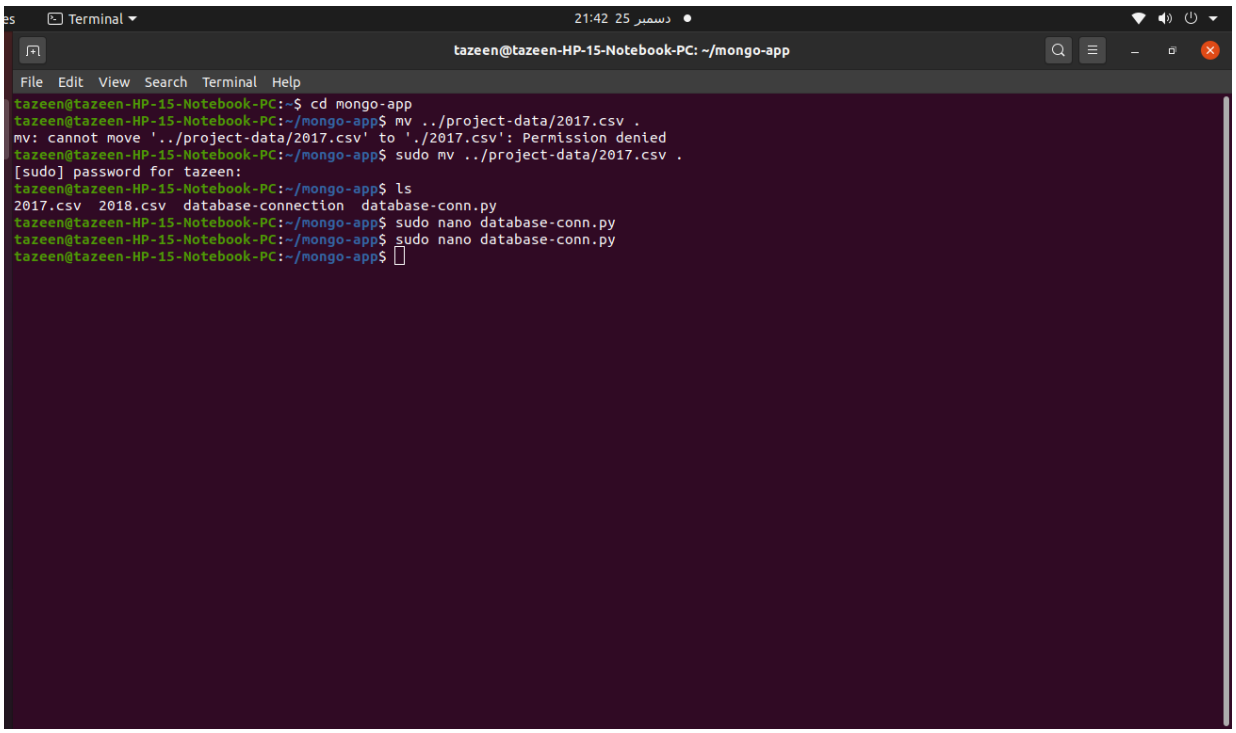

Terminal
01:44 24 دسمبر
tazeen@tazeen-HP-15-Notebook-PC: ~/mongo-app
tazeen@tazeen-HP-15-Notebook-PC:~$ ls -a
. .bashrc Desktop Downloads mongo-app Pictures .rediscli_history Templates
.. .cache docker-compose.yml get-docker.sh mongo-data .profile snap .thunderbird
.bash_history .config Dockerfile .gnupg .mozilla Public .ssh Videos
.bash_logout database-connection Documents .local Music .python_history .sudo_as_admin_successful
tazeen@tazeen-HP-15-Notebook-PC:~$ cd mongo-app
tazeen@tazeen-HP-15-Notebook-PC:~/mongo-app$ sudo nano database-connection
tazeen@tazeen-HP-15-Notebook-PC:~/mongo-app$



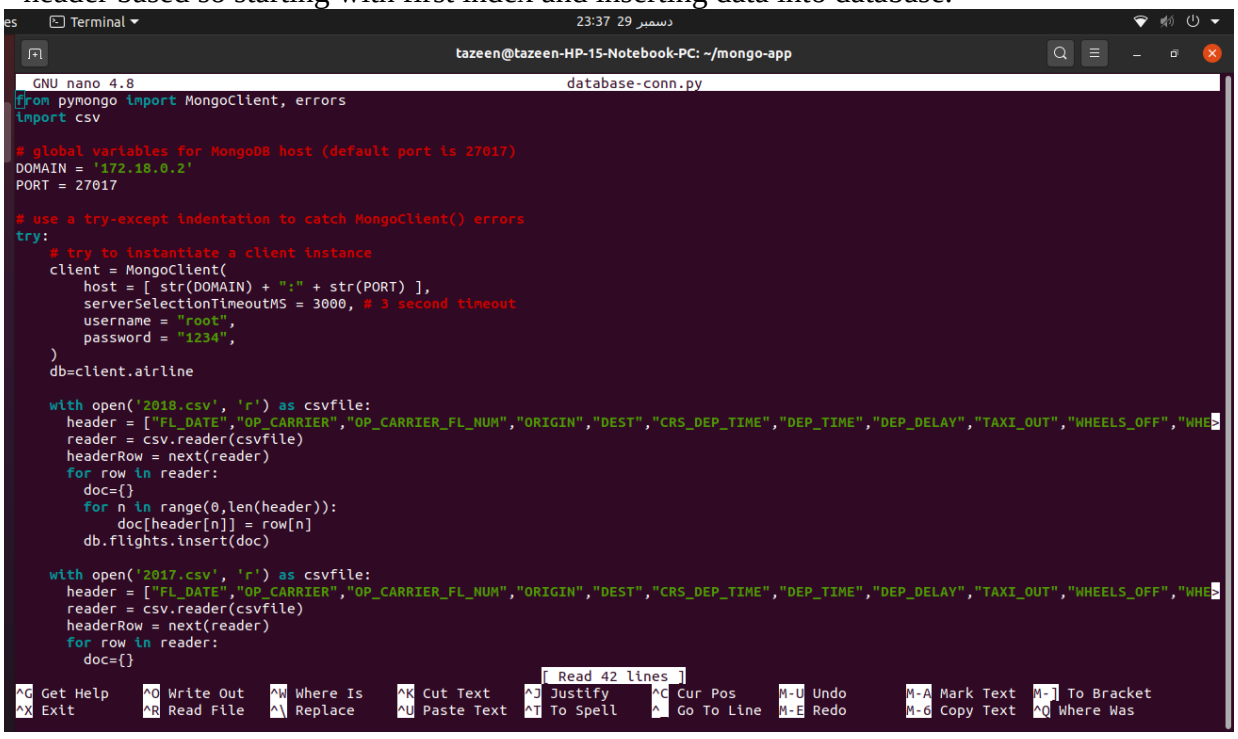
Execute the python script from docker to test the mongodb connection

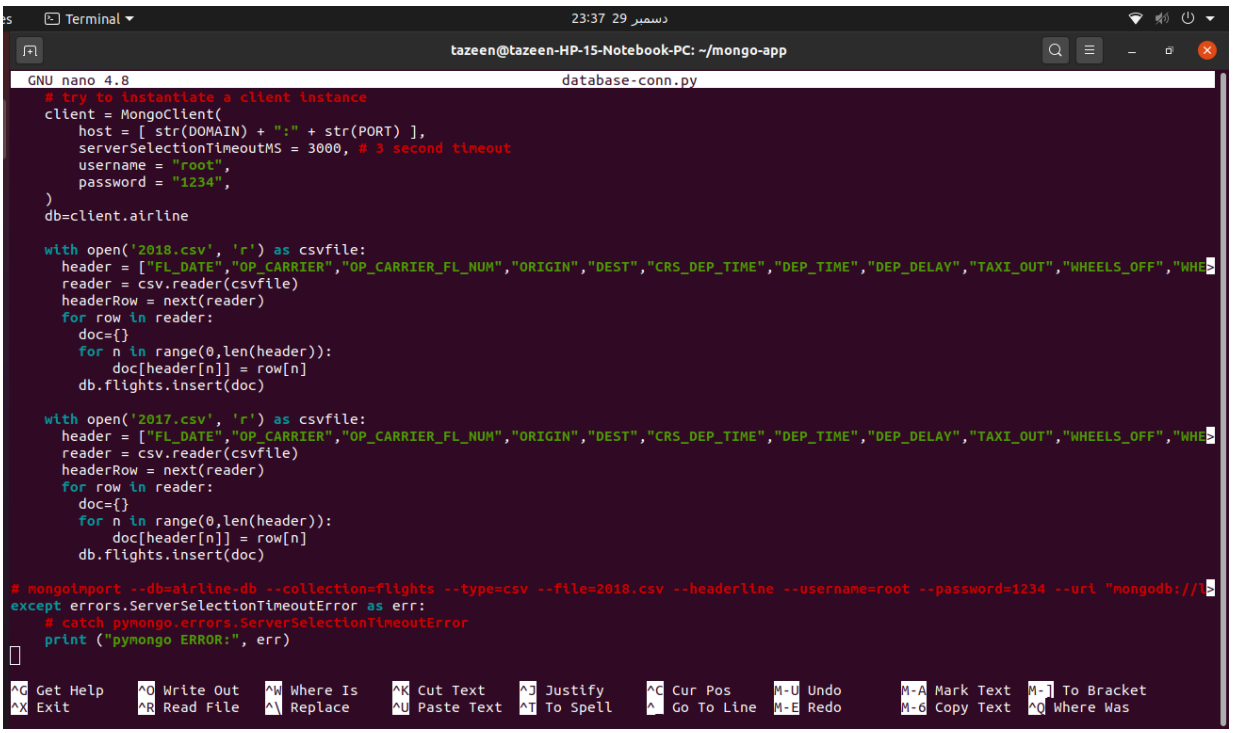


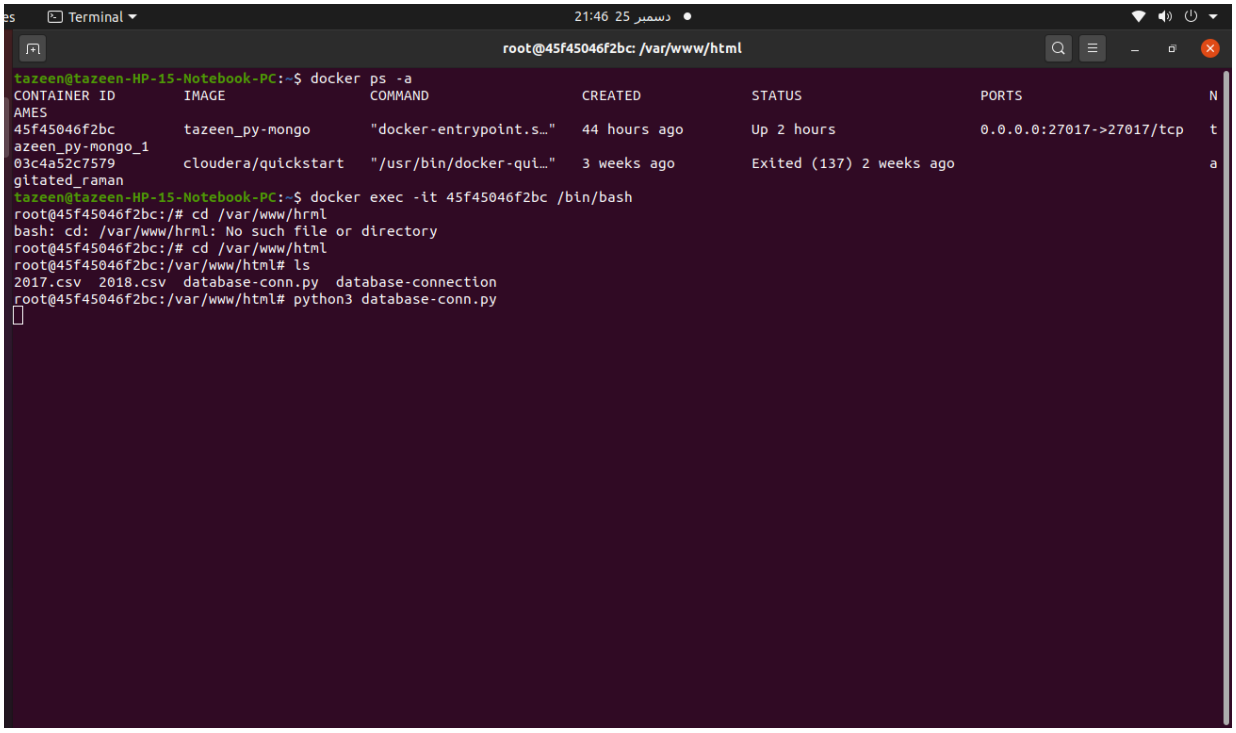
Creating python script to save data in mongodb



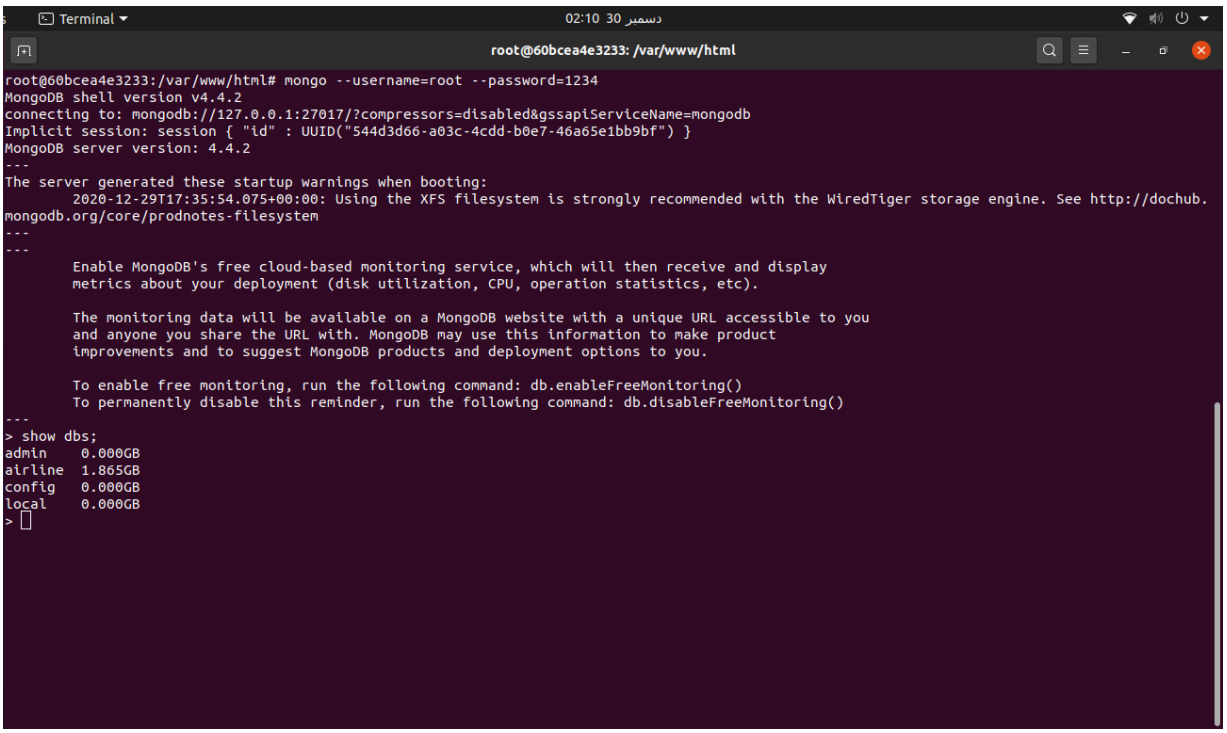
Saving data from 2018.csv and 2017.csv which is placed in mongo-app folder. These csv files are header based so starting with first index and inserting data into database.





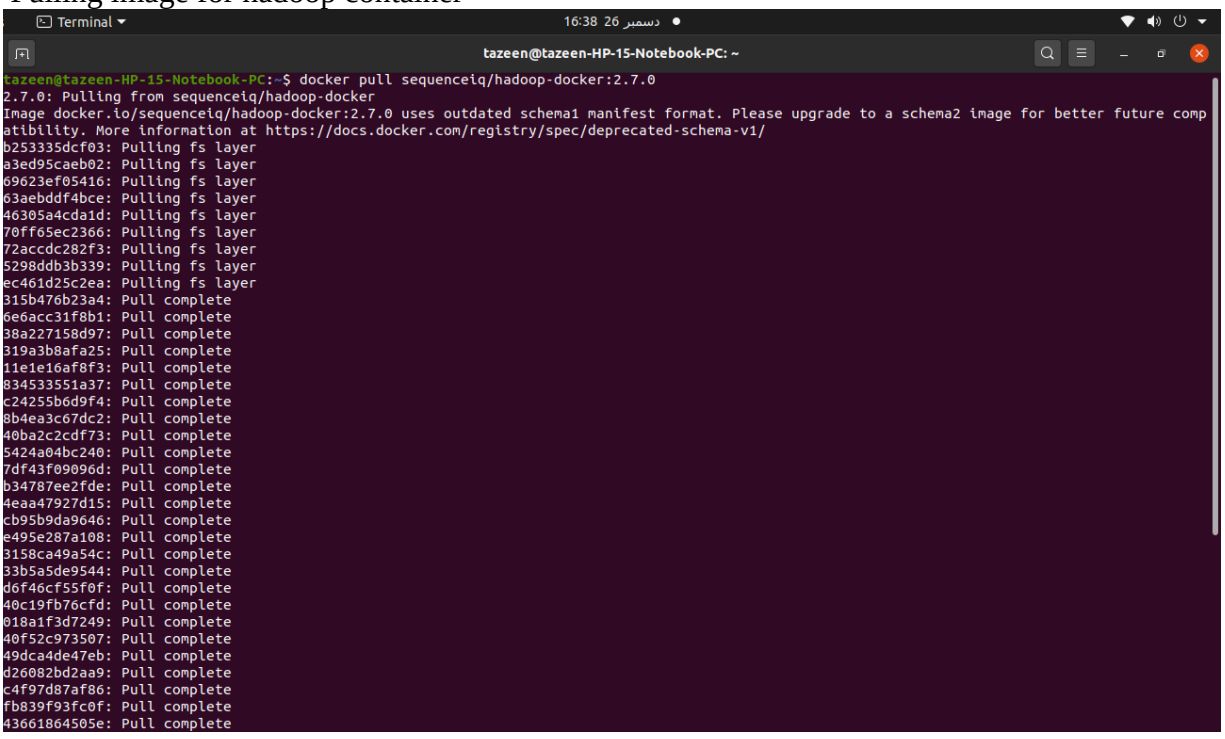


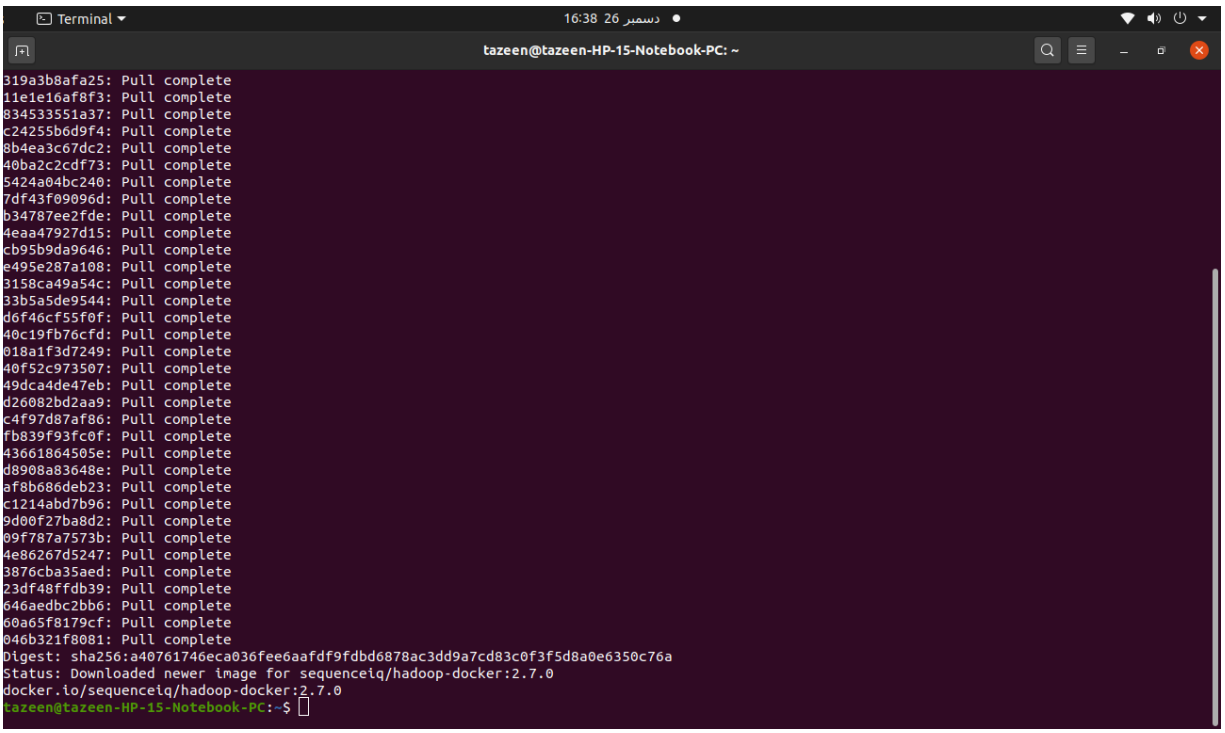
Connecting to mongodb to check that data is inserted in airline database.



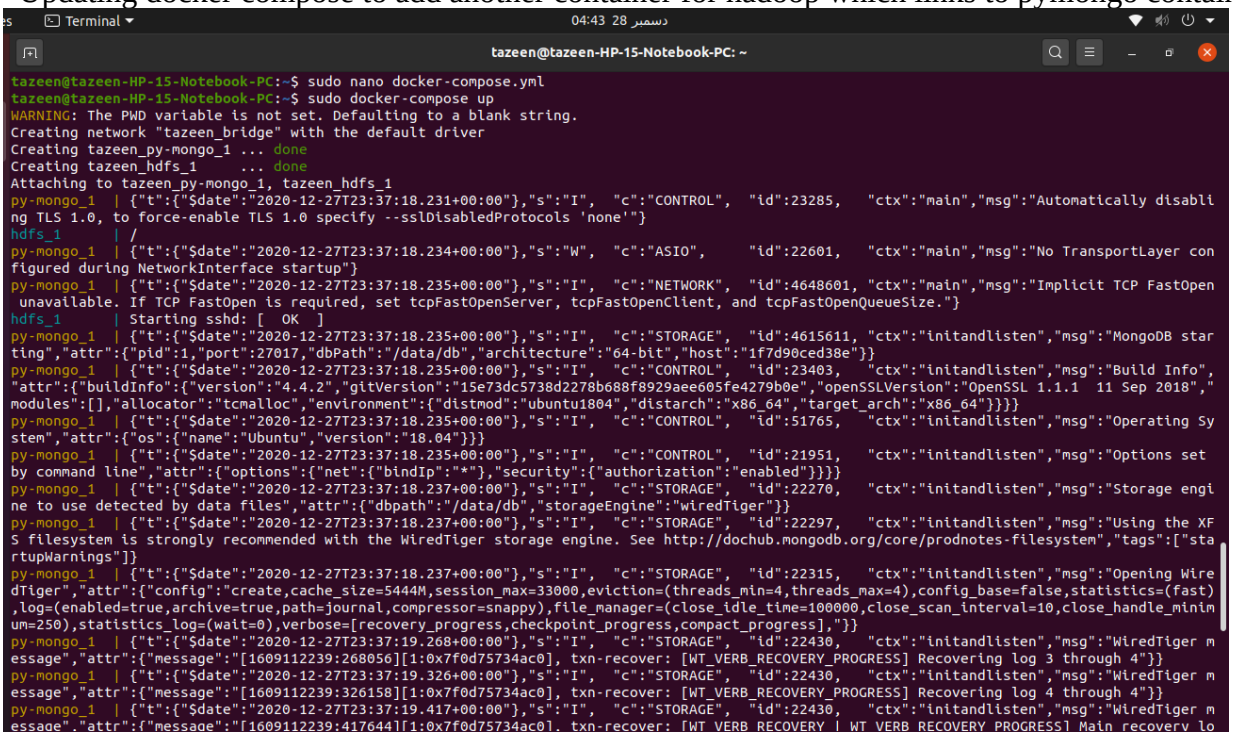
II- Batch processing layer:

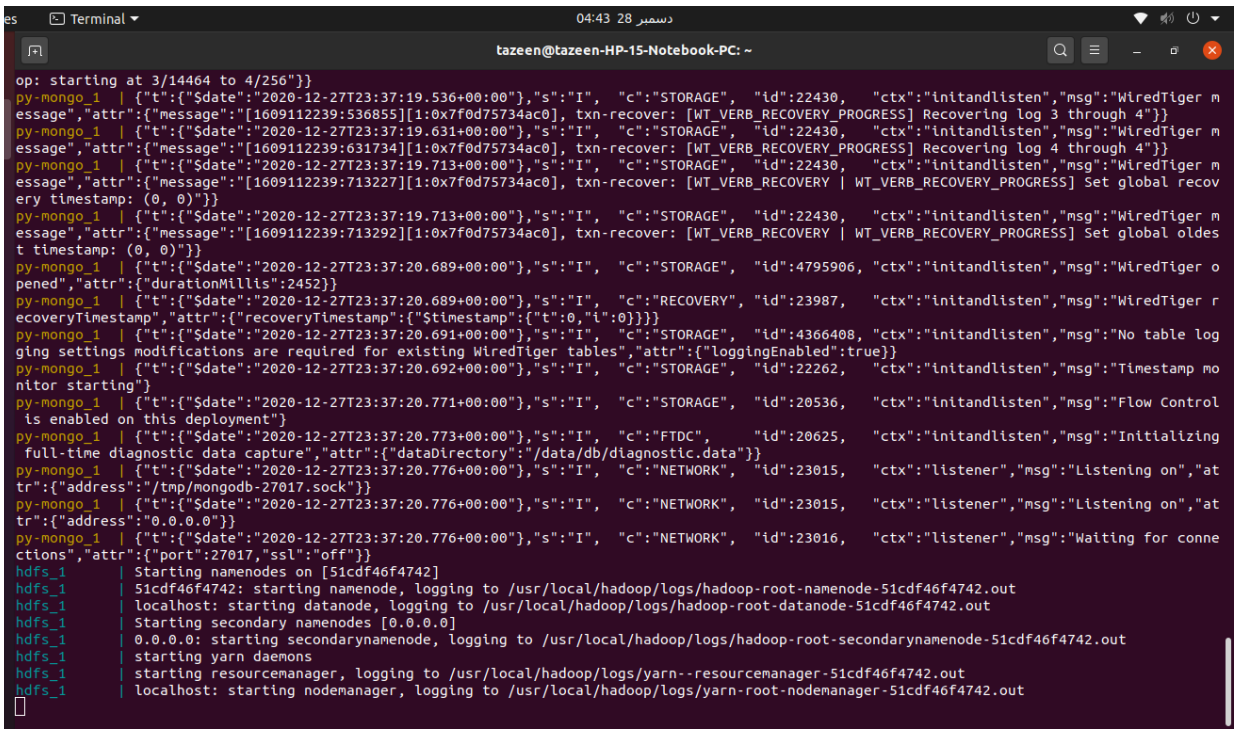
Pulling image for hadoop container

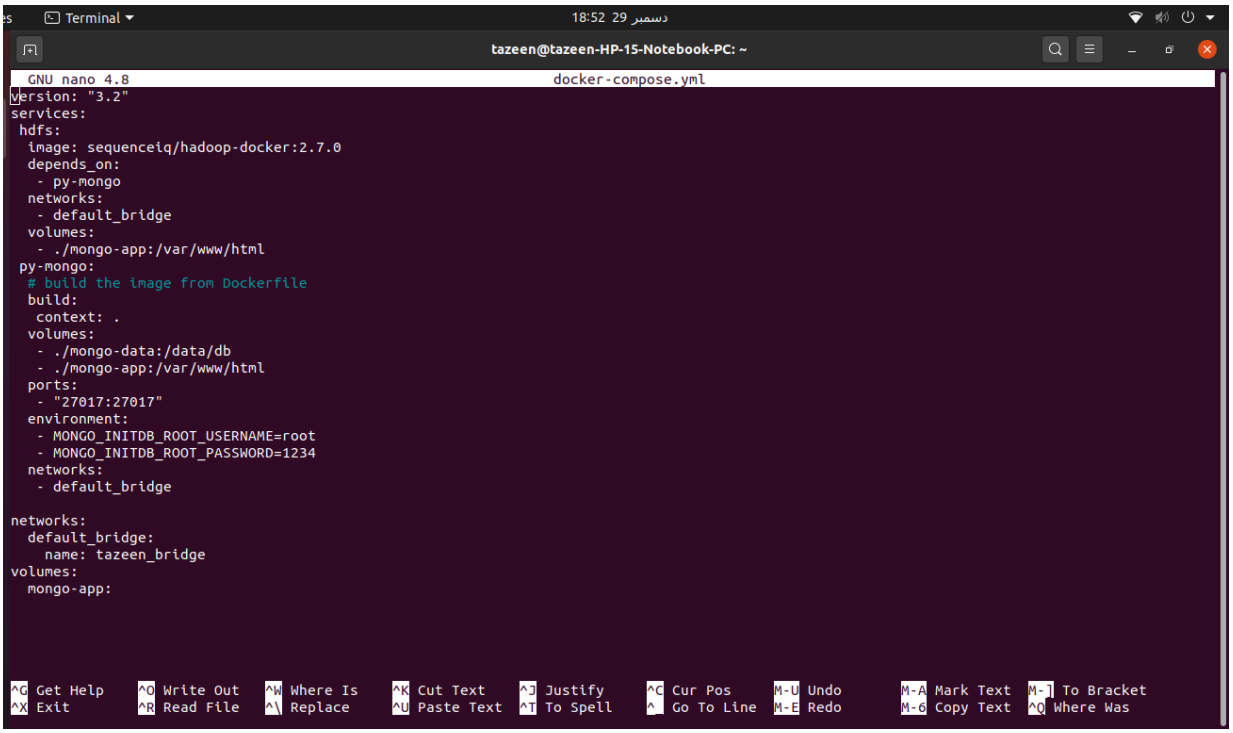




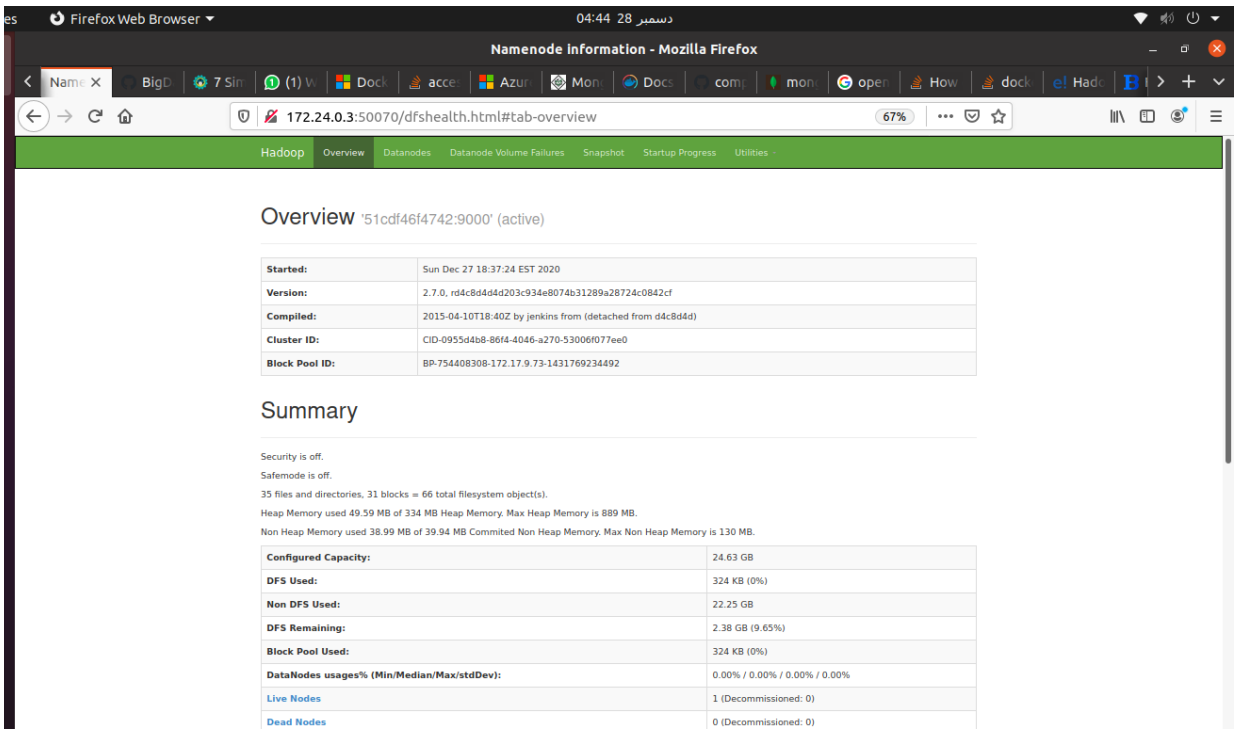
Updating docker compose to add another container for hadoop which links to pymongo container



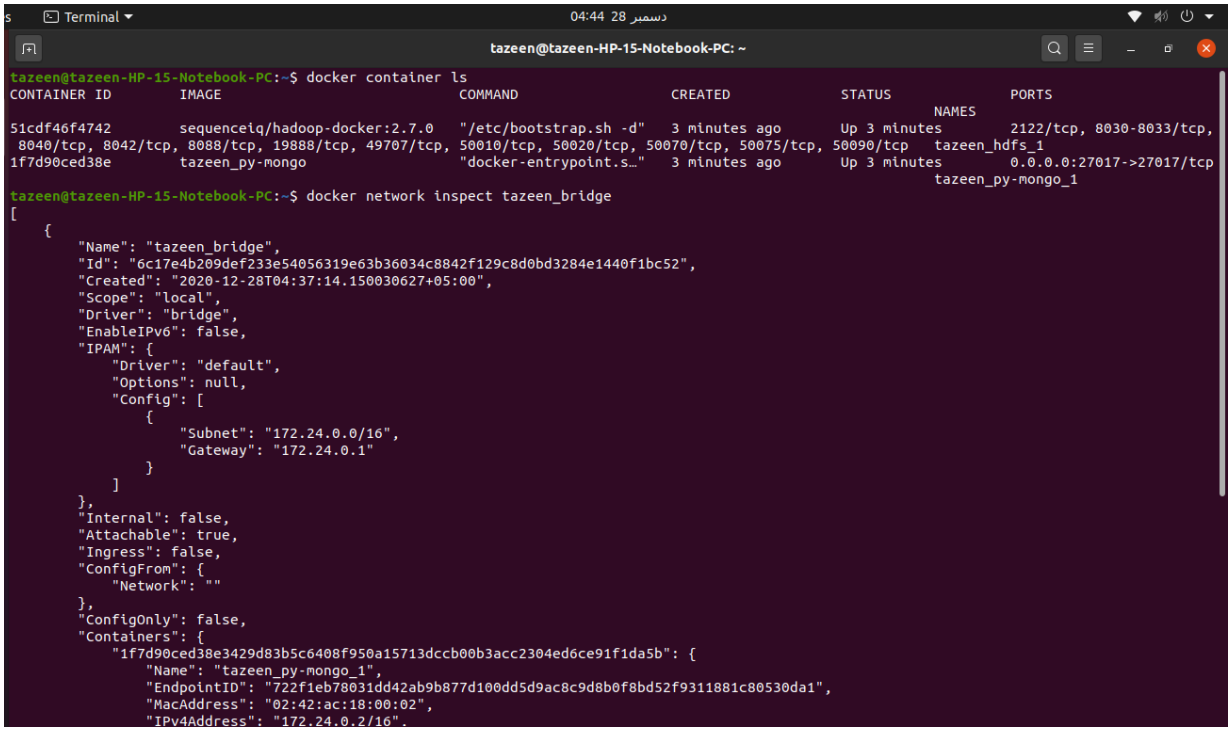


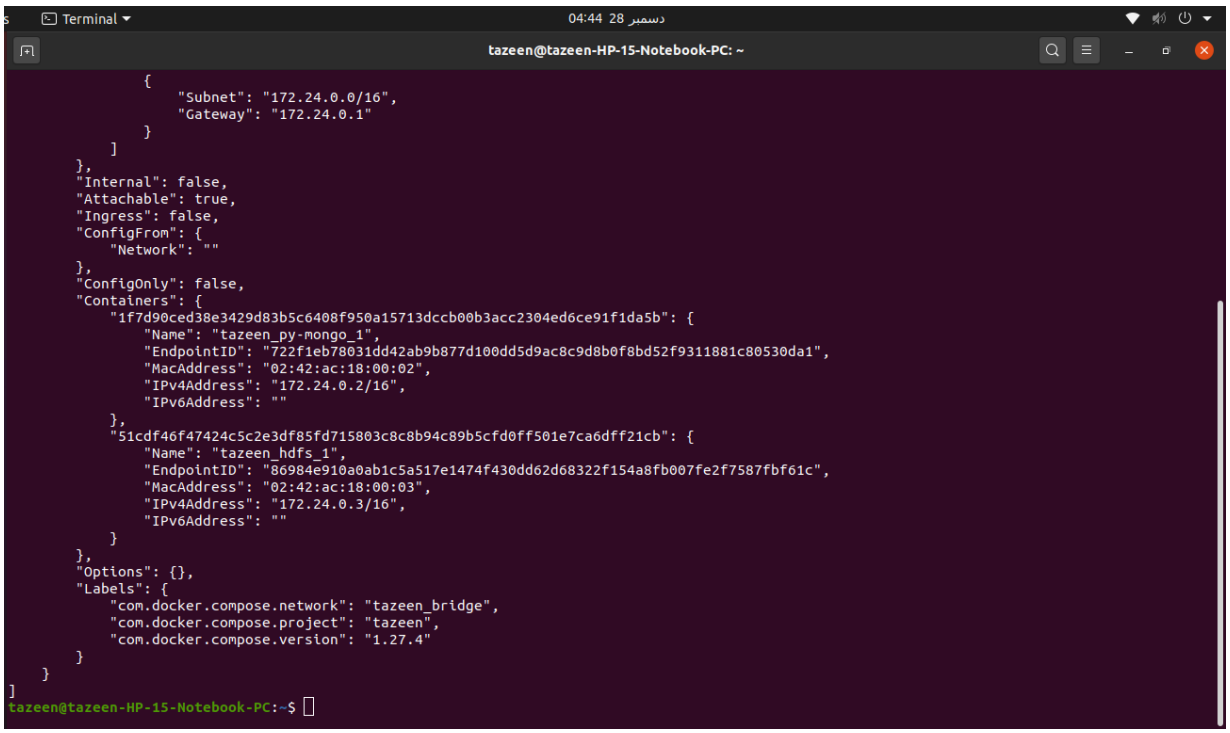


Hadoop container is correctly set up which can be viewed as:



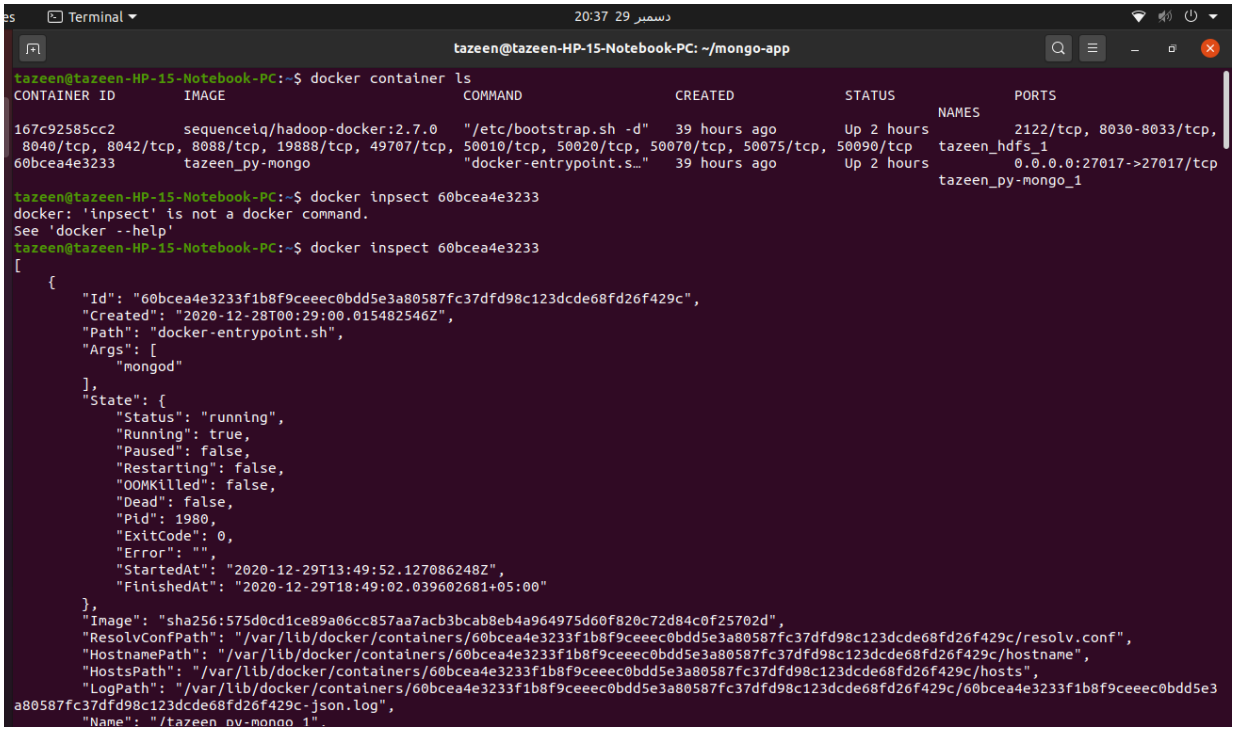
The network used to connect both containers is: tazeen_bridge which is a user defined bridge network. Both containers are connected on the same network which can be seen in below screenshot.

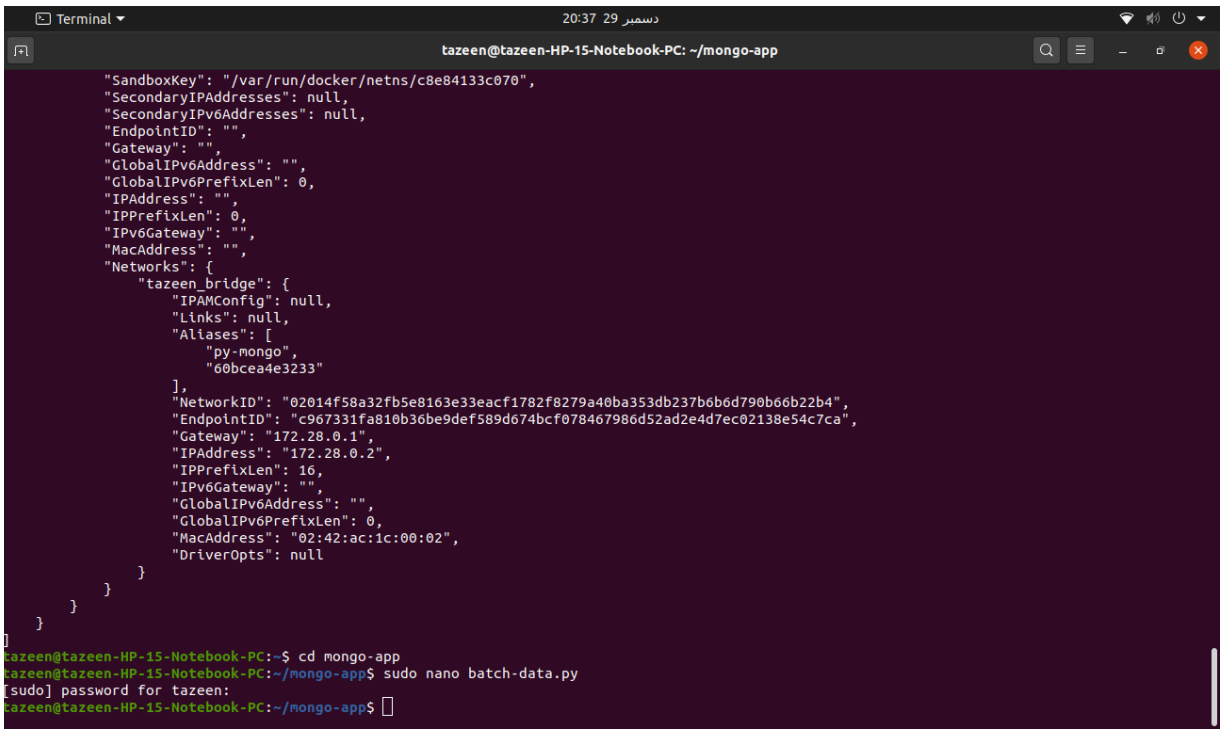




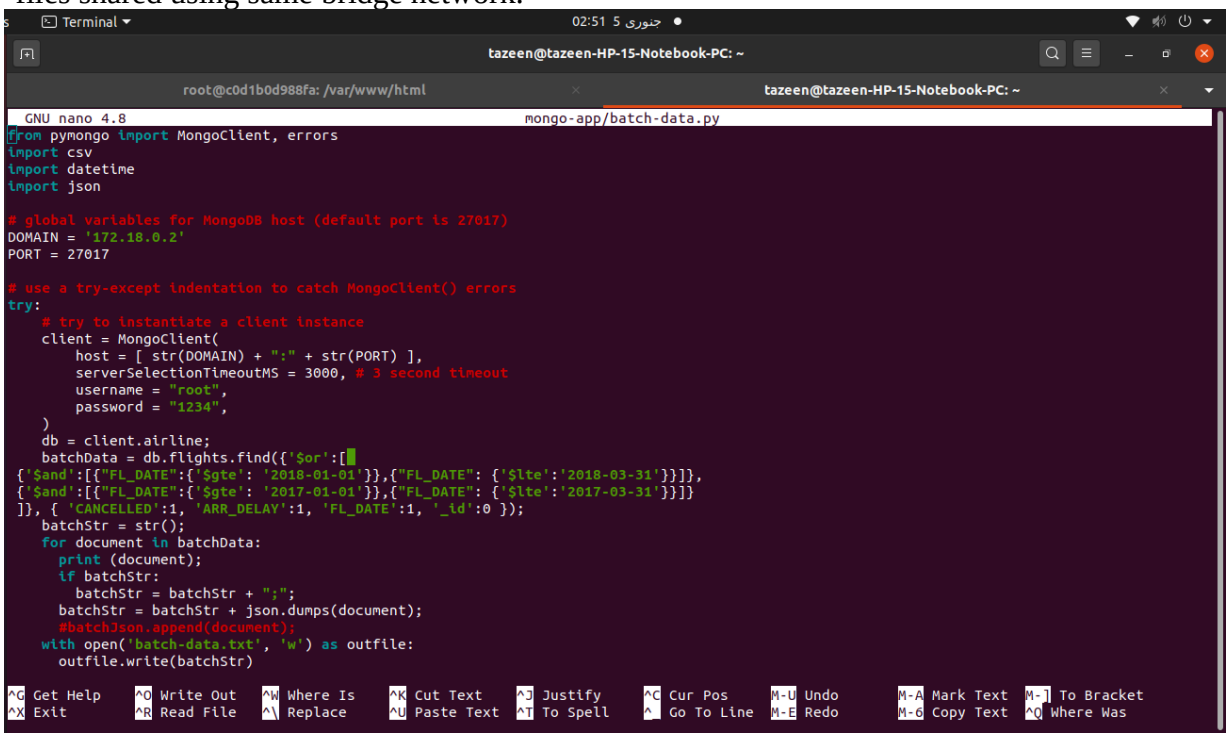
Now getting data for hadoop batch processing from mongodb. First checking the IP address to connect to database using pymongo container

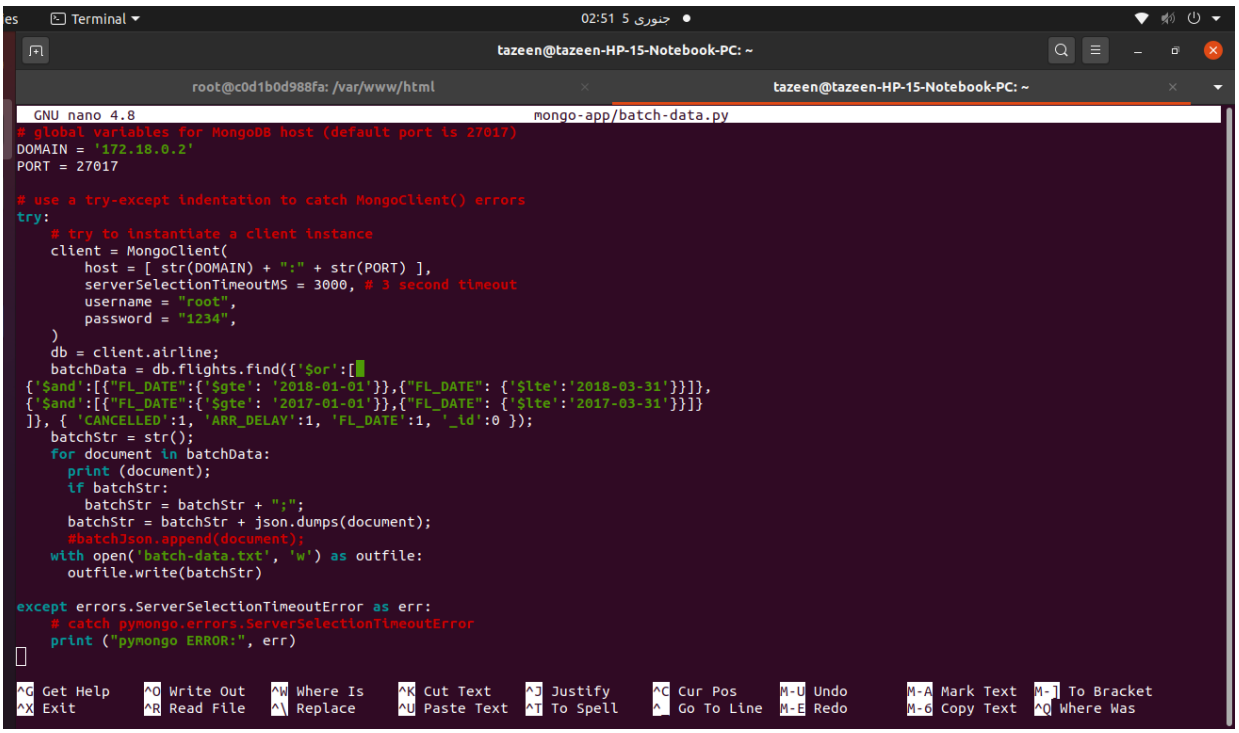
.



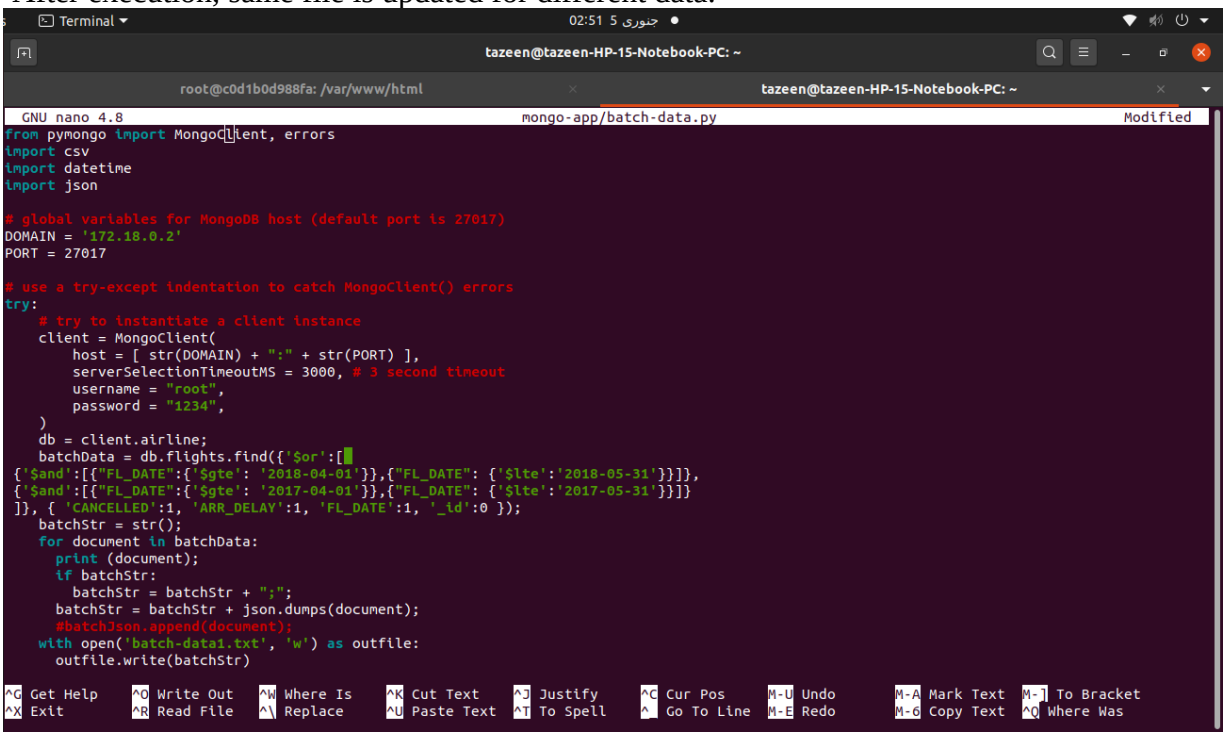


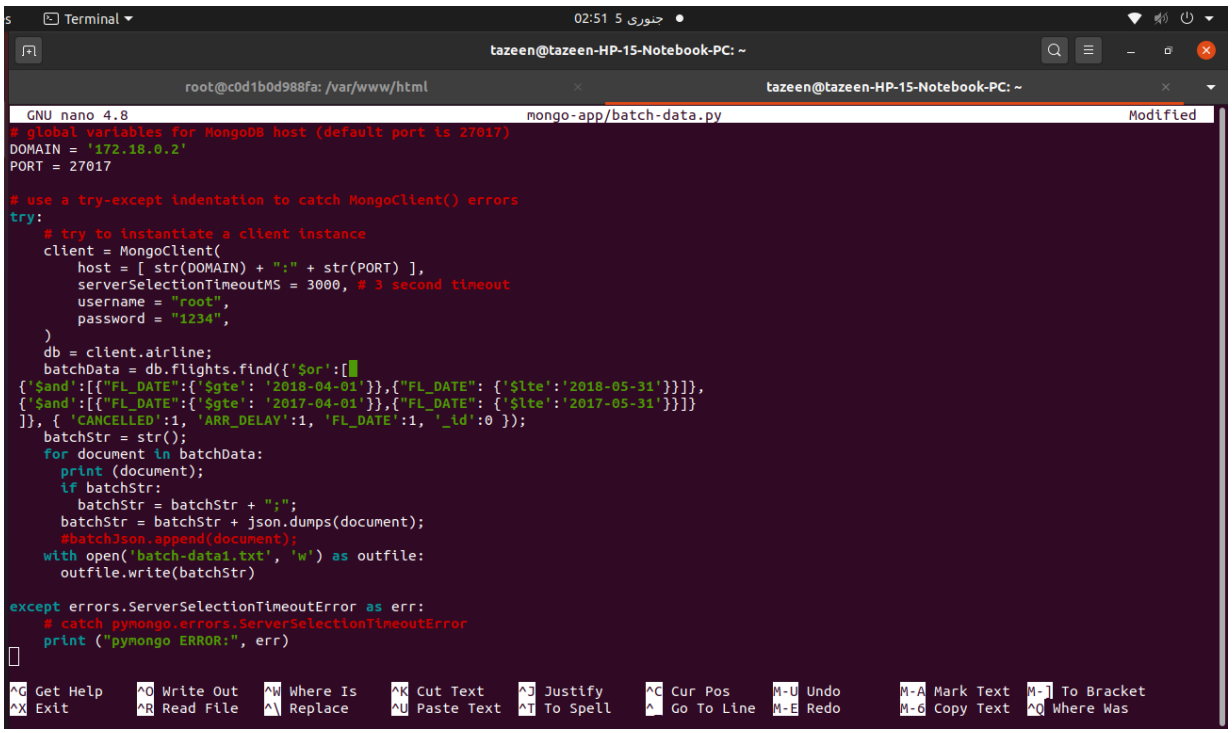
Creating multiple python scripts to export mongodb data for hadoop container. Because scripts take a lot of time in processing and stuck due to less resources. Each script creates text file data for specific months, total data is used from 1/1/2017 – 31/5/2017 and 1/1/2018 – 31/5/2018. The screenshots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



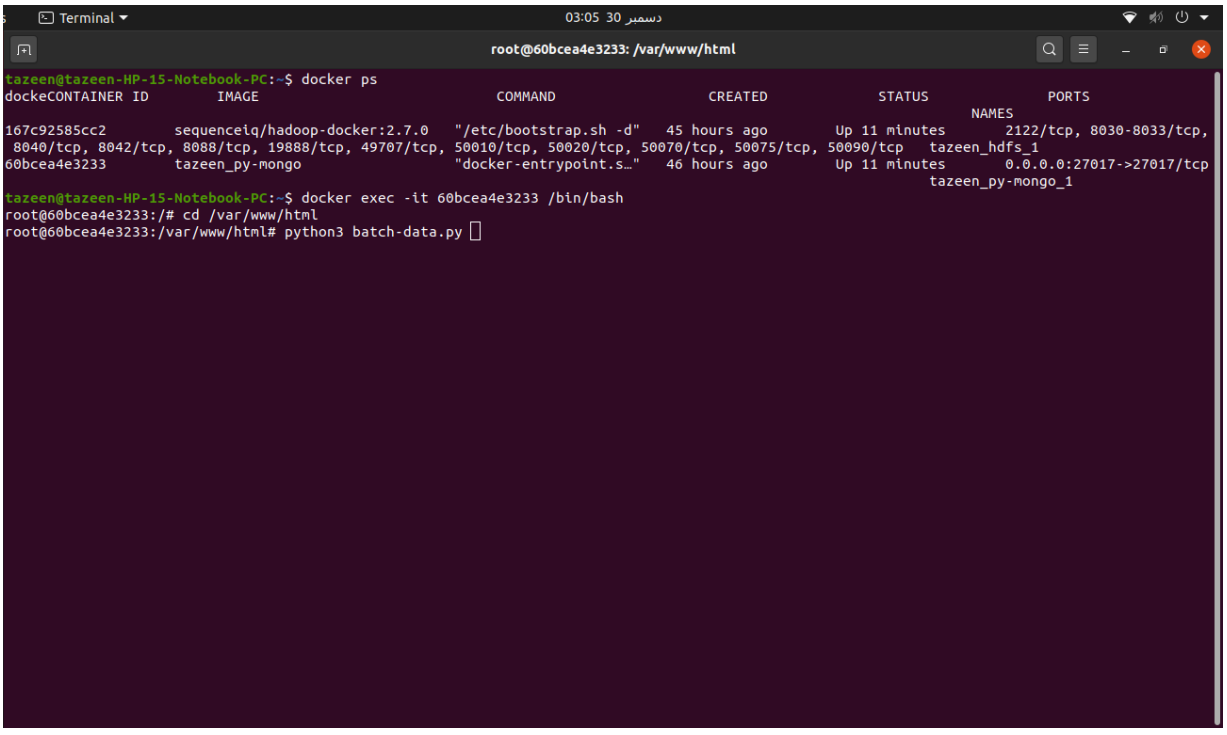


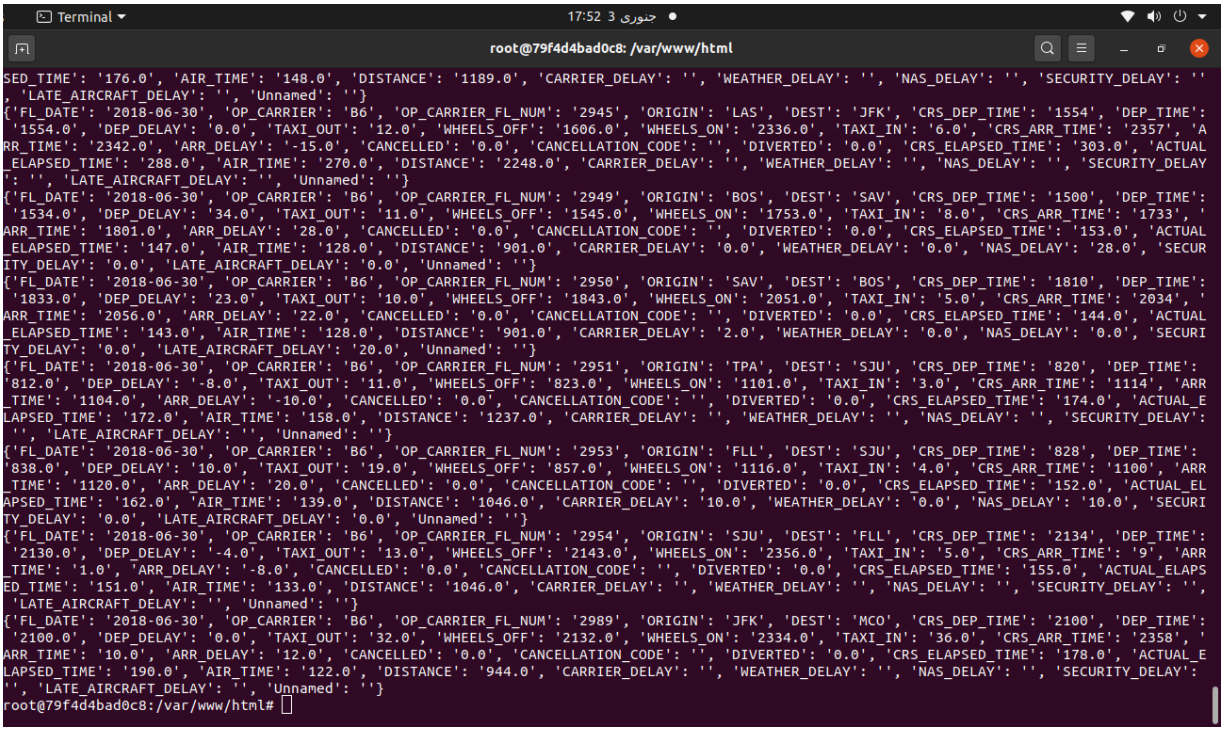
After execution, same file is updated for different data.





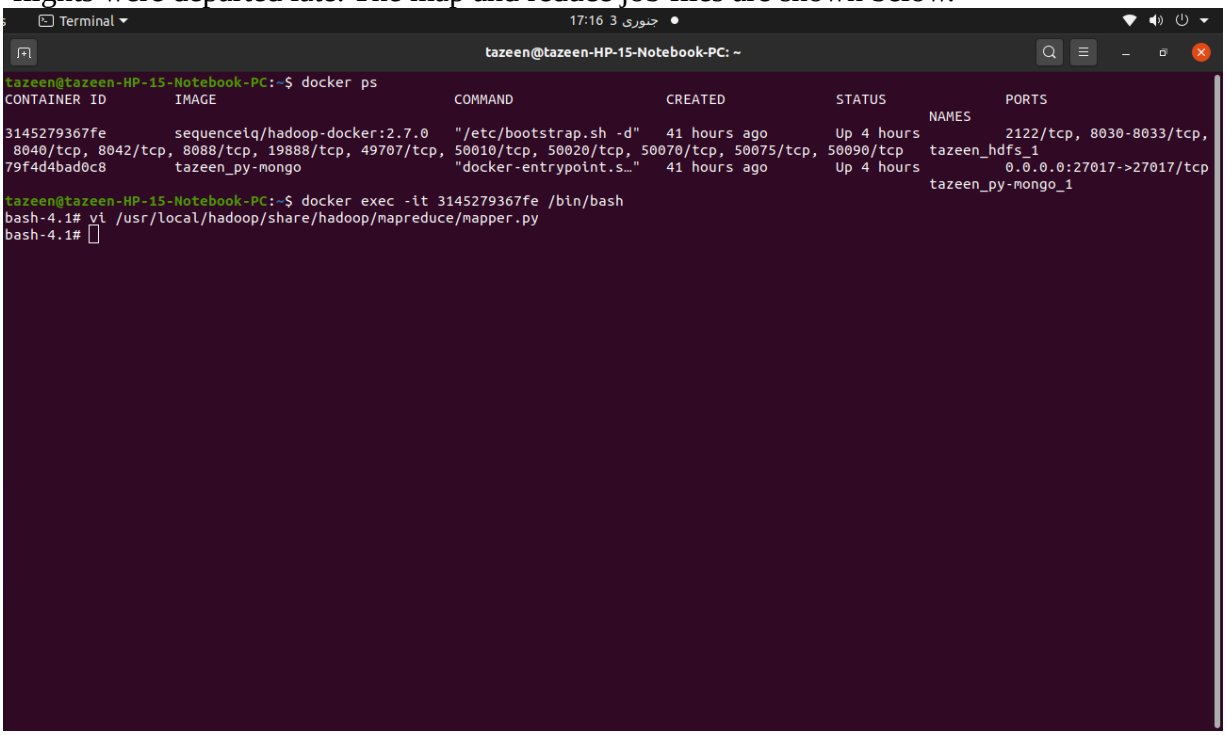
Executing the scripts by the following command

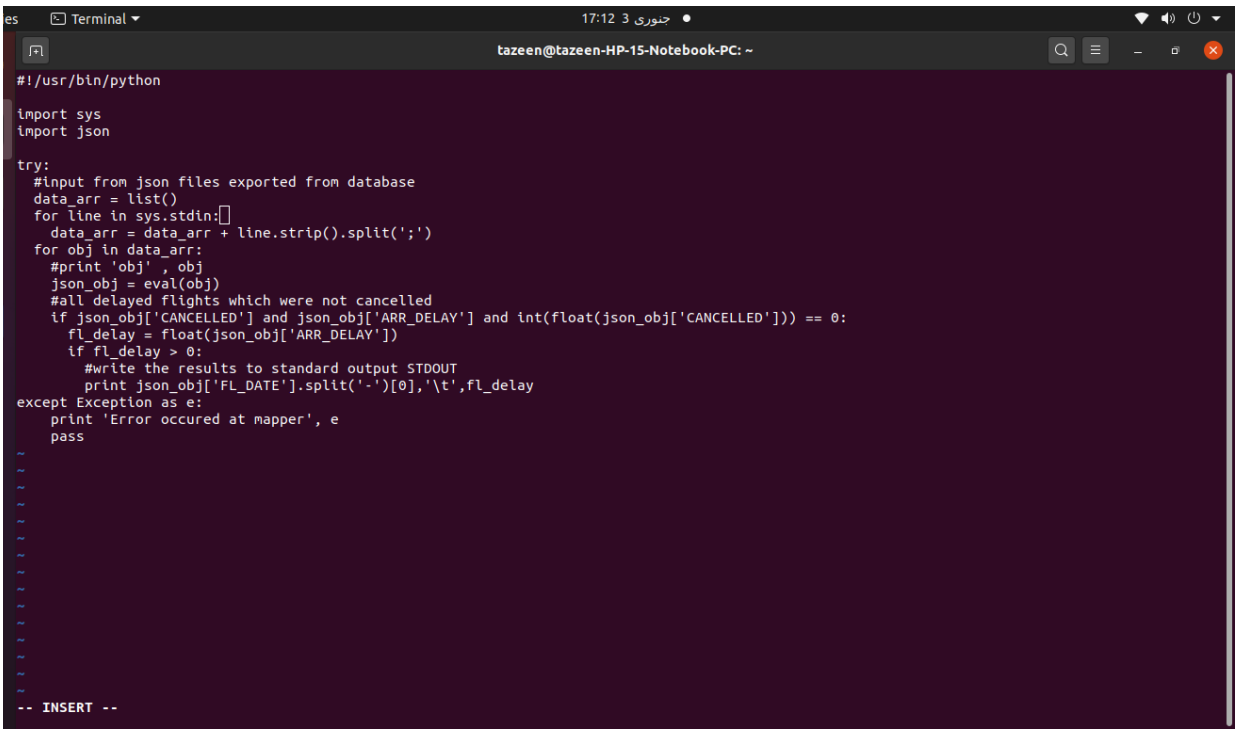


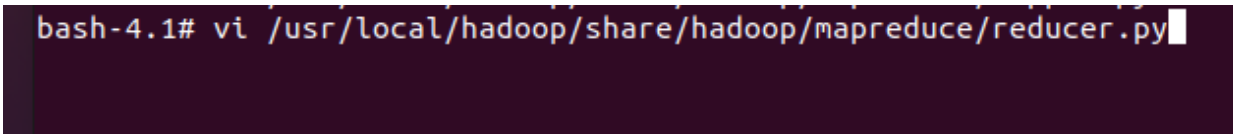


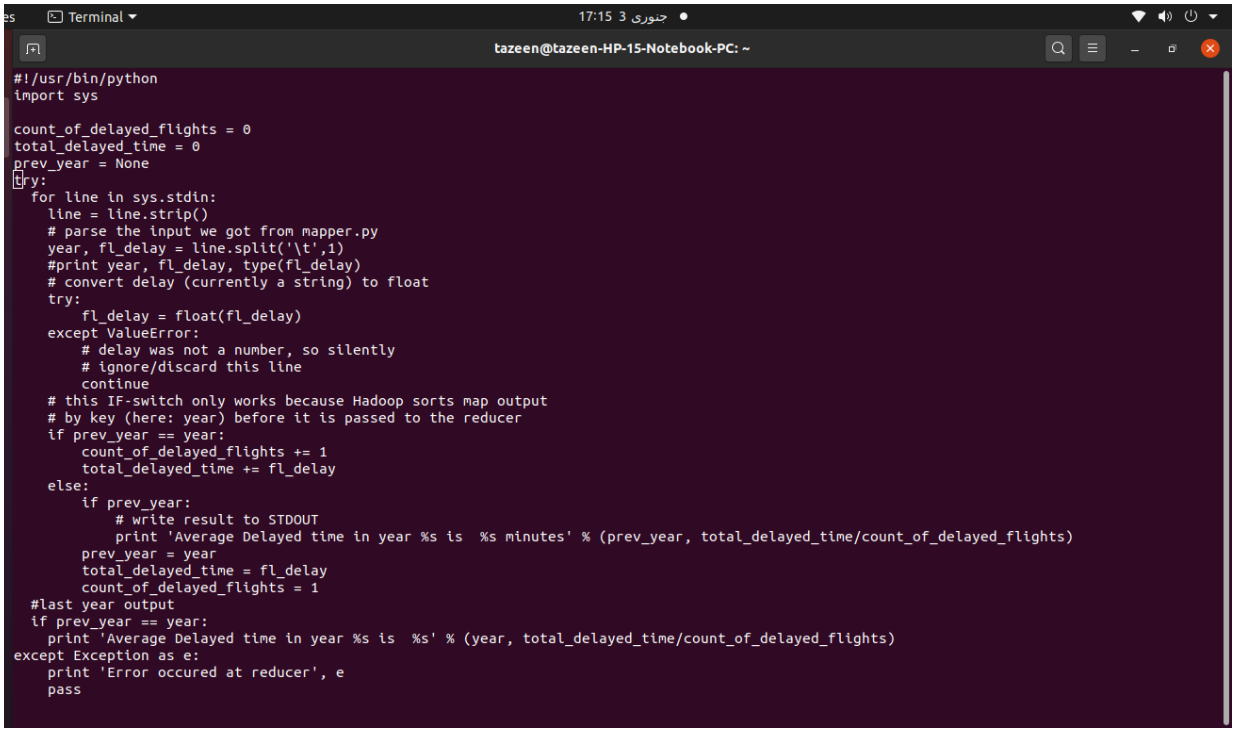
After executing the batch-data.py script with different data, two files batch-data.txt and batch-data1.txt are created which will be used in HDFS.

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.

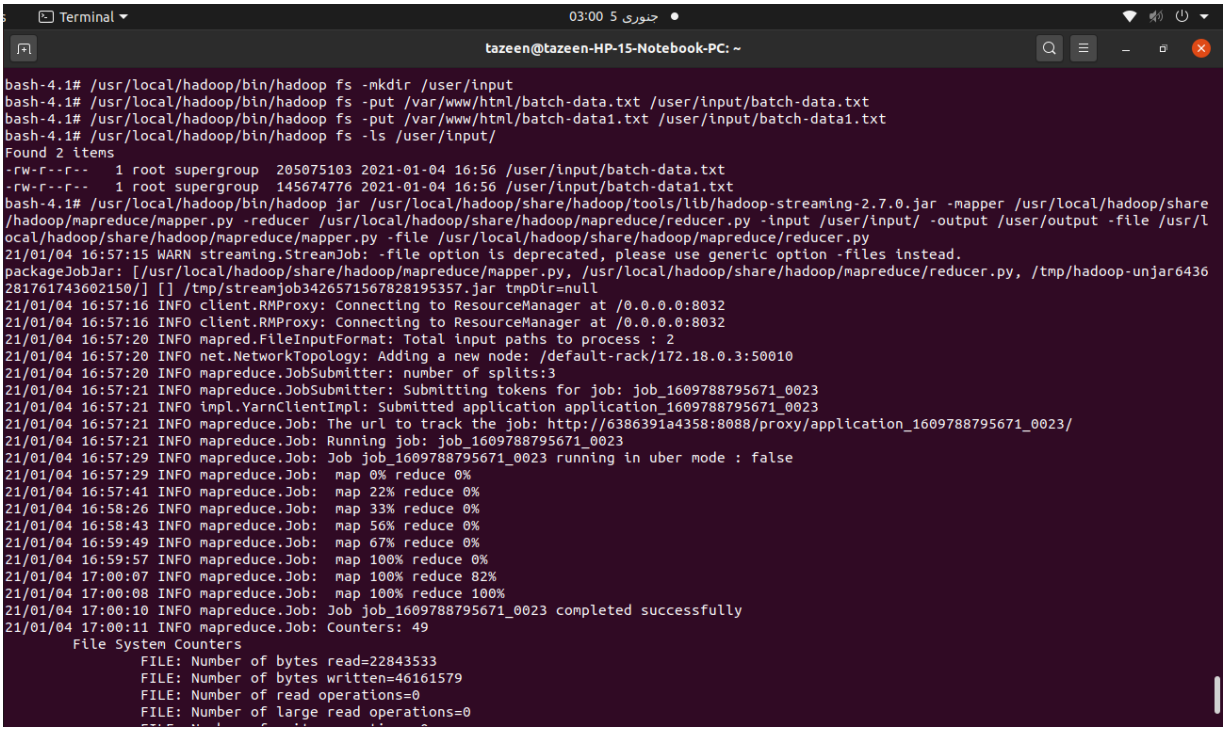




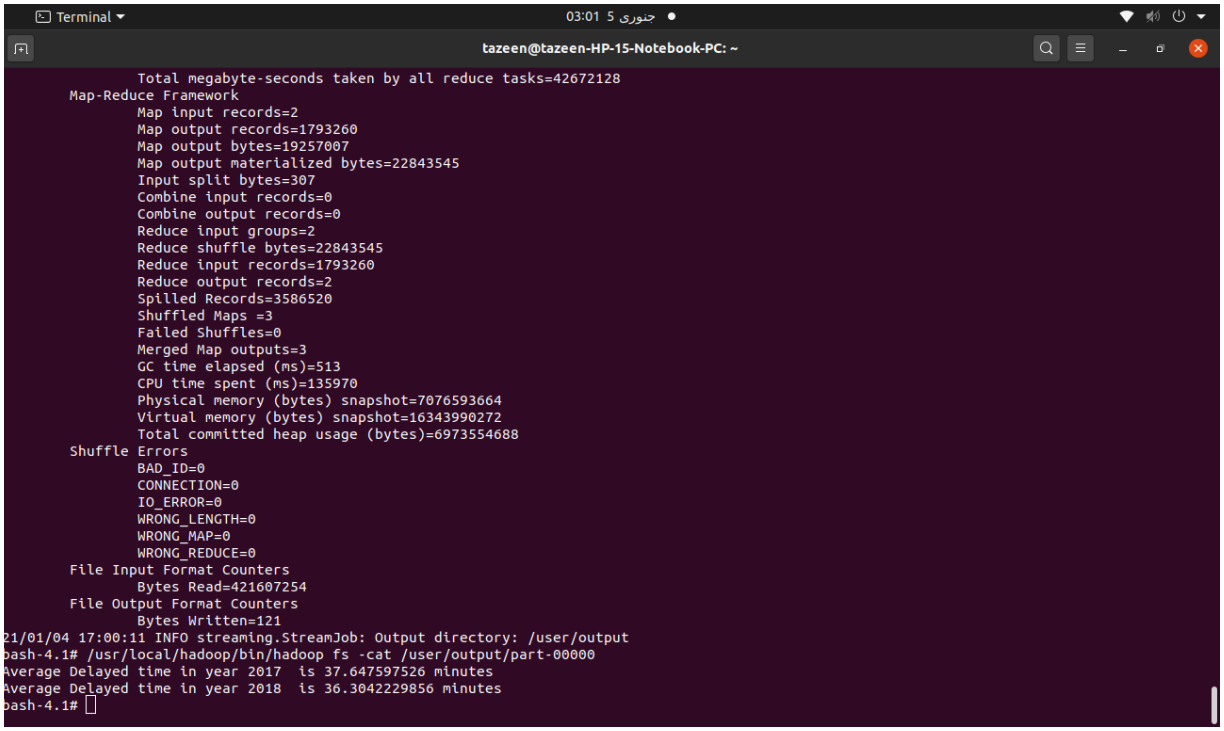




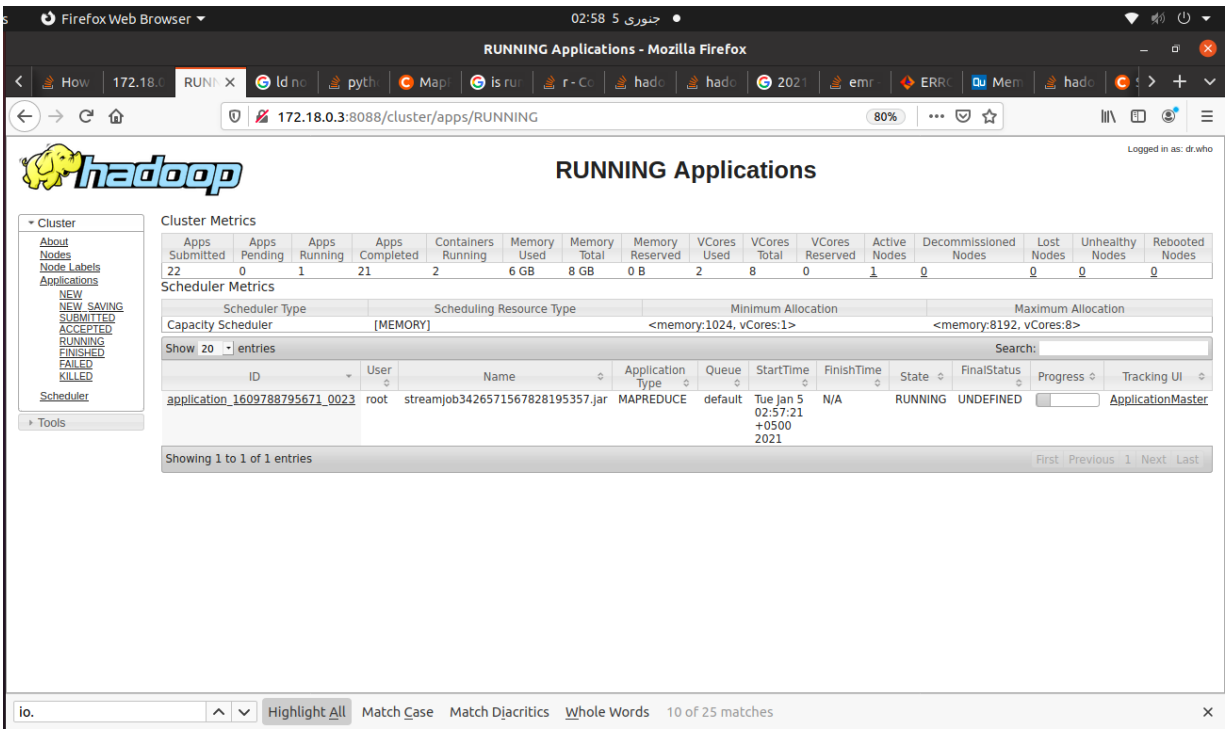
Now putting these batch data files (batch-data.txt and batch-data1.txt) in HDFS. A new folder is created as /user/input which will be for hadoop input data and output will be saved in /user/output.

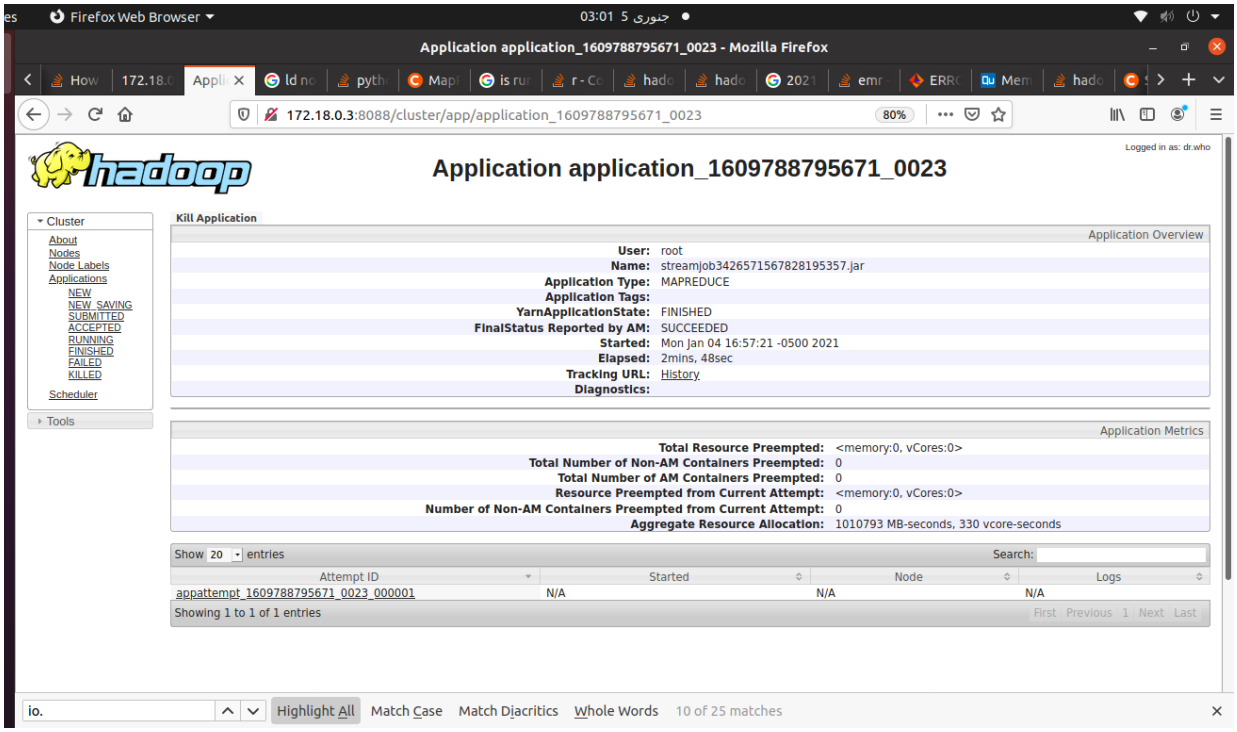


The job is successfully executed and output is saved in /user/output. The output can be viewed as ‘cat /user/output/part-00000.txt’ which contains the output on success.



Navigating to hadoop url to see the applications as below:



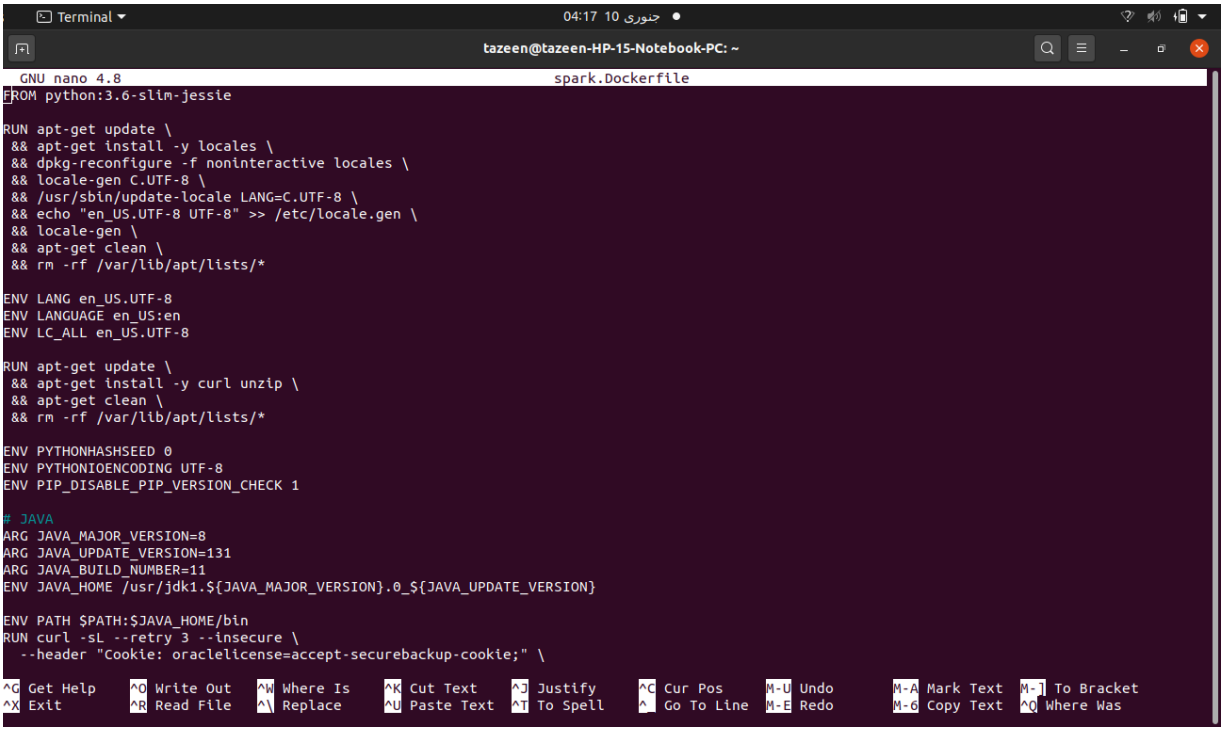


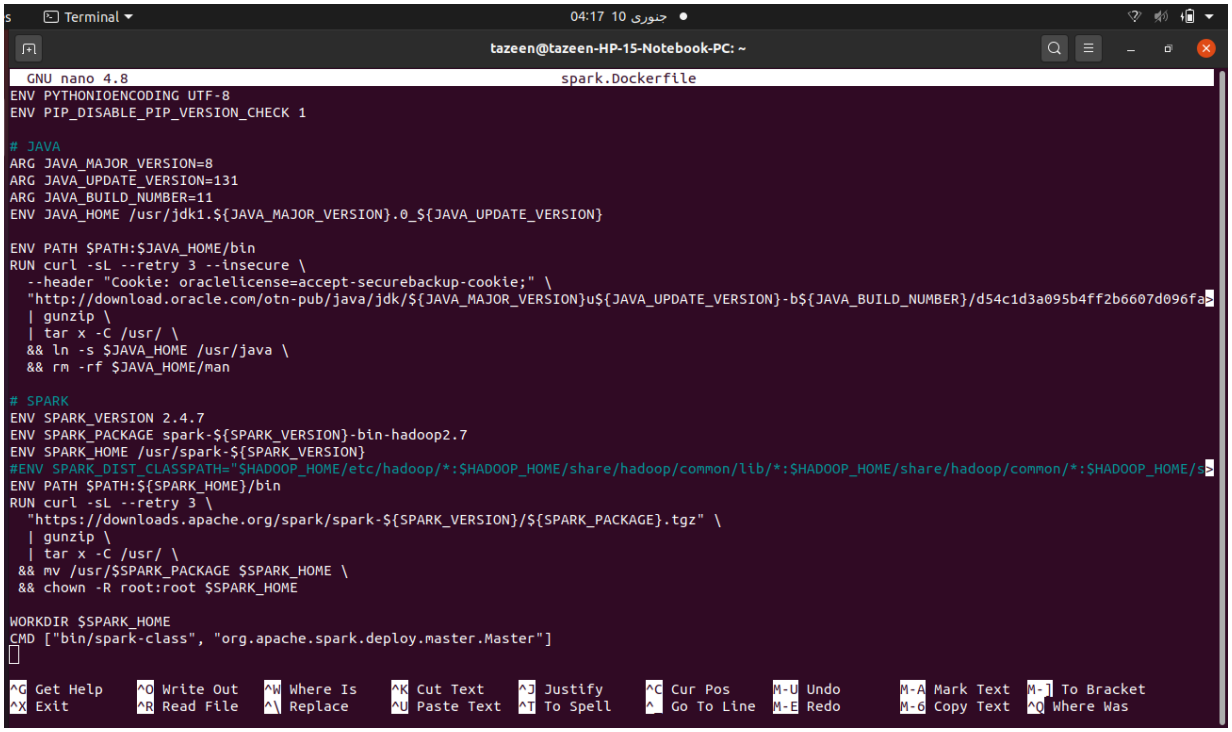
III- Streaming Layer

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:

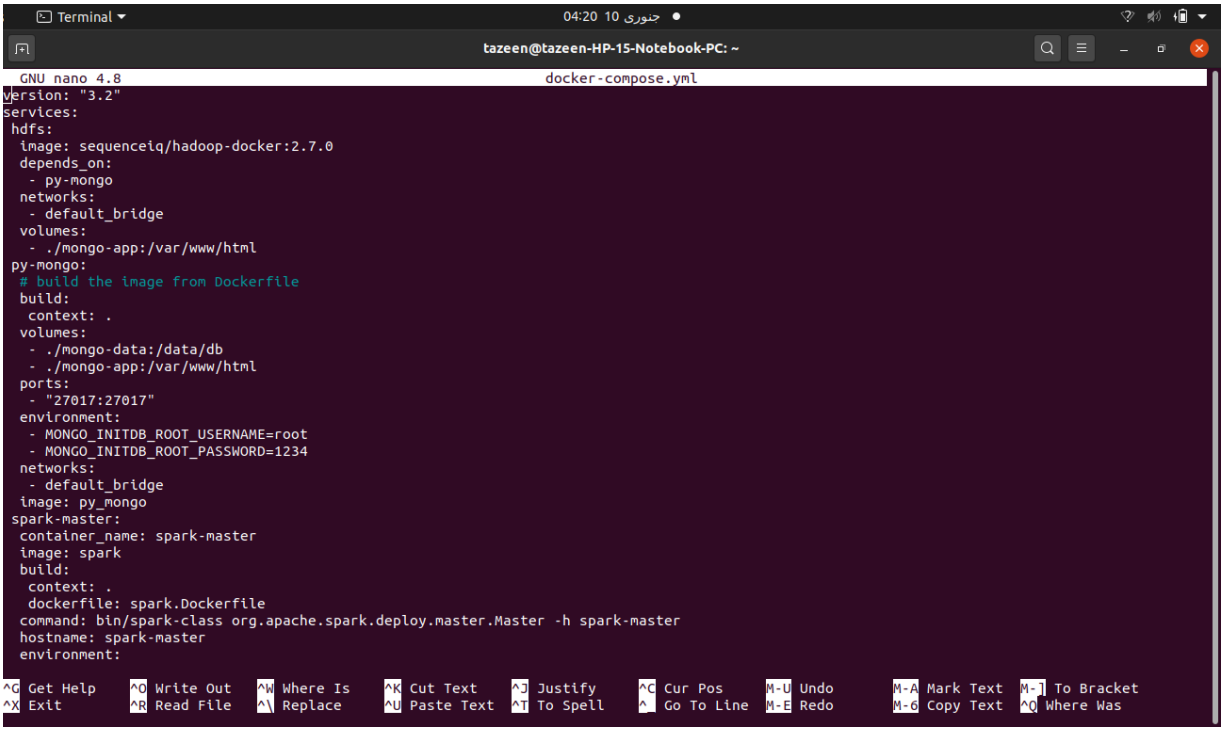
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano spark.Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```

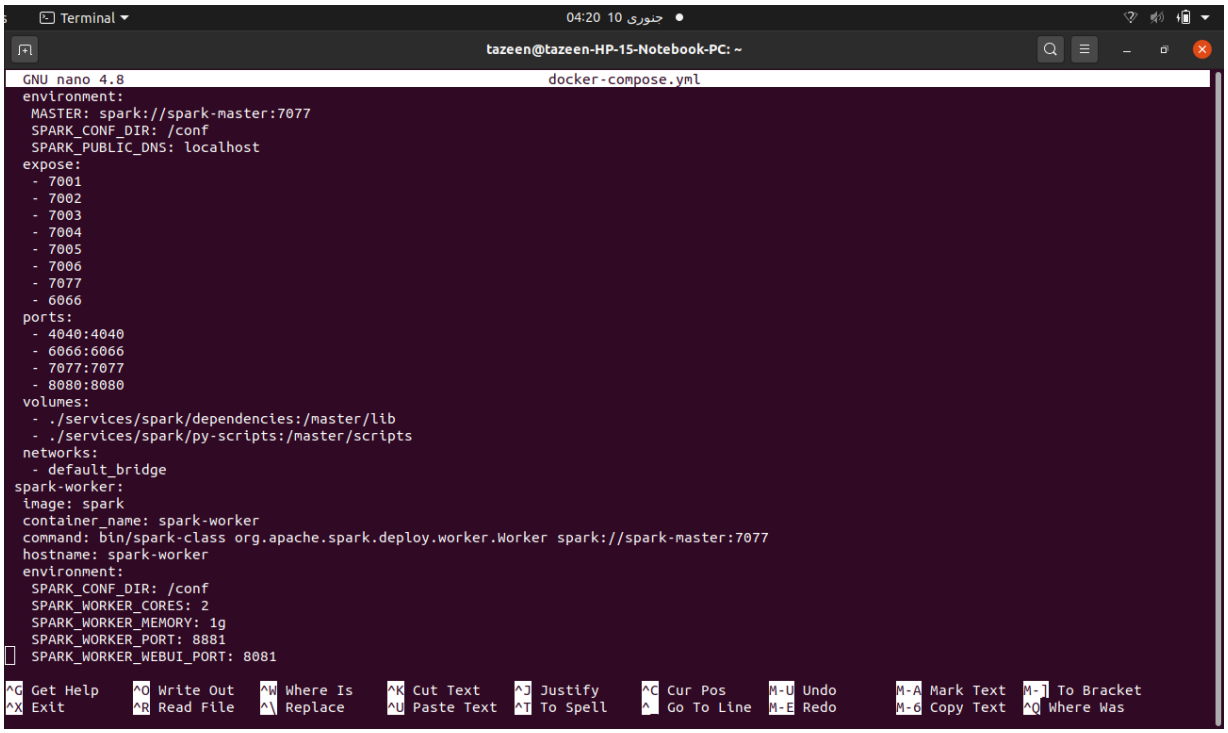


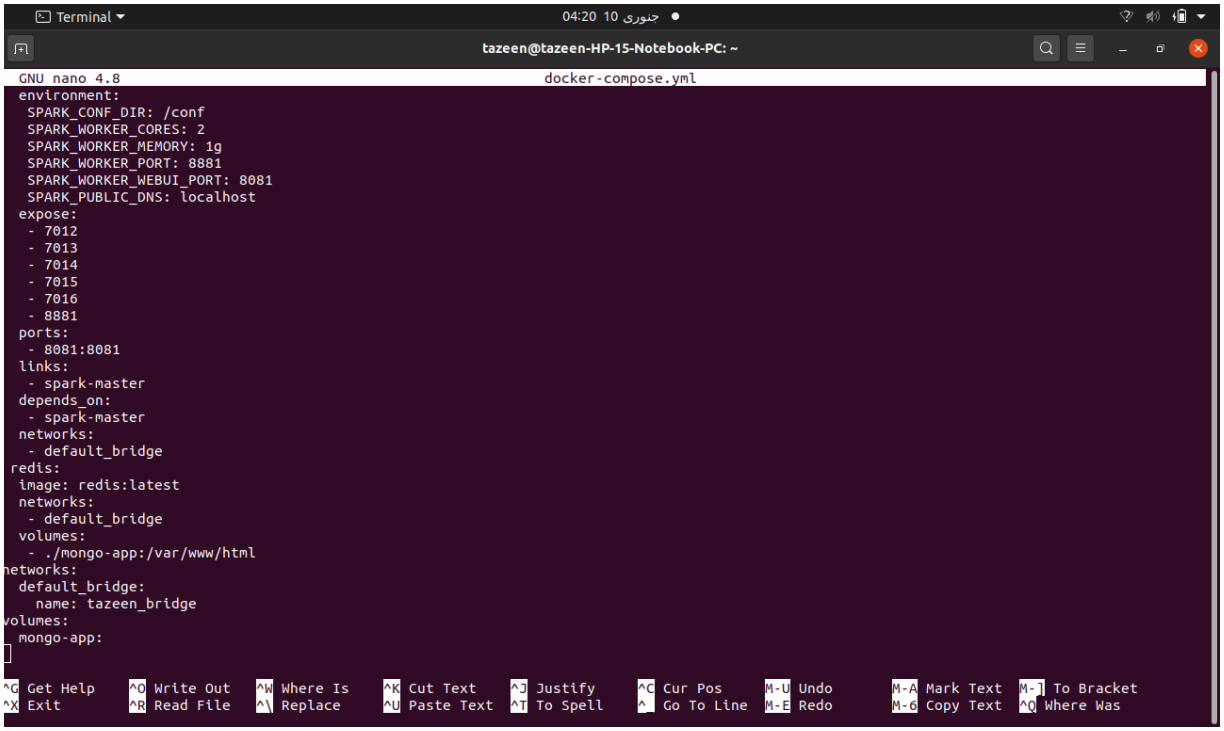


Updating the docker-compose.yml file for streaming layer. Spark master and worker docker are created as separate dockers and another docker is created for redis.

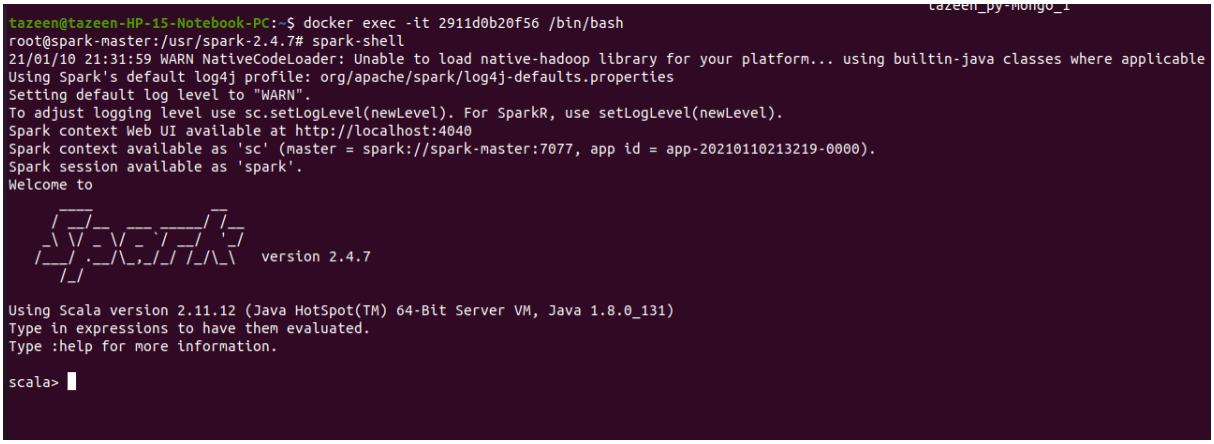
```
tazeen@tazeen-HP-15-Notebook-PC:~$ sudo nano spark.Dockerfile
tazeen@tazeen-HP-15-Notebook-PC:~$
```


Terminal
04:20 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
GNU nano 4.8 docker-compose.yml
version: "3.2"
services:
hdfs:
image: sequenceiq/hadoop-docker:2.7.0
depends_on:
- py-mongo
networks:
- default_bridge
volumes:
- ./mongo-app:/var/www/html
py-mongo:
build the image from Dockerfile
build:
context: .
volumes:
- ./mongo-data:/data/db
- ./mongo-app:/var/www/html
ports:
- "27017:27017"
environment:
- MONGO_INITDB_ROOT_USERNAME=root
- MONGO_INITDB_ROOT_PASSWORD=1234
networks:
- default_bridge
image: py_mongo
spark-master:
container_name: spark-master
image: spark
build:
context: .
dockerfile: spark.Dockerfile
command: bin/spark-class org.apache.spark.deploy.master.Master -h spark-master
hostname: spark-master
environment:
^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

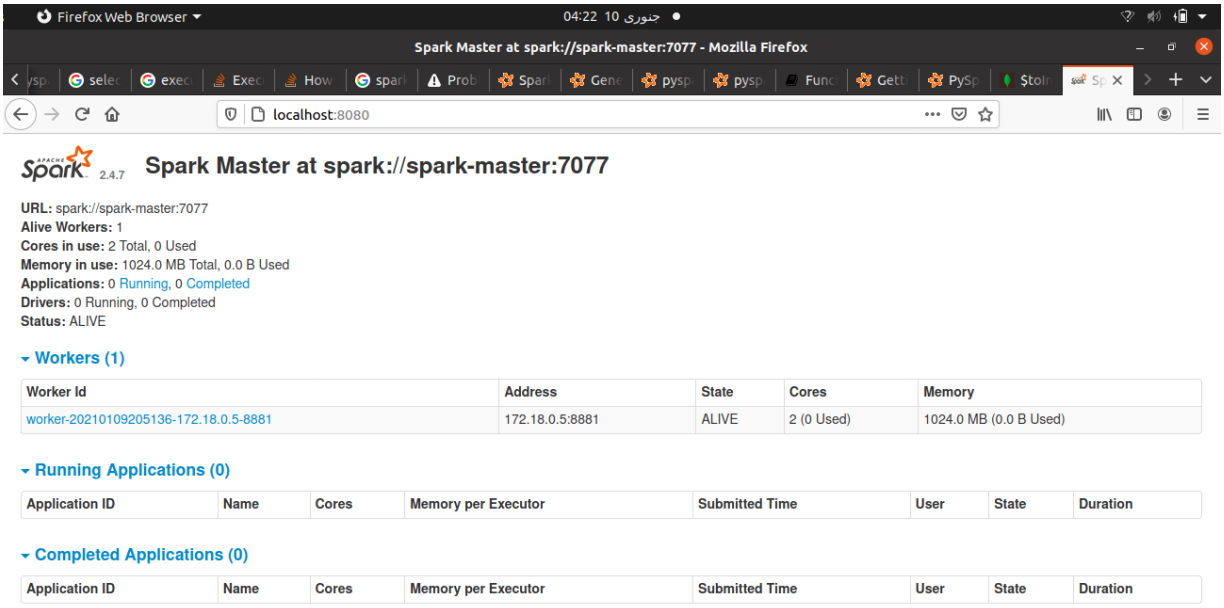

Terminal
04:20 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
GNU nano 4.8 docker-compose.yml
environment:
MASTER: spark://spark-master:7077
SPARK_CONF_DIR: /conf
SPARK_PUBLIC_DNS: localhost
expose:
- 7001
- 7002
- 7003
- 7004
- 7005
- 7006
- 7077
- 6066
ports:
- 4040:4040
- 6066:6066
- 7077:7077
- 8080:8080
volumes:
- ./services/spark/dependencies:/master/lib
- ./services/spark/py-scripts:/master/scripts
networks:
- default_bridge
spark-worker:
image: spark
container_name: spark-worker
command: bin/spark-class org.apache.spark.deploy.worker.Worker spark://spark-master:7077
hostname: spark-worker
environment:
SPARK_CONF_DIR: /conf
SPARK_WORKER_CORES: 2
SPARK_WORKER_MEMORY: 1g
SPARK_WORKER_PORT: 8881
SPARK_WORKER_WEBUI_PORT: 8081
^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

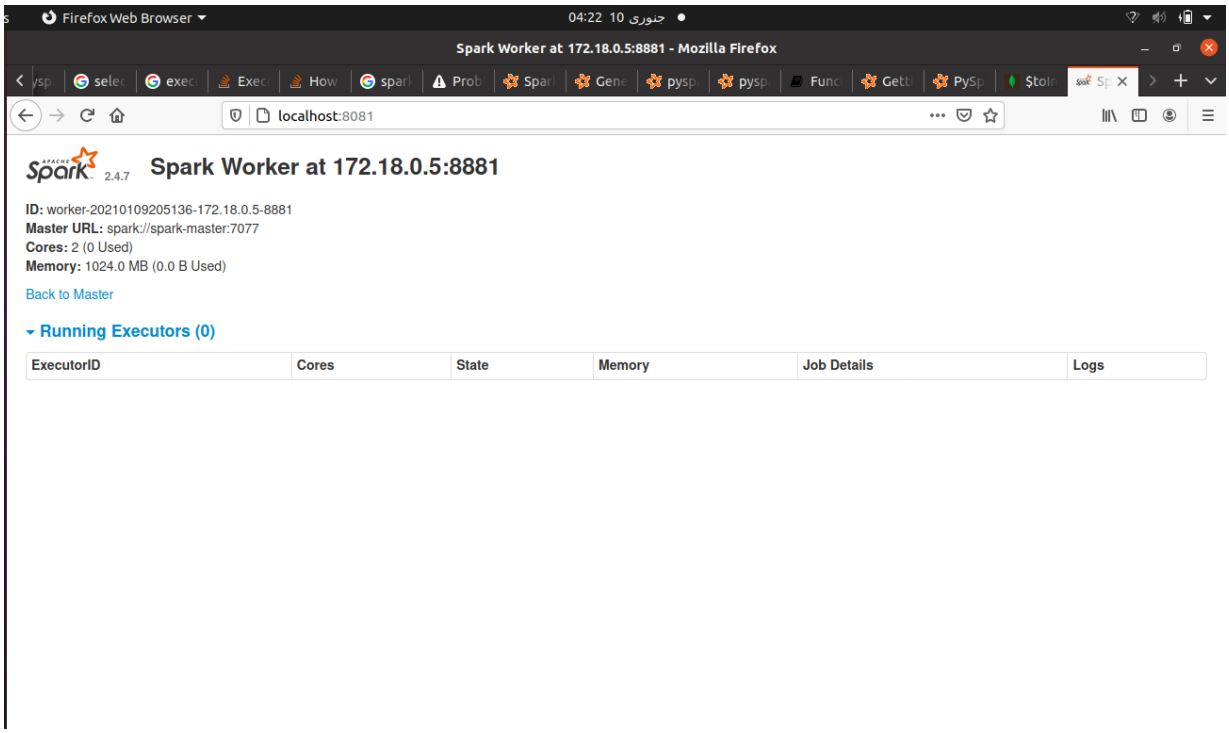


Spark is configured correctly and opening spark master terminal:

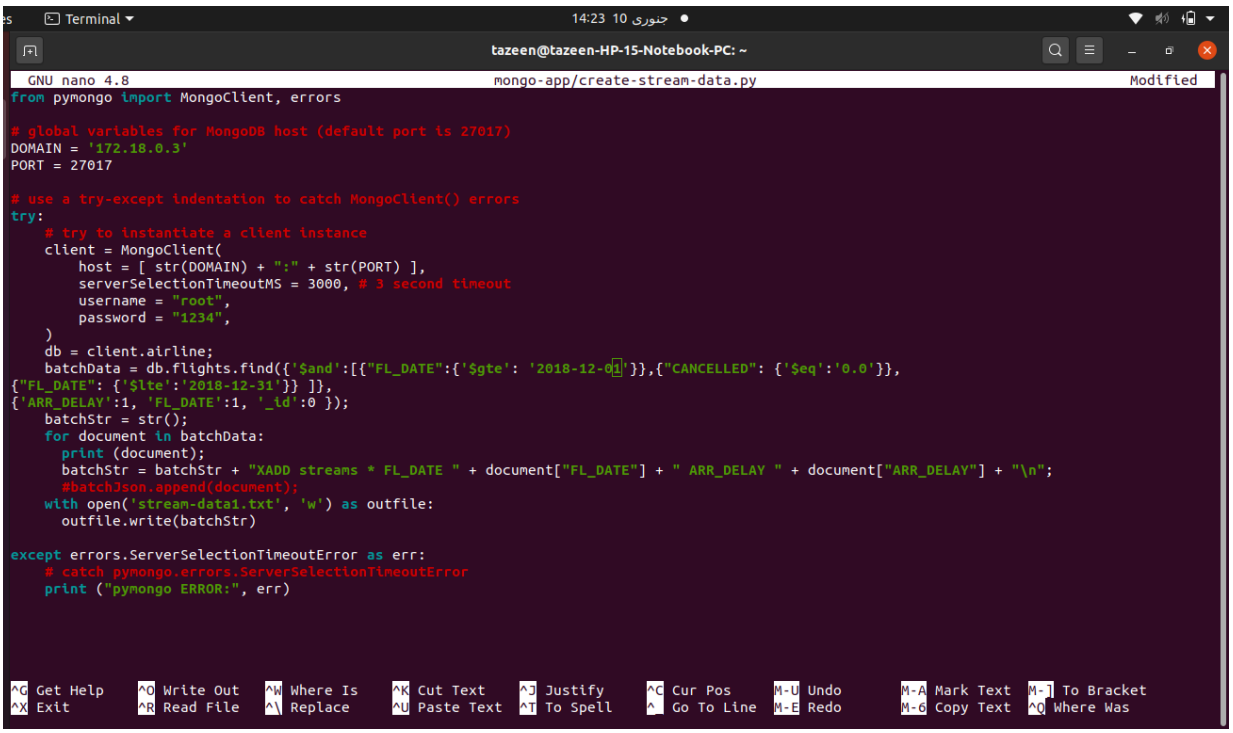


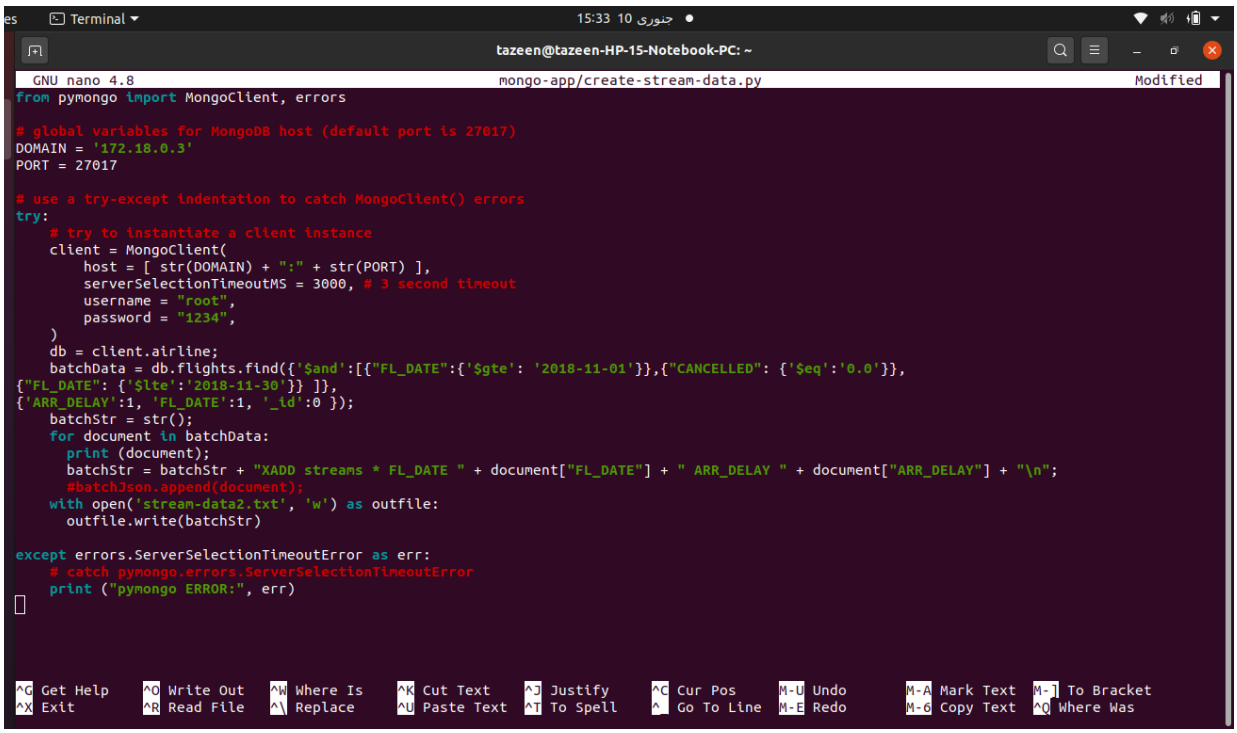
Navigate to localhost:8080 to see spark master is running.



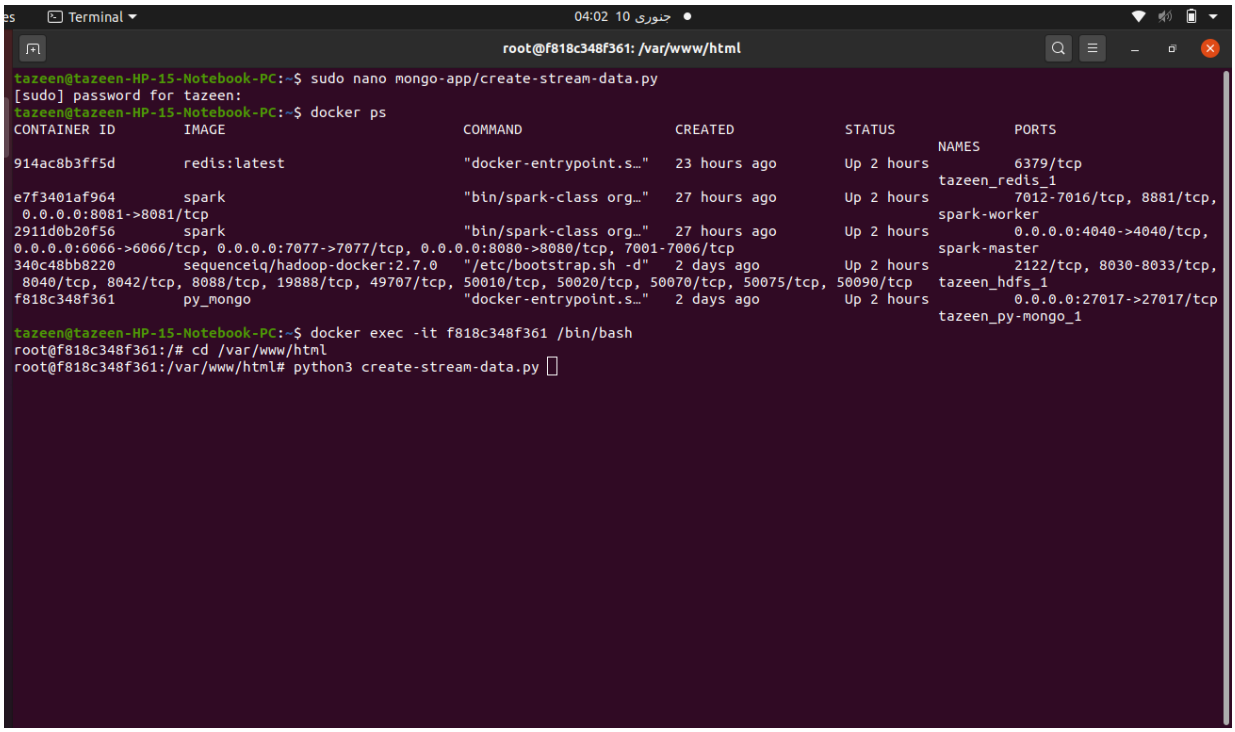


To create streaming data, a text file is generated with redis streams from mongodb database. The data for November and December 2018. The mongodb script is saved as mongo-app/create-stream-data.py to generate data is below. I will only find the data for the flights which are not canceled, therefore,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


Terminal
14:23 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
GNU nano 4.8
mongo-app/create-stream-data.py
Modified
from pymongo import MongoClient, errors
# global variables for MongoDB host (default port is 27017)
DOMAIN = '172.18.0.3'
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;
batchData = db.flights.find({'$and':[{"FL_DATE":{'$gte': '2018-12-01'}},{"CANCELLED": {'$eq':'0.0'}},
{"FL_DATE": {'$lte':'2018-12-31'}}]},
{'ARR_DELAY':1, 'FL_DATE':1, '_id':0 });
batchStr = str();
for document in batchData:
print (document);
batchStr = batchStr + "XADD streams * FL_DATE " + document["FL_DATE"] + " ARR_DELAY " + document["ARR_DELAY"] + "\n";
#batchJson.append(document);
with open('stream-data1.txt', 'w') as outfile:
outfile.write(batchStr)
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

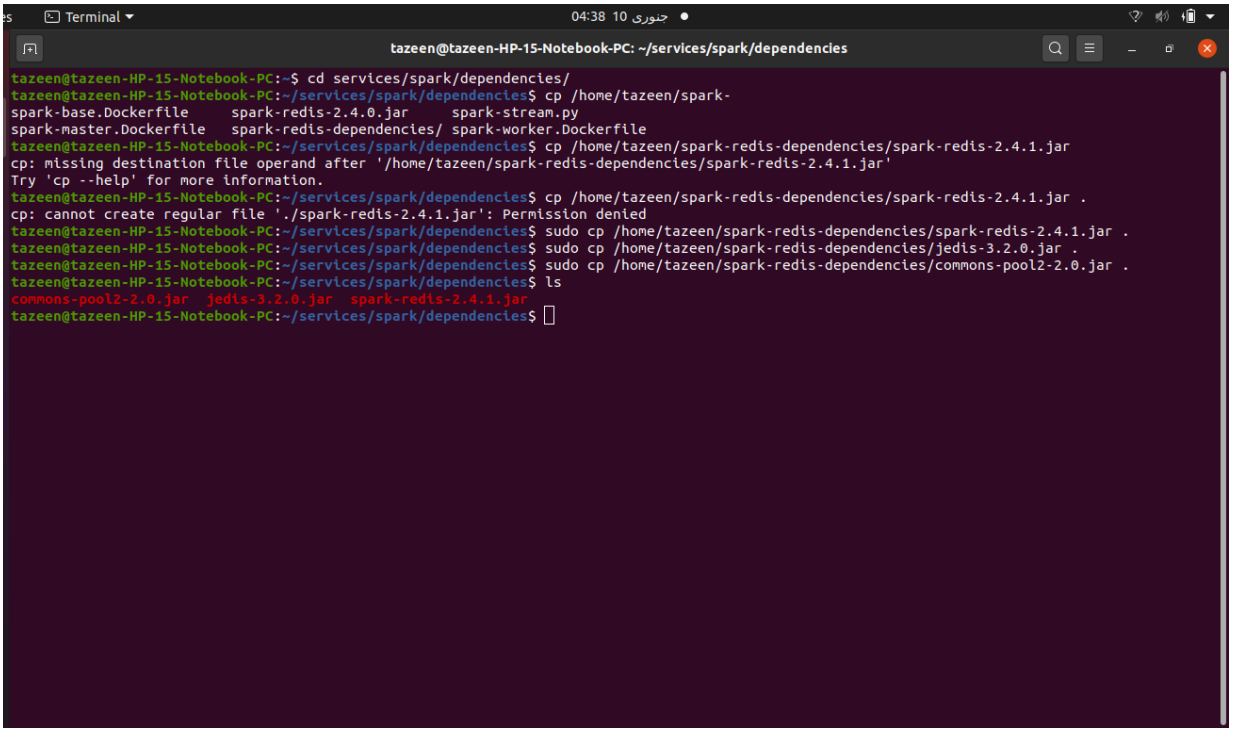


Now executing the script in mongo db docker container. It will create two stream data files as tream-data1.txt and stream-data2.txt. These files will be used by redis docker container to generate streams.

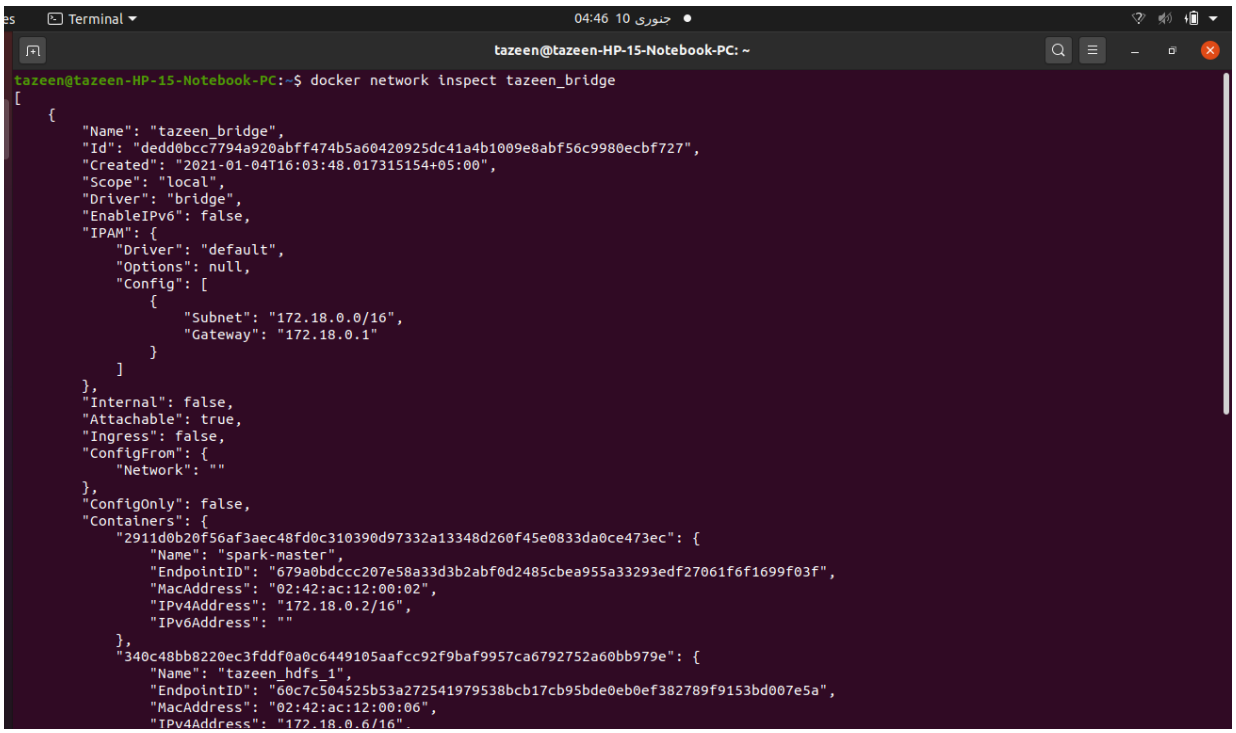


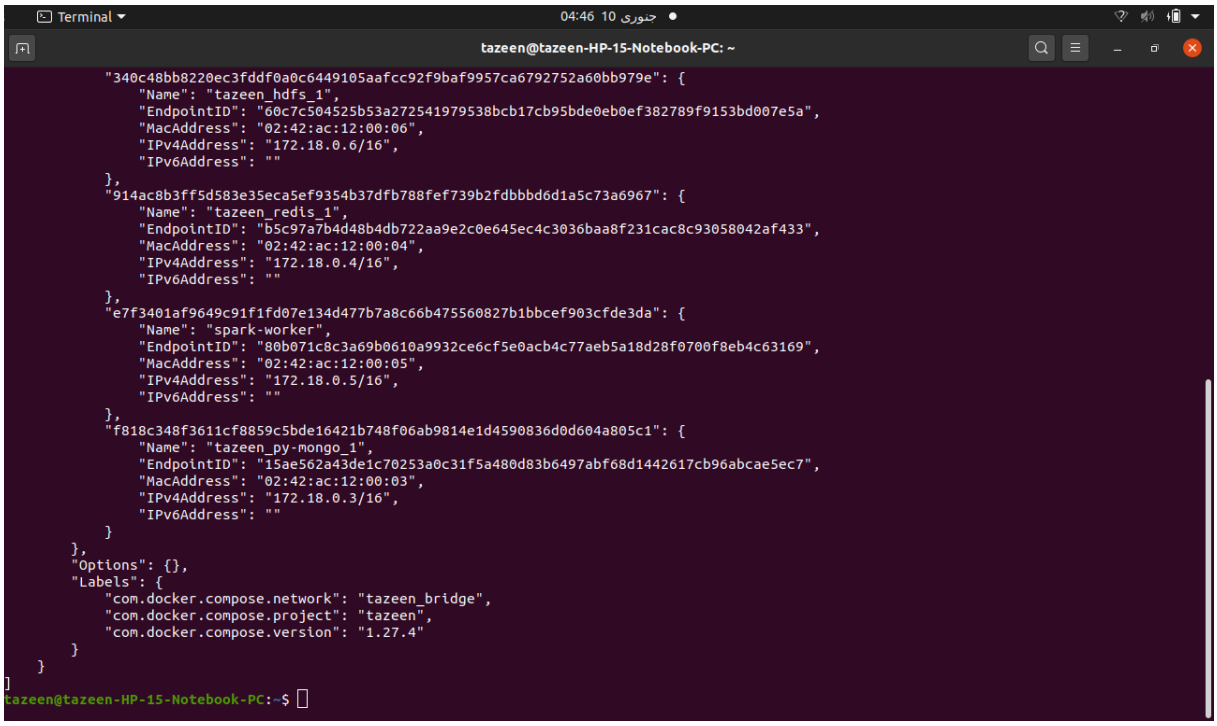
Let’s execute the script to execute streaming data from redis to spark master. The dependencies required for spark is mounted on /master/lib folder in spark master. These dependencies consists of jar files required to read redis streams in pyspark.

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

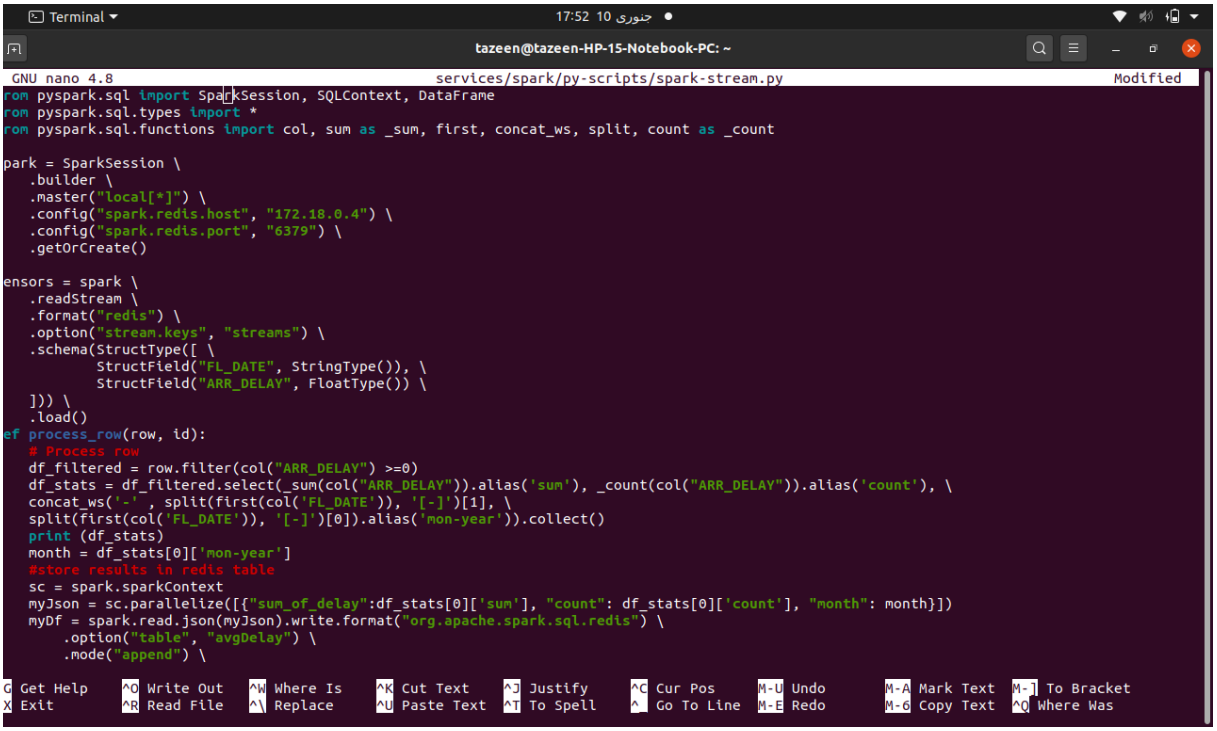


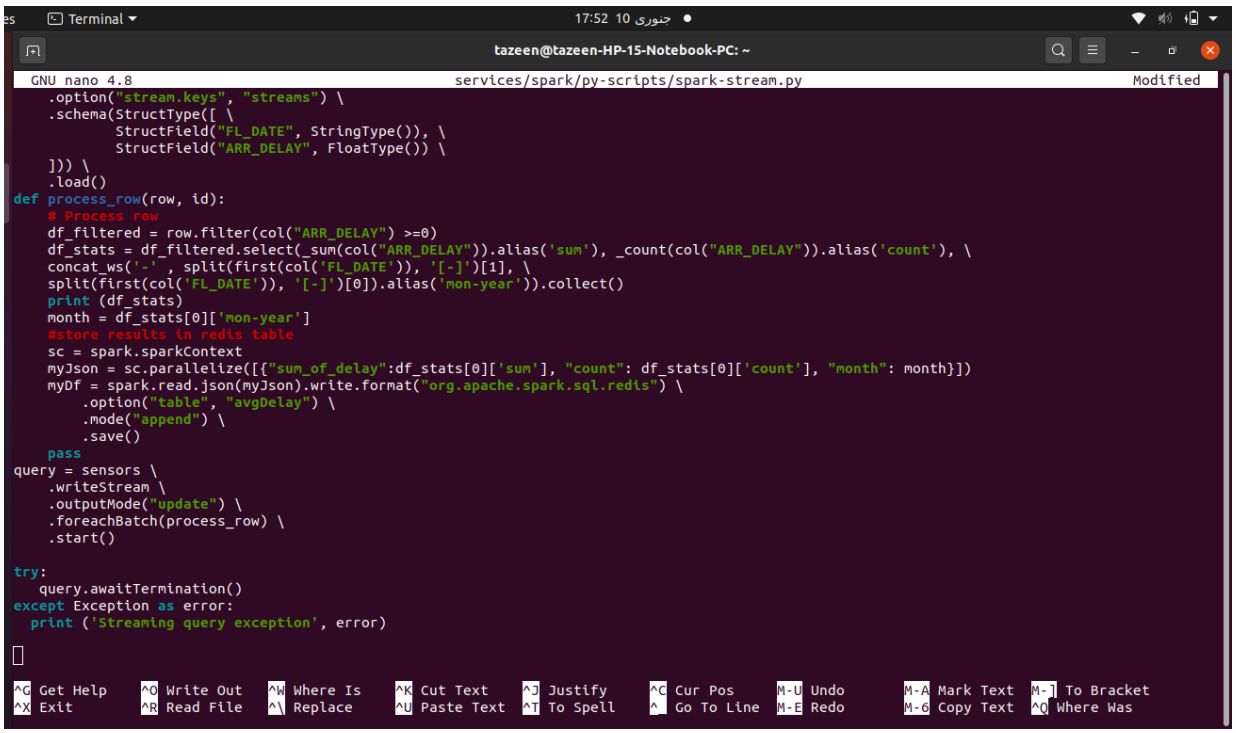
The python script to execute streaming reads streaming data from redis host, it’s host ip can be seen from bridge network. For redis-docker IP address is : 172.18.0.4


Terminal
04:46 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
tazeen@tazeen-HP-15-Notebook-PC:~$ docker network inspect tazeen_bridge
[
{
"Name": "tazeen_bridge",
"Id": "dedd0bcc7794a920abff474b5a60420925dc41a4b1009e8abf56c9980ecbf727",
"Created": "2021-01-04T16:03:48.017315154+05:00",
"Scope": "local",
"Driver": "bridge",
"EnableIPv6": false,
"IPAM": {
"Driver": "default",
"Options": null,
"Config": [
{
"Subnet": "172.18.0.0/16",
"Gateway": "172.18.0.1"
}
]
},
"Internal": false,
"Attachable": true,
"Ingress": false,
"ConfigFrom": {
"Network": ""
},
"ConfigOnly": false,
"Containers": {
"2911d0b20f56af3aec48fd0c310390d97332a13348d260f45e0833da0ce473ec": {
"Name": "spark-master",
"EndpointID": "679a0bdccc207e58a33d3b2abf0d2485cbea955a33293edf27061f6f1699f03f",
"MacAddress": "02:42:ac:12:00:02",
"IPv4Address": "172.18.0.2/16",
"IPv6Address": ""
},
"340c48bb8220ec3fddf0a0c6449105aafcc92f9baf9957ca6792752a60bb979e": {
"Name": "tazeen_hdfs_1",
"EndpointID": "60c7c504525b53a272541979538bcb17cb95bde0eb0ef382789f9153bd007e5a",
"MacAddress": "02:42:ac:12:00:06",

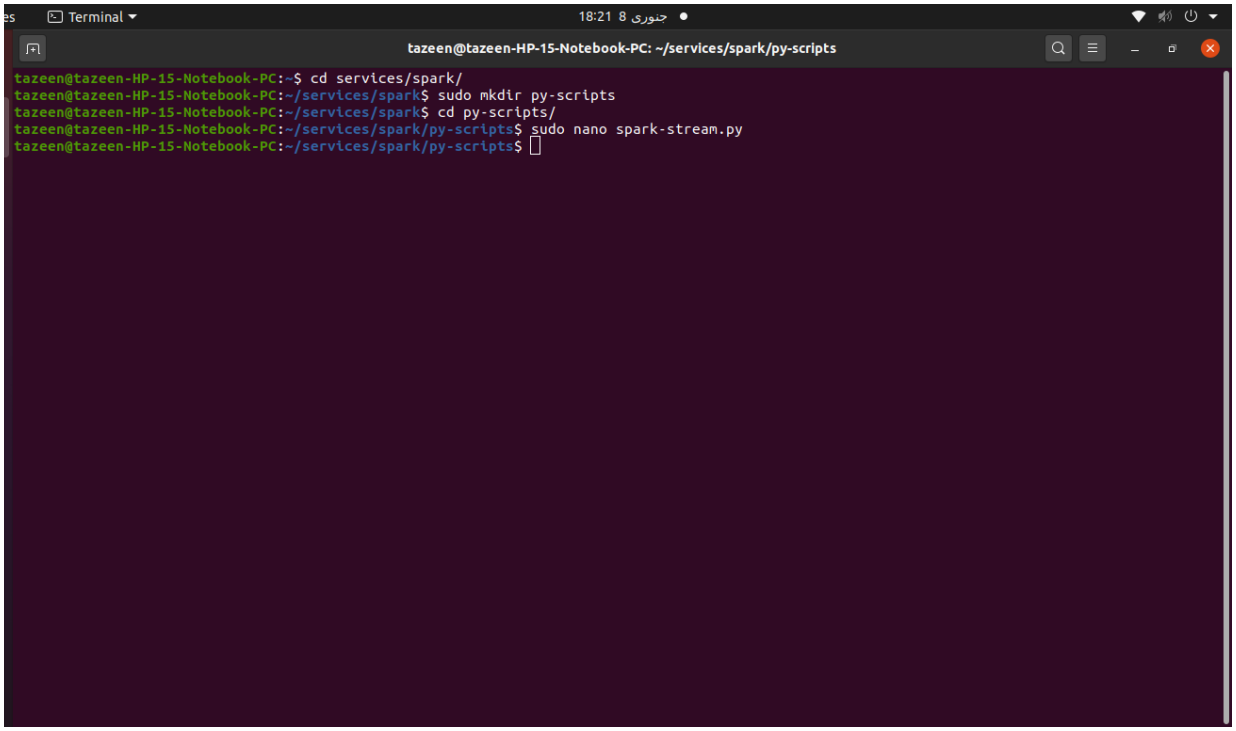


The python script is summing the delay time for positive flight delays, the number of rows provided in each stream and the month of flight date. The script is following:

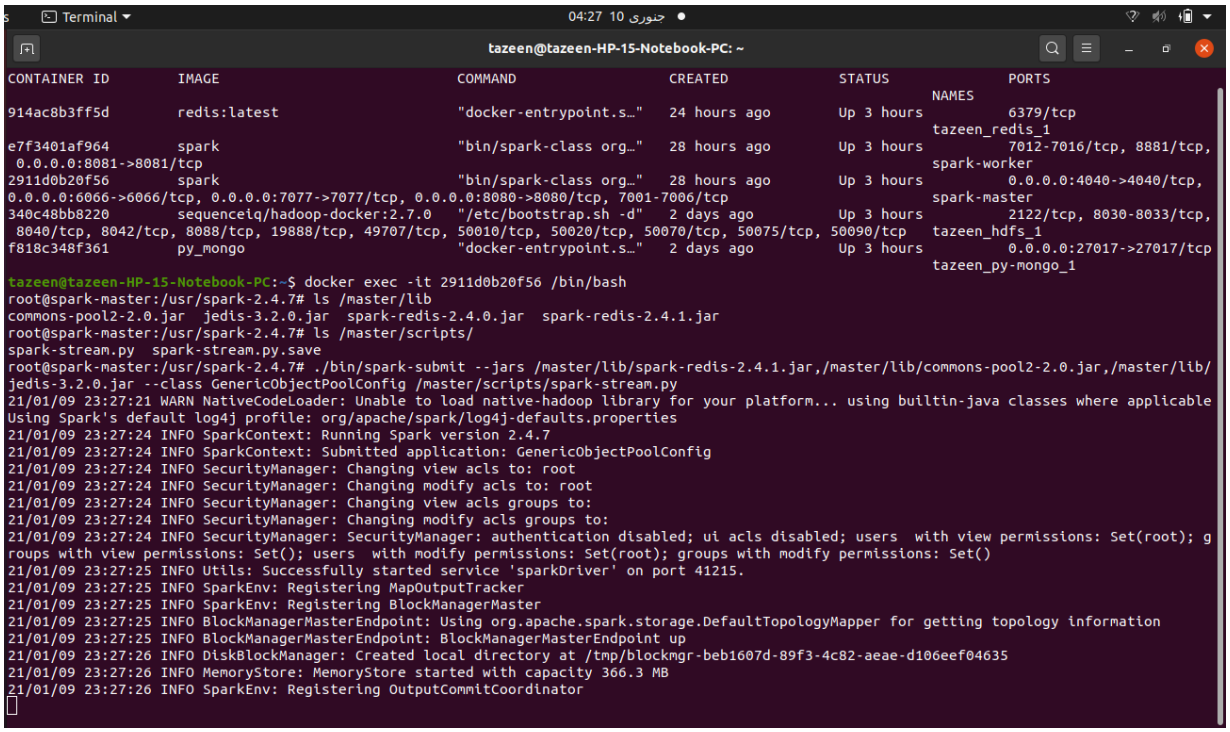




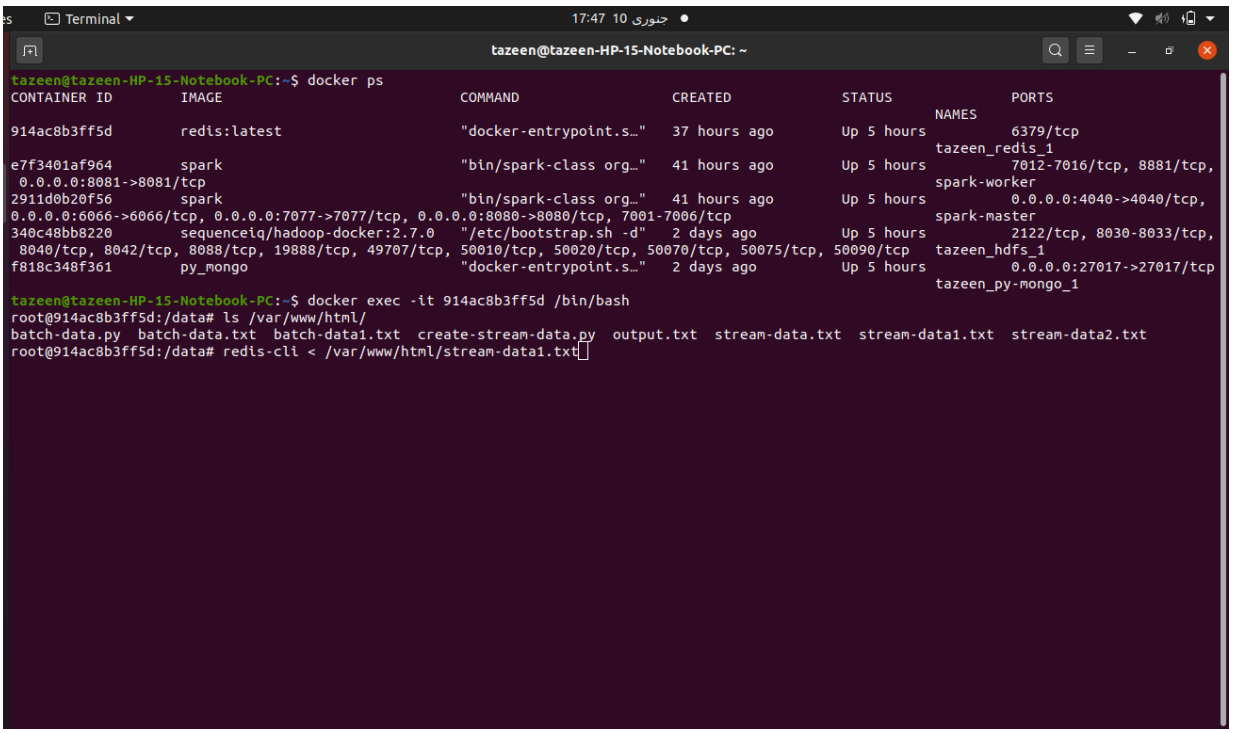
Now copying the script in ./services/spark/py-scripts/ folders which is mounted on /master/scripts folder in spark master.

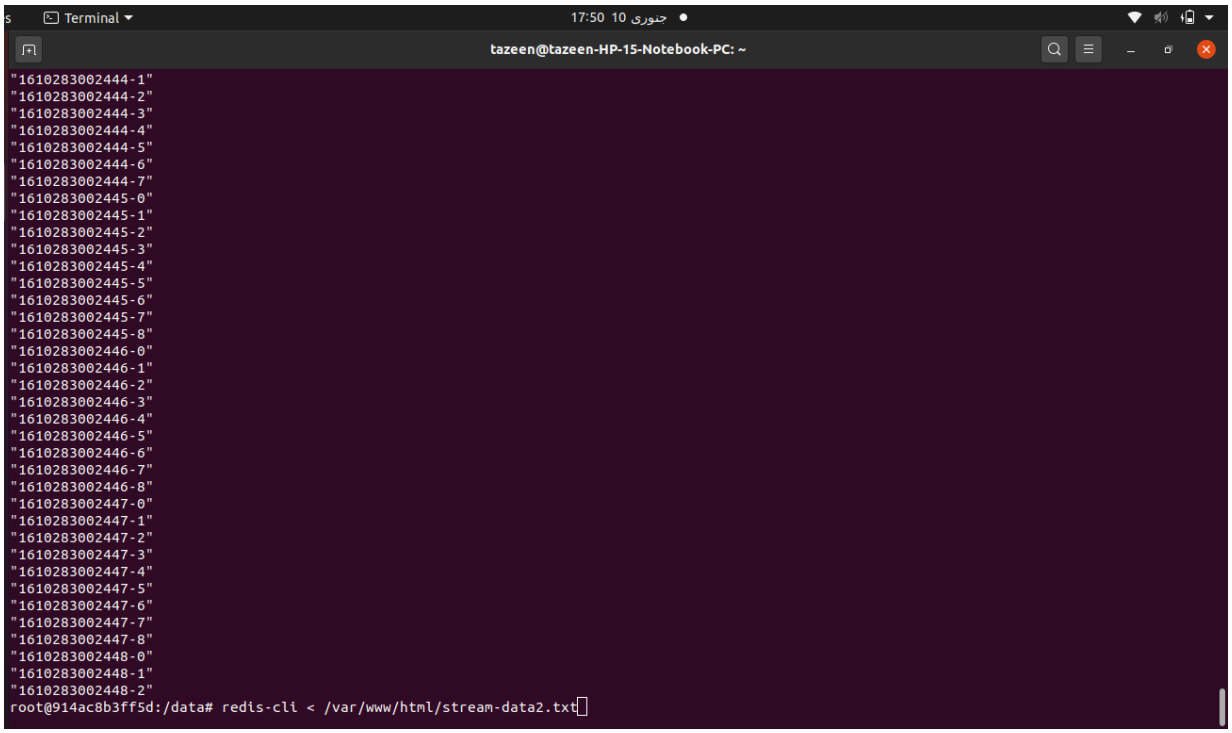


Connecting to the spark master terminal and executing the script for streaming.

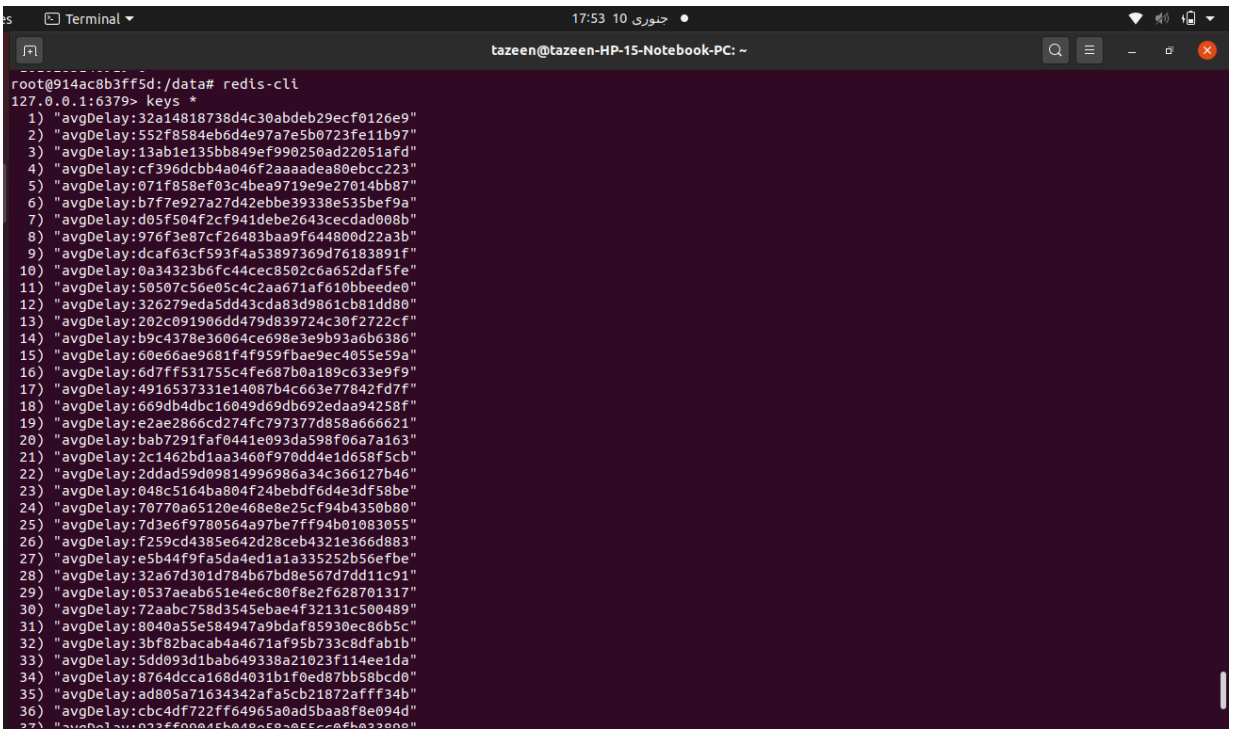


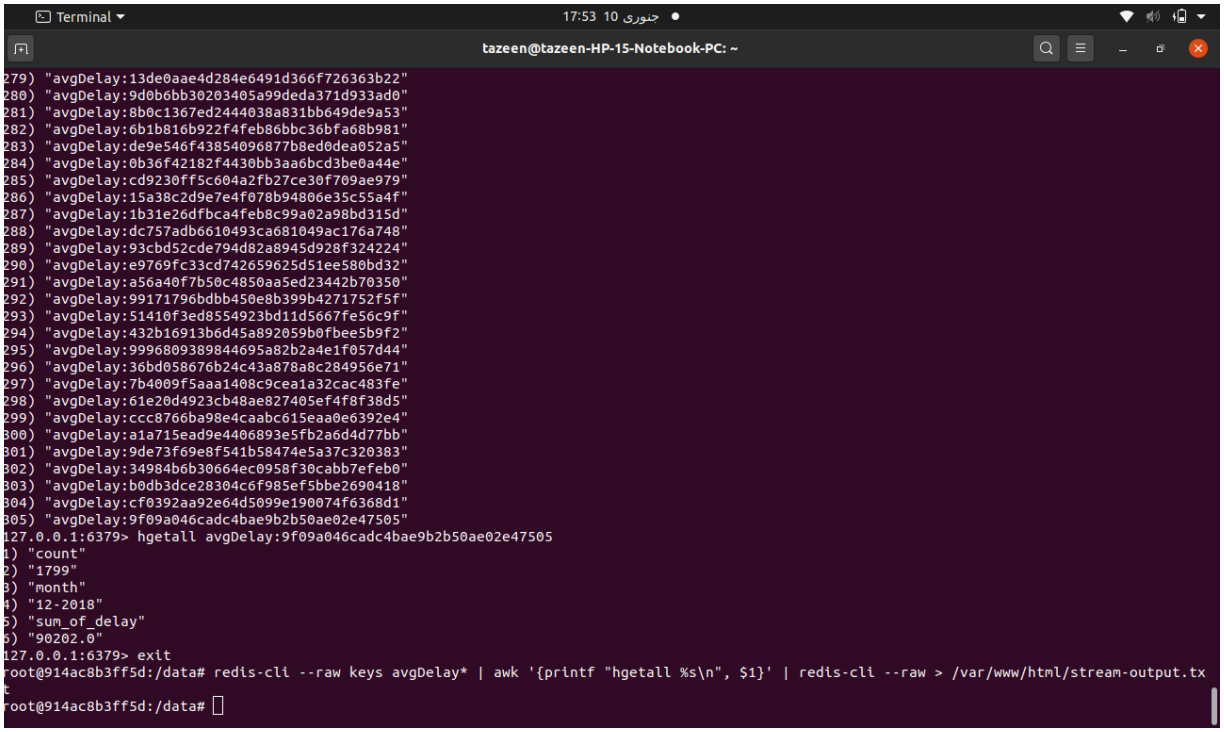
Now connecting to the redis docker container to send streams to spark. It consists of stream-data1.txt and stream-data2.txt as the redis streams for November 2018 and December 2018. The data is mounted as /var/www/html in redis docker.





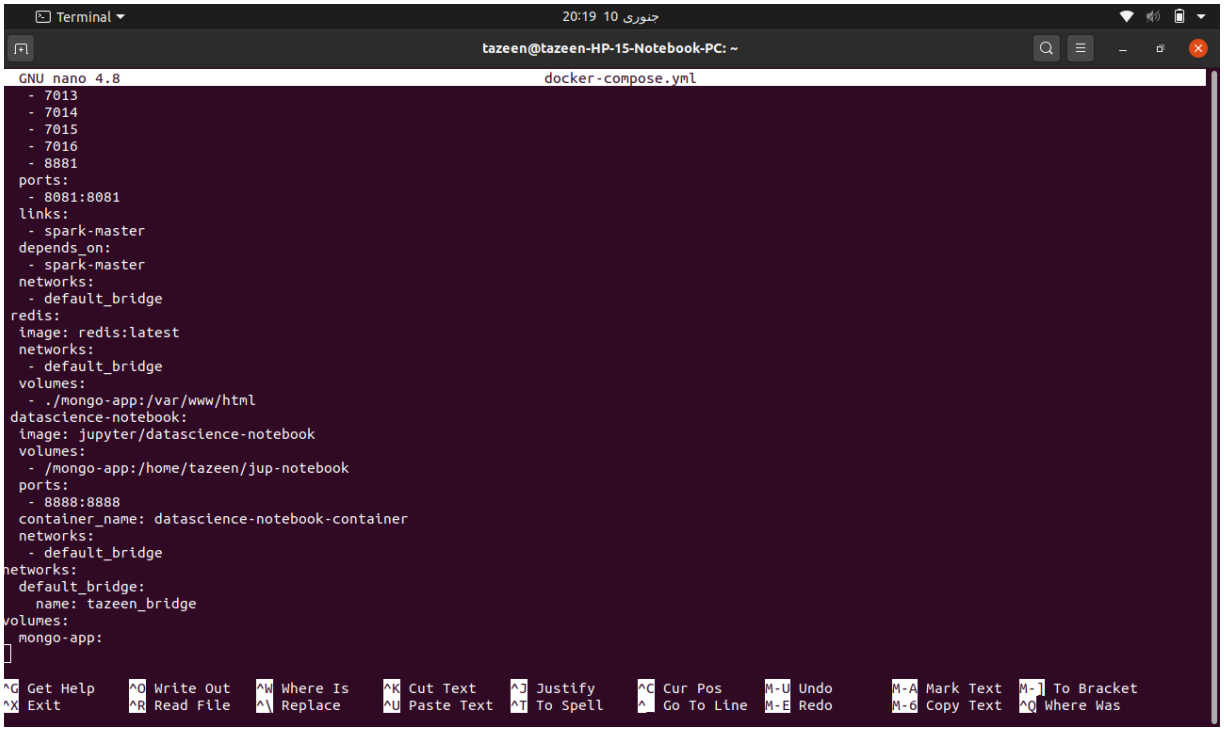
Using the cli to check the potput saved in redis table. Keys * lists all the keys created in redis. All keys starting with aygAverage are created by streaming. So saving all these results in stream-output.txt file. The output file will be saved in mongo-app folder in disk as /var/www/html is mounted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


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"

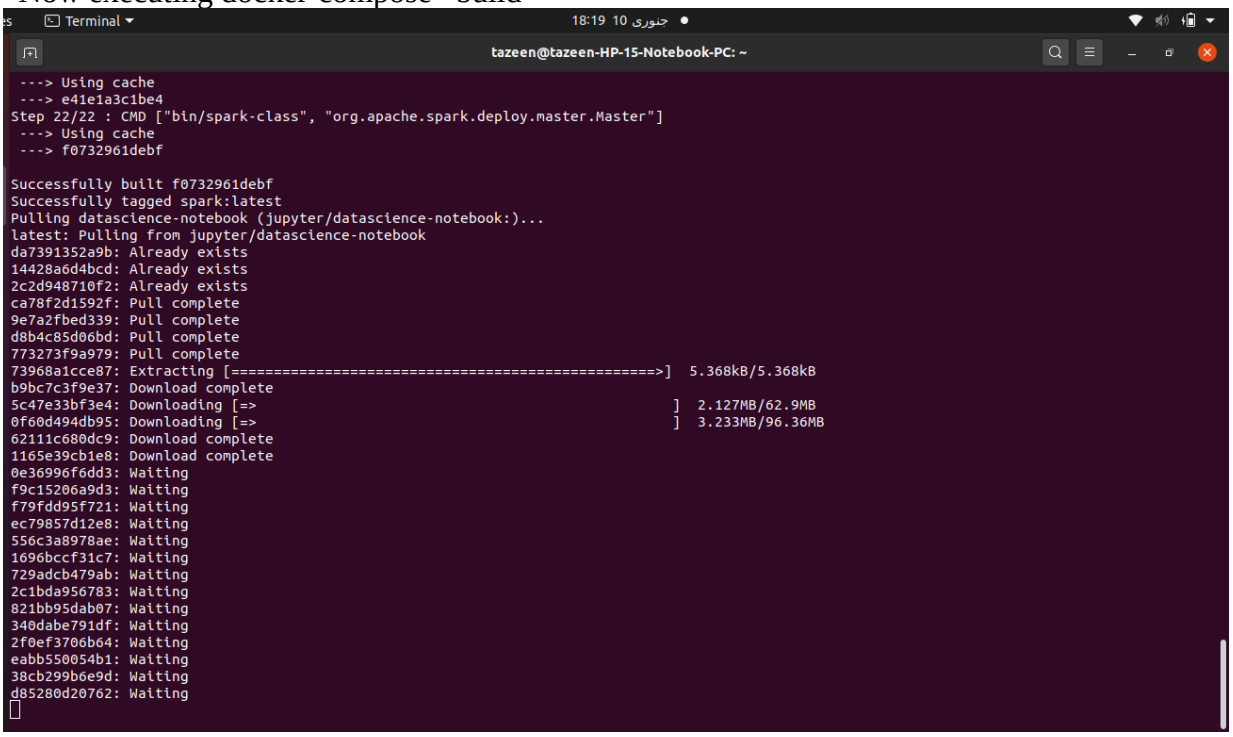

Terminal
17:53 10 جنوری
tazeen@tazeen-HP-15-Notebook-PC: ~
279) "avgDelay:13de0aae4d284e6491d366f726363b22"
280) "avgDelay:9d0b6bb30203405a99deda371d933ad0"
281) "avgDelay:8b0c1367ed2444038a831bb649de9a53"
282) "avgDelay:6b1b816b922f4feb86bbc36bfa68b981"
283) "avgDelay:de9e546f43854096877b8ed0dea052a5"
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:99968093898446 95a82b2a4e1f057d44"
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#

IV- Serving Layer

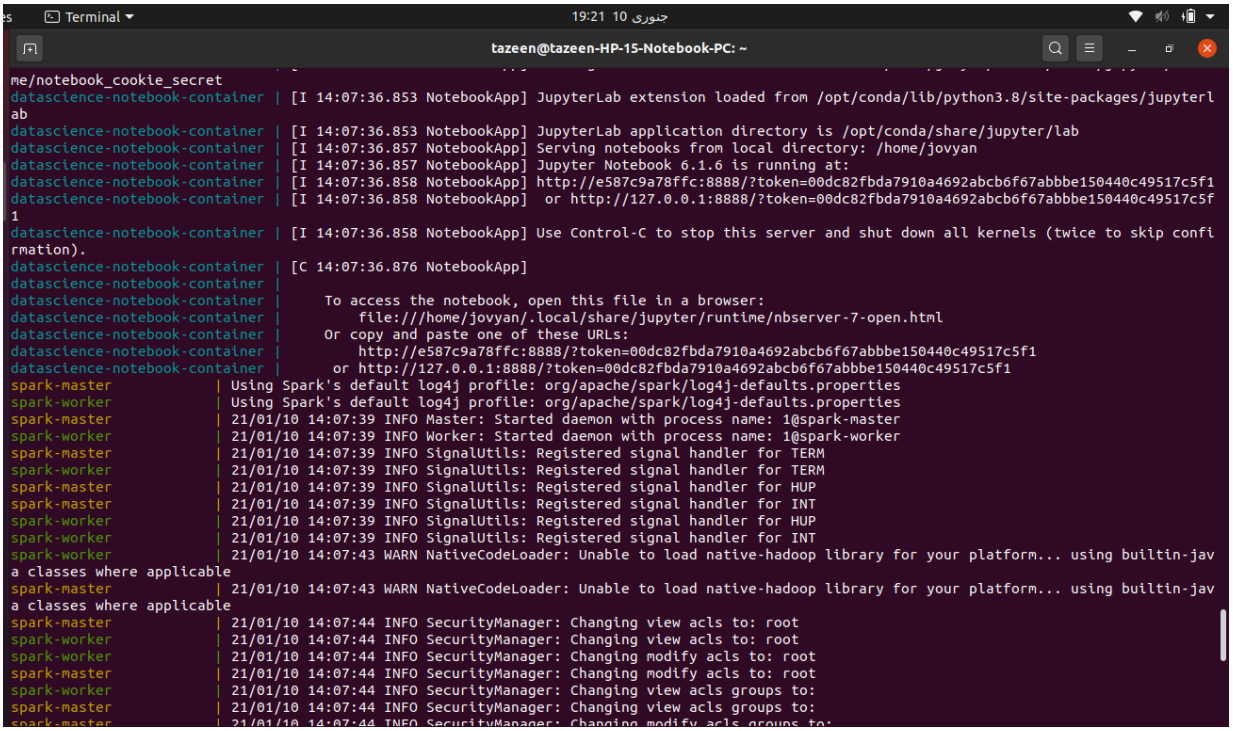
The last step is to create the serving layer which will show the results from batch and stream layers. To create a serving layer, jupyter notebook image is used in container. The updated docker-compose.yml is adds the jupyter image as below:



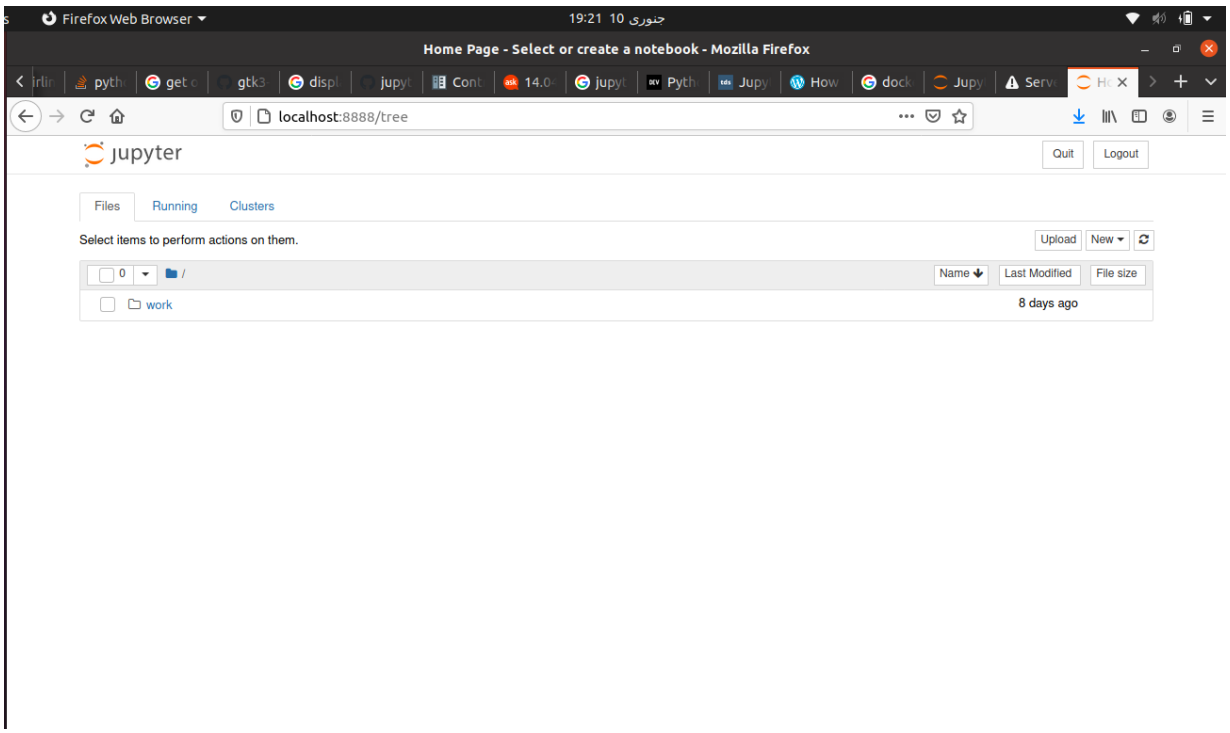
Now executing docker-compose –build



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

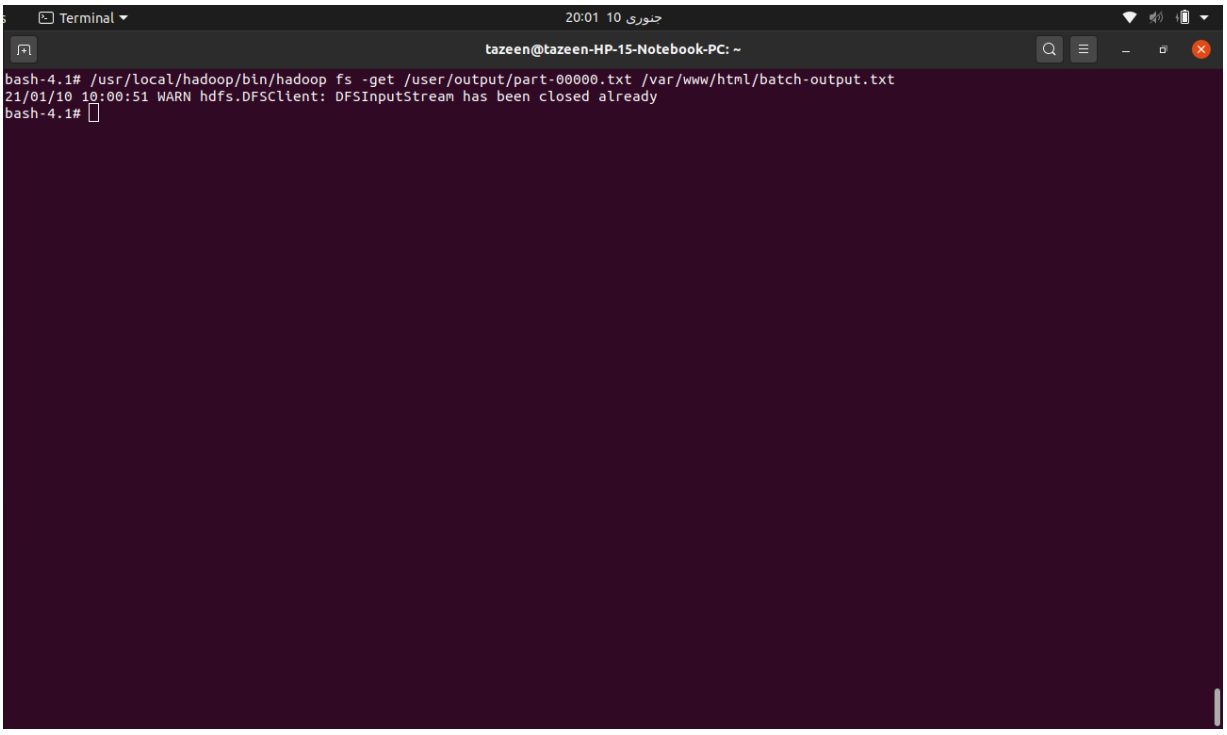


Navigating to the jupyter notebook on the url provided.

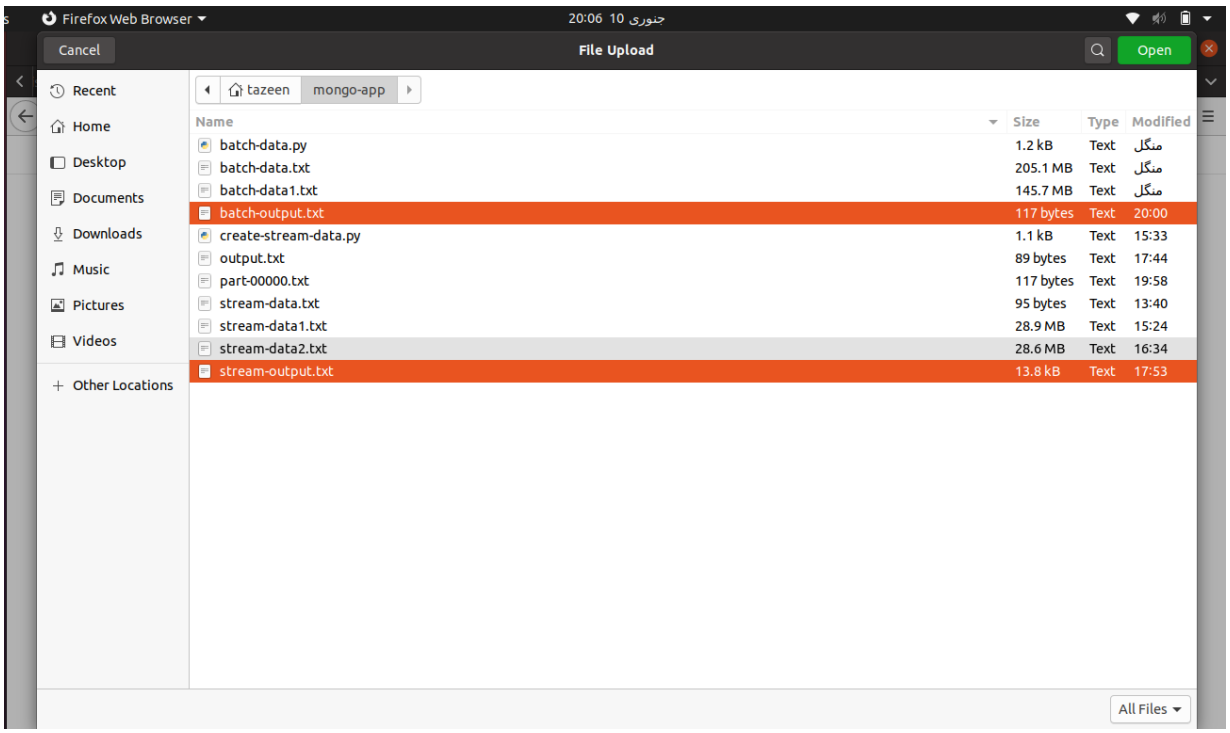


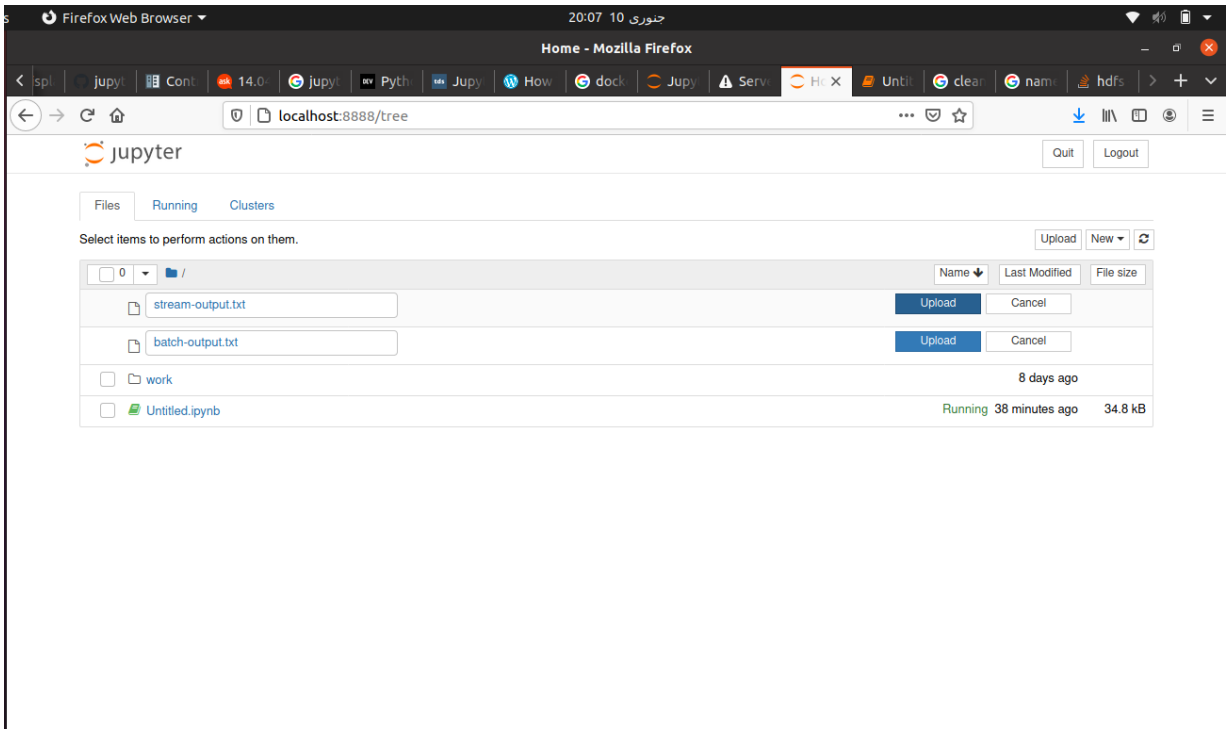
Now uploading output files in jupyter. These files are save in mongo-app folder which is mounted in the jupyter container.

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

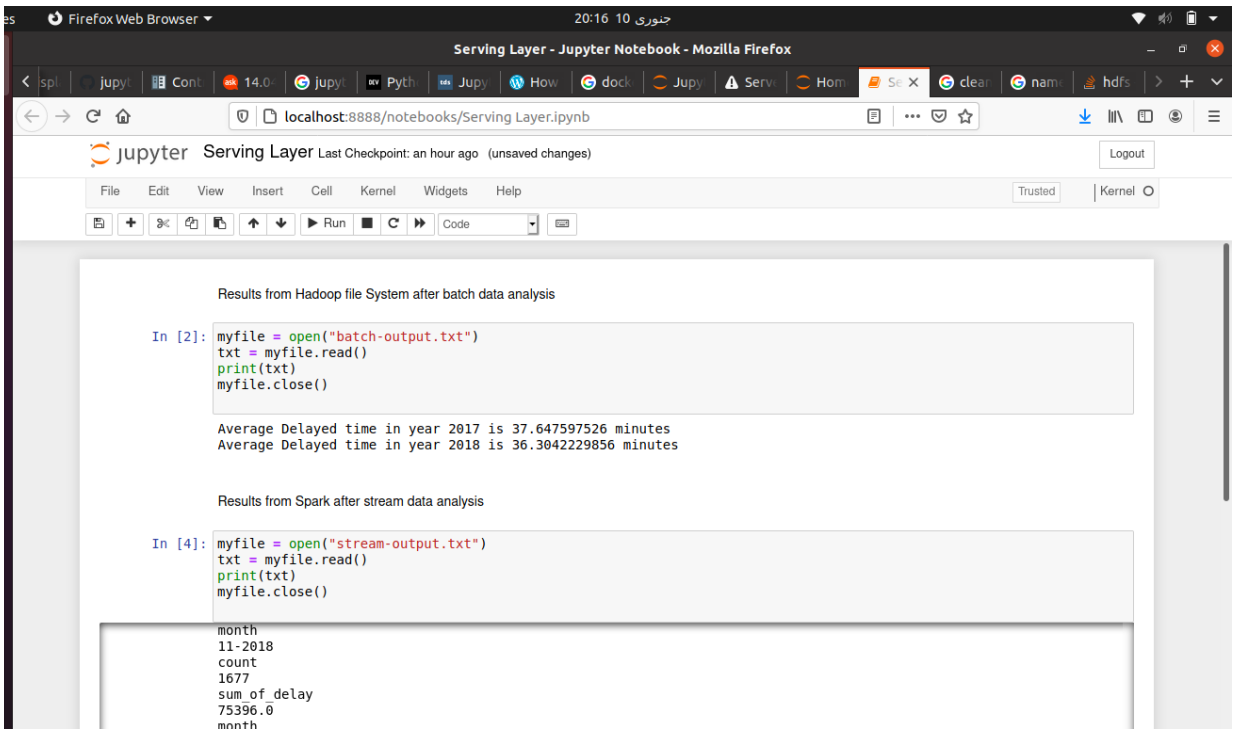


The stream-output file in already saved in mongo-app folder as previously shown. Now uploading these files in jupyter notebook.





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



5. Results

I was able to complete the lambda architecture and list few commands for redis lab. Multiple containers on the same network through bridge network. A custom bridge network-tazeen_bridge is created for communication. The results of lambda architecture are displayed in jupyter lab. It can be seen by the map/reduce job that average delayed flights time is around 36 minutes for year 2018 and around 37 minutes for year 2017. There is not a lot of delayed time in flights in both years. More such analysis can be done for more years. Streaming job has been quite fast and results are saved in redis quicker.

Problems faced:

It was difficult project because of different layers and communication. Each step was challenge and working alone increased the difficulty. Understanding communication with hadoop and mongo database, then creating a map/ reduce job, communication with redis and spark then showing results in jupyter lab. I had difficulty in understanding each step and had to search a lot. Spark SQL and queries are quite difficult and new to me. I had spent a lot of time to understand each process and execute it. There were also problem due system resources, a lot of time was required for huge data. Script execution would stuck and need to start again for large data. Overall it was a lot of learning but very difficult to implement. It took a lot of efforts and time to complete it.