### Data Challenge – Solution:

Details on how to run the application on Linux or OS X (Dockerizing the whole solution is preferred)

# Assumptions:

1. The user is able to run docker as non-root user without sudo.

This is required since the python program to get the IP address uses docker inspect command to find the dynamic ip.

How to ensure that non-root user can execute docker commands:

1. sudo groupadd docker
2. sudo gpasswd -a $USER docker
3. newgrp docker

Test by running docker ps -a command

2. You have a docker-compose setup on your host machine (Linux or OSX)

3. A git user to download and execute git clone commands

# Installation steps:

1. Get the application code:

Download the zip file from google drive :

Run :

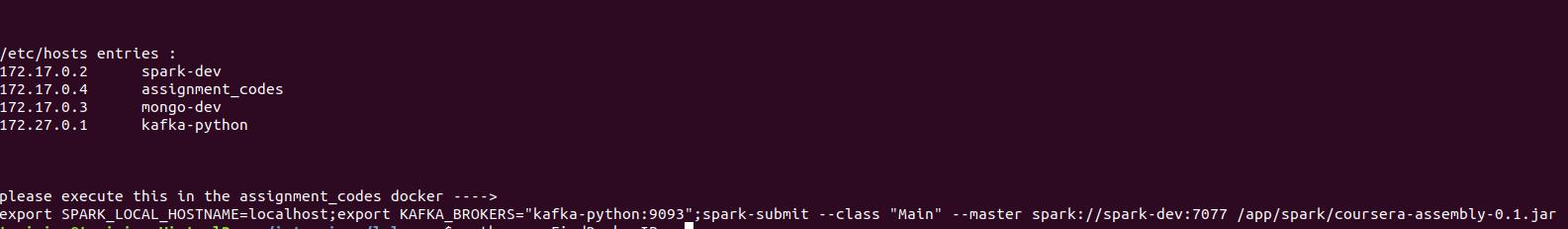
tar -xvf lalamove.zip

cd lalamove

docker build --tag my-submission:latest .

training@training-VirtualBox:~/interviews/lalamove$ python -x FindDockerIP.py

At the end of this step, you will see the different docker images and their IP.



2. Setup ‘/etc/hosts’ in the host machine using the command output of Step 1. This will allow us to produce to kafka from the host machine

3. Next, we can run the producer\_sample.py to send messages to newly created kafka using the below command:

Sample hosts file on my machine:

training@training-VirtualBox:~/interviews/lalamove$ cat /etc/hosts

127.0.0.1 localhost

127.0.1.1 training-VirtualBox

# The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

#kafka hosts

172.27.0.1 kafka-python

#mongo hosts

172.17.0.2 mongo-dev

4. Generate the data :

training@training-VirtualBox:~/interviews/lalamove$ python3 -x producer\_sample.py kafka-python:9093

*5. Run the Application on the assignment docker:*

*docker exec -it assignment\_codes bash*

*root@assignment\_codes:/app# export SPARK\_LOCAL\_HOSTNAME=localhost;export KAFKA\_BROKERS="kafka-python:9093";spark-submit --class "Main" --master spark://spark-dev:7077 /app/spark/coursera-assembly-0.1.jar*

*6. Verify the data being saved to mongo*

# Offset Management in Kafka for restarts and exactly-once semantics:

We are storing the offsets in mongodb after successfully processing the stream. This ensures that even if the spark-streaming application goes down, it will restart from the saved offset.

The below snippet handles this logic:

val fromOffsets = MongoDao.getTopicOffset().map(resultSet => {

*println*(s"topic : **$**{resultSet.topic} partitions: **$**{resultSet.partition} offset: **$**{resultSet.offset}")

new TopicPartition(resultSet.topic, resultSet.partition) -> resultSet.offset

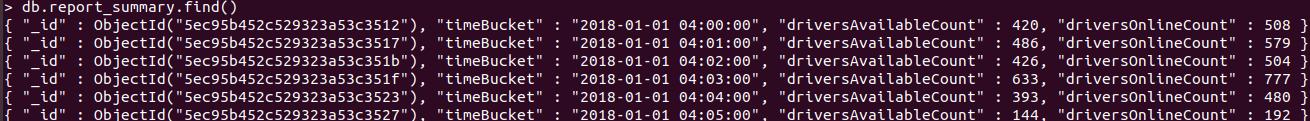
}

).toMap

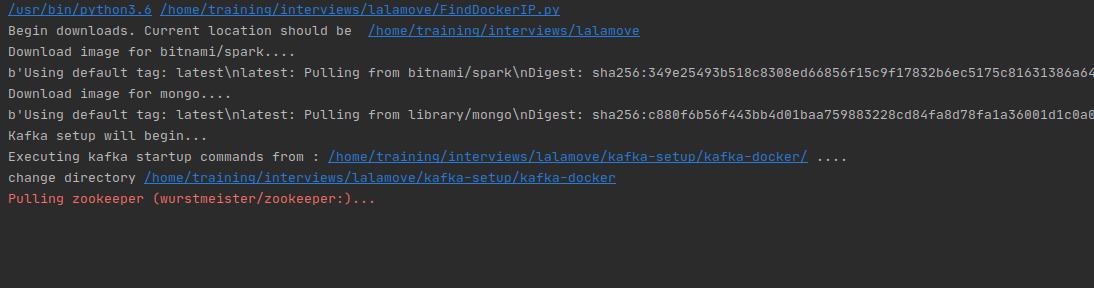
As shown below, the offset is preserved in a mongodb collection:



The streaming application saves the cumulative sum of online drivers and available for pickup drivers as a json in mongodb as well.



Automated deployment of the stack:



The final list of hostnames and their IP is displayed by the program.

We can use this to set our ‘/etc/hosts’ file if necessary.

Steps to follow for running the program:

Enter the docker container named assignment\_codes :

