Student Name: Tay Kai Xiang Matriculation Number: A0200236Y

Link to GitHub repository: https://github.com/kaixiangtay/CS3219_OTOT_D

Software required:
Docker: https://docs.docker.com/get-docker/
Docker Compose: https://docs.docker.com/compose/install/

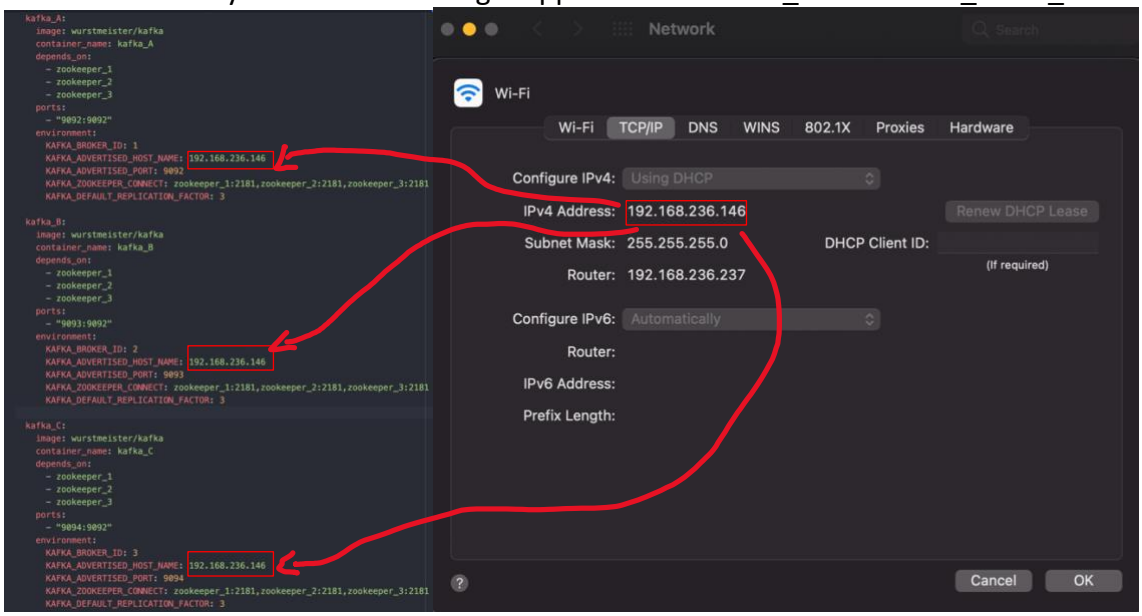
Docker Images to be used:
Kafka: https://hub.docker.com/r/wurstmeister/kafka/
Zookeeper: https://hub.docker.com/r/bitnami/zookeeper
Kafkacat: https://hub.docker.com/r/edenhill/kafkacat
Kafdrop: https://hub.docker.com/r/obsidiandynamics/kafdrop

Step 1: Ensure that Docker and Docker Compose has successfully installed.

Step 2: Clone the project folder on Github into local computer.

Step 3: Open up the Terminal and switch to the project folder directory.

Step 4: Inside the docker-compose.yml file, please change the KAFKA_ADVERTISED_HOST_NAME to your network IP Address accordingly. Below are screenshots of my IPv4 address being mapped to the KAFKA_ADVERTISED_HOST_NAME.



Step 5: Then, execute the command **docker-compose up -d** where the Docker containers will be build and running in the background. For this task, I have created 3 zookeepers and 3 Kafka brokers after reading from http://hbase.apache.org/book.html#zookeeper

> *How many ZooKeepers should I run?*
>
> You can run a ZooKeeper ensemble that comprises 1 node only but in production it is recommended that you run a ZooKeeper ensemble of 3, 5 or 7 machines; the more members an ensemble has, the more tolerant the ensemble is of host failures. Also, run an odd number of machines. In ZooKeeper, an even number of peers is supported, but it is normally not used because an even sized ensemble requires, proportionally, more peers to form a quorum than an odd sized ensemble requires. For example, an ensemble with 4 peers requires 3 to form a quorum, while an ensemble with 5 also requires 3 to form a quorum. Thus, an ensemble of 5 allows 2 peers to fail, and thus is more fault tolerant than the ensemble of 4, which allows only 1 down peer.

Expected output:

```
(base) tkx@Tays-MacBook-Pro OTOT_D % docker-compose up -d
Creating network "otot_d_default" with the default driver
Creating zookeeper_2 ... done
Creating zookeeper_3 ... done
Creating zookeeper_1 ... done
Creating kafka_B     ... done
Creating kafka_C     ... done
Creating kafka_A     ... done
Creating kafka-web-ui ... done
```

Step 6: Next, we can execute the command **docker ps** where the list of containers being running will be displayed in the Terminal.

```
(base) tkx@Tays-MacBook-Pro OTOT_D % docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED         STATUS         PORTS                                                                                  NAMES
0c849b9b4289   obsidiandynamics/kafdrop     "/kafdrop.sh"            2 minutes ago   Up 2 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp                                              kafka-web-ui
398d5e7a3834   wurstmeister/kafka           "start-kafka.sh"         2 minutes ago   Up 2 minutes   0.0.0.0:9094->9092/tcp, :::9094->9092/tcp                                              kafka_C
a553c5429846   wurstmeister/kafka           "start-kafka.sh"         2 minutes ago   Up 2 minutes   0.0.0.0:9092->9092/tcp, :::9092->9092/tcp                                              kafka_A
ba3ea8a342d2   wurstmeister/kafka           "start-kafka.sh"         2 minutes ago   Up 2 minutes   0.0.0.0:9093->9092/tcp, :::9093->9092/tcp                                              kafka_B
ea03363d7ca7   bitnami/zookeeper:latest     "/opt/bitnami/script…"   2 minutes ago   Up 2 minutes   8080/tcp, 0.0.0.0:57927->2181/tcp, 0.0.0.0:57928->2888/tcp, 0.0.0.0:57929->3888/tcp   zookeeper_1
d038b404ee78   bitnami/zookeeper:latest     "/opt/bitnami/script…"   2 minutes ago   Up 2 minutes   8080/tcp, 0.0.0.0:57930->2181/tcp, 0.0.0.0:57931->2888/tcp, 0.0.0.0:57932->3888/tcp   zookeeper_2
75694a40a751   bitnami/zookeeper:latest     "/opt/bitnami/script…"   2 minutes ago   Up 2 minutes   8080/tcp, 0.0.0.0:57924->2181/tcp, 0.0.0.0:57925->2888/tcp, 0.0.0.0:57926->3888/tcp   zookeeper_3
```

Step 7: As seen in step 6, kafka-web-ui container is running at local port 8080 where I configured in .yml.

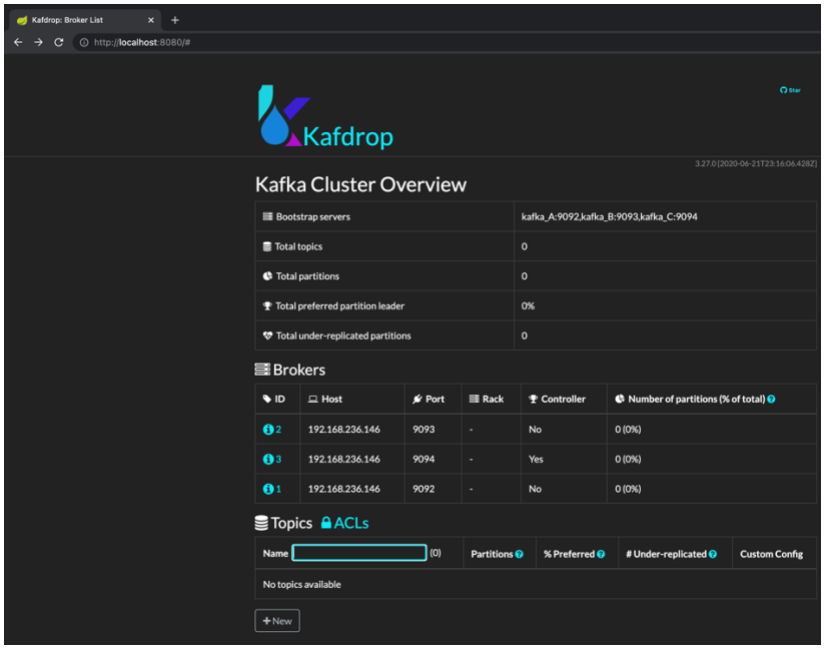
```yaml
kafka-ui:
  image: obsidiandynamics/kafdrop
  container_name: kafka-web-ui
  environment:
    - KAFKA_BROKERCONNECT=kafka_A:9092,kafka_B:9093,kafka_C:9094
    - SERVER_PORT=8080
  ports:
    - "8080:8080"
```

We can open up http://localhost:8080 where the Kafdrop, a web UI can be used for viewing Kafka topics and browsing consumer groups will be running. It also displays information such as brokers, topics, partitions, consumers and lets you view the messages.

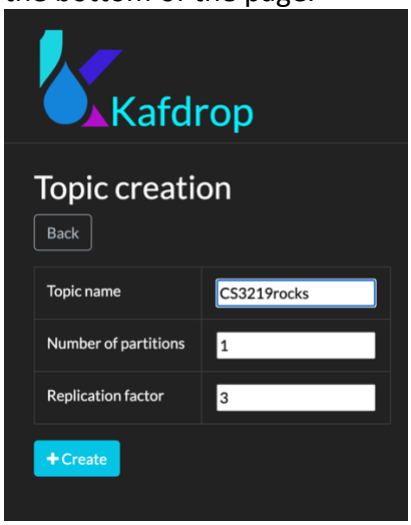
Reference: https://dev.to/ekoutanov/kafdrop-an-open-source-kafka-web-ui-mbn

Currently, under the Brokers section in the main page, we can see that kafka broker with ID 3 running on port 9094 which is kafka_C is the controller/leader.

One important thing to note: Assignment of leader/controller is random and determine by the zookeepers.



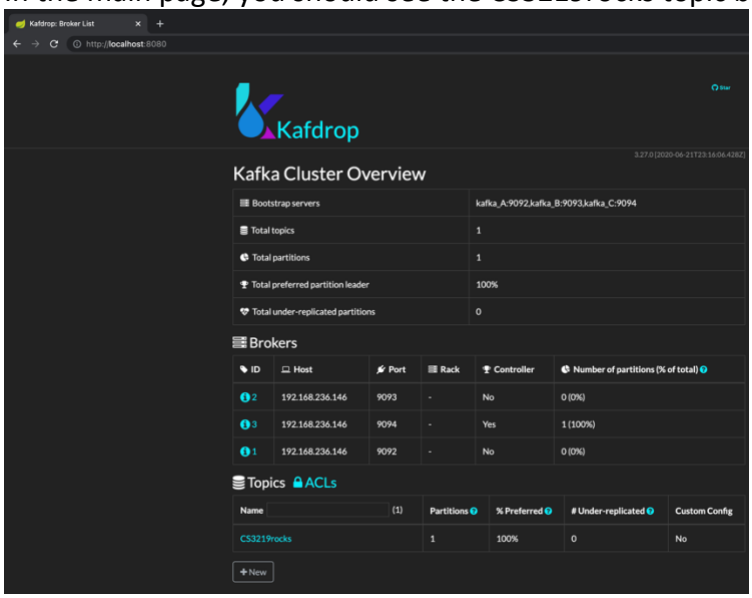
Step 8: We will be using the Kafdrop to create a Kafka topic. We can click on +New button at the bottom of the page.



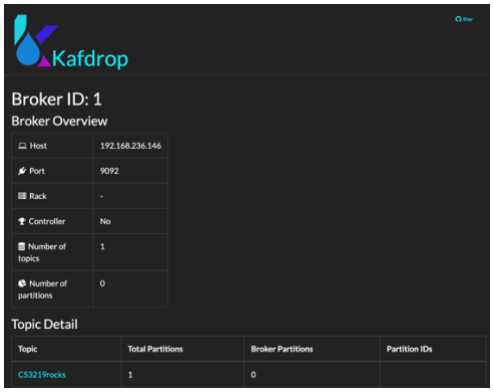
Here is an example of how I create a Kafka topic. Notice, I decided to put replication factor 3 which means that for the topic CS3219rocks, three copies will be created and distributed evenly to the Kafka brokers which are kafka_A, kafka_B, kafka_C.

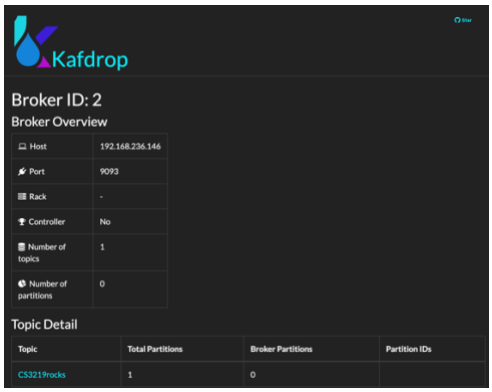
Once successful, you will be able to see this message below and you can click the Back button to go back to the main page.

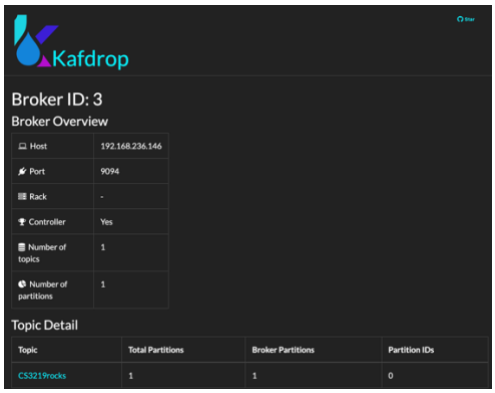
In the main page, you should see the CS3219rocks topic being displayed.



You can click on the ID of the Kafka Broker and verify all of them have subscribed to the topic CS3219rocks.







Step 9: Next, we can execute **docker pull edenhill/kafkacat:1.6.0** to retrieve the kafkacat Docker image which will be used for troubleshooting Kafka deployments and can be used during the process where Kafka message is being produced and consumed.

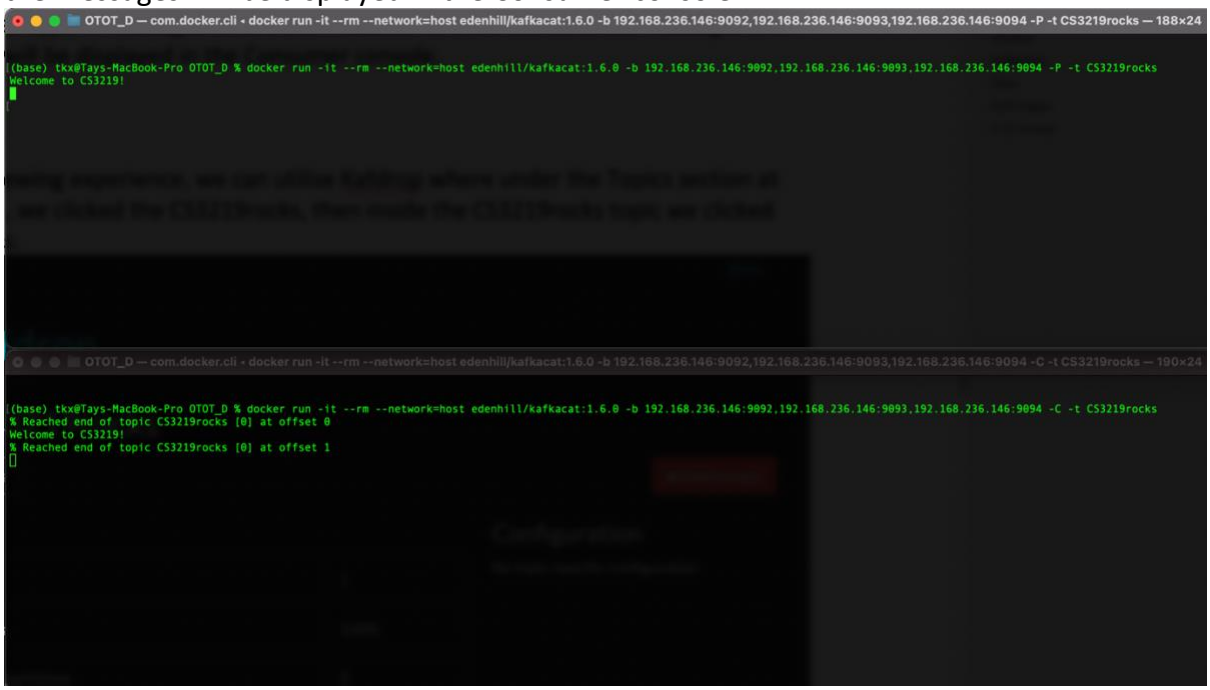
Step 10: Next, open another console and navigate to the same project folder directory, CS3219_OTOT_D.

In both windows, choose one of the 2 following commands and execute once in each window.

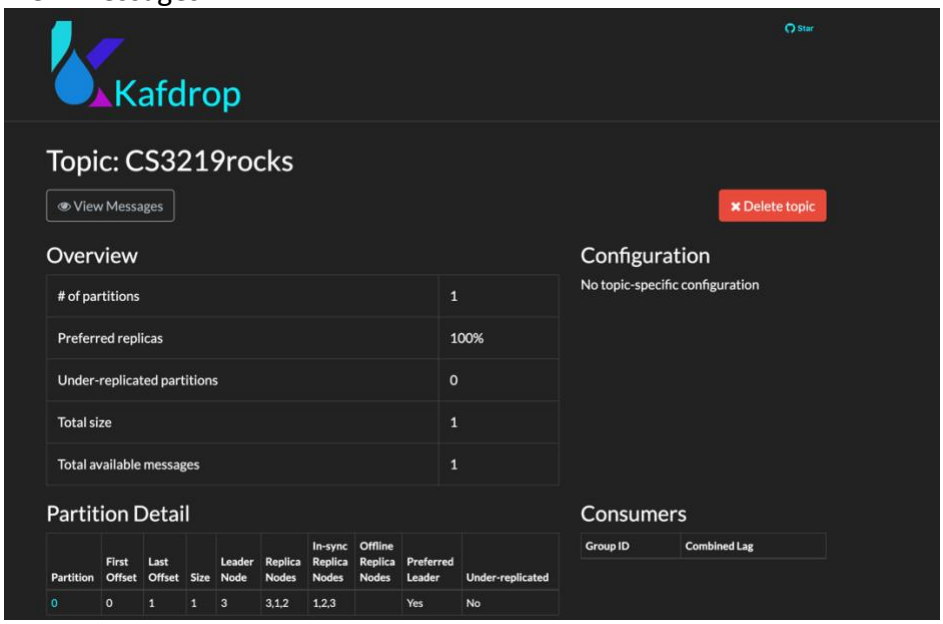
docker run -it --rm --network=host edenhill/kafkacat:1.6.0 -b 192.168.236.146:9092,192.168.236.146:9093,192.168.236.146:9094 -P -t CS3219rocks

docker run -it --rm --network=host edenhill/kafkacat:1.6.0 -b 192.168.236.146:9092,192.168.236.146:9093,192.168.236.146:9094 -C -t CS3219rocks

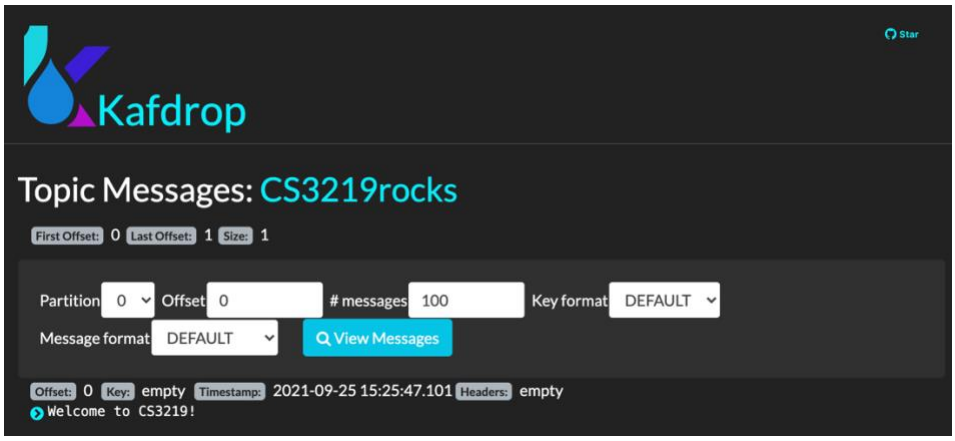
The first command is setting the console window to be Producer mode where Kafka messages will be produced to the Kafka topic CS3219rocks while the second command is setting the console window to be Consumer mode where the Kafka messages produced to the Kafka topic CS3219rocks will be subscribed.
Step 11: We can type in messages where the Producer console will send out messages and the messages will be displayed in the Consumer console.



For a better viewing experience, we can utilise Kafdrop where under the Topics section at the main page, we clicked the CS3219rocks, then inside the CS3219rocks topic we clicked View messages.



It will bring you to this Topic Messages. If the page is not loaded yet, we can click the blue button View Messages which will display the message as well as the Timestamp which is the entry time of the Kafka message.

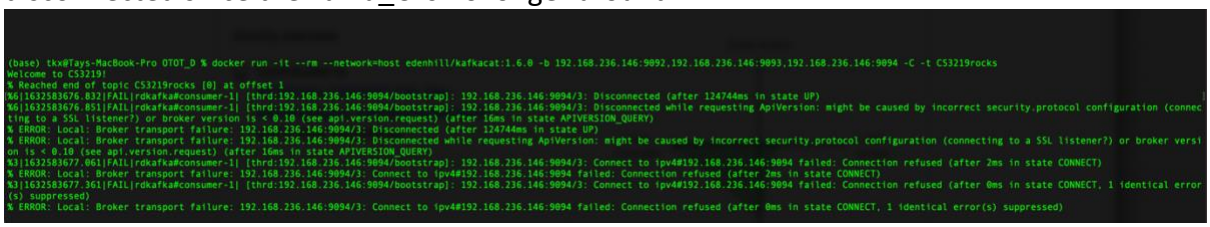


Note: The Timestamp is based on GMT timing so for Singapore timing we have to add 8 hours.

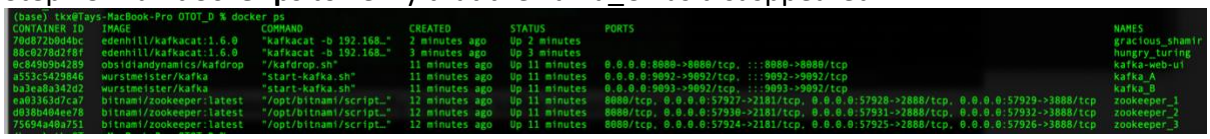
Step 12: Now, to test the management of master node failure, we will kill off kafka_C, the controller which is the leader / master node.

Open up the third separate console and navigate to the CS3219_OTOT_D directory. We will force remove the running container kafka_C by executing the command **docker rm -f kafka_C**.

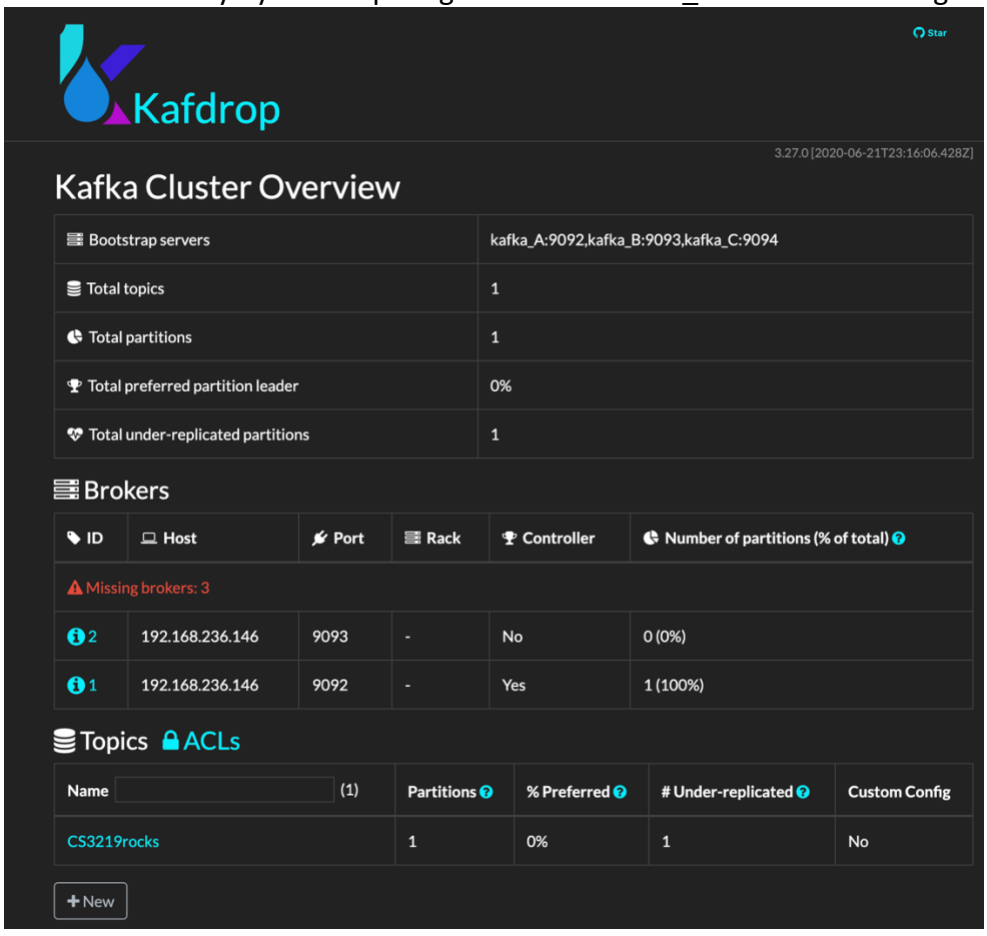
Immediately, the Consumer output will prompt errors where 192.168.236.146:9094 is disconnected since the kafka_C is no longer around.



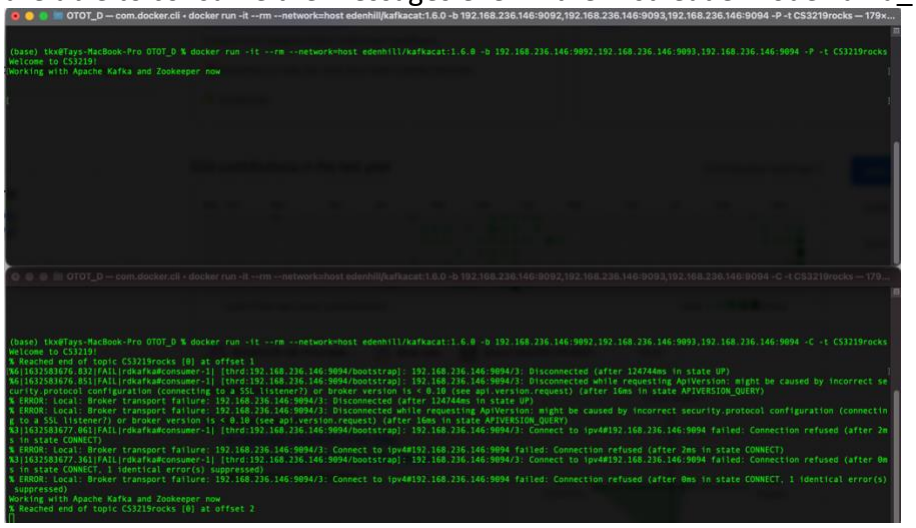
Step 13: Run **docker ps** to verify that the kafka_C has disasppeared.



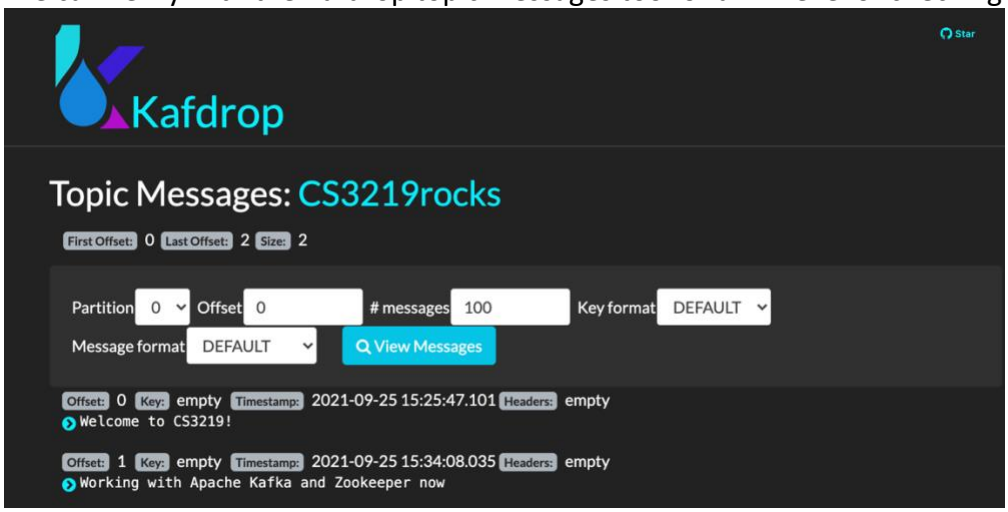
Alternatively, refresh the Kafdrop page and you should see that kafka_C, the Kafka broker with ID 3 running on port 9094 has disappeared and a new leader / controller has been elected randomly by Zookeeper again which is kafka_A with ID 1 running on port 9092.



Step 14: Try to send a new message with the same Producer console and you will be able to receive the same message in the Consumer console. This shows that the remaining nodes are able to consume the messages even if the first leader node kafka_C has went down.



We can verify with the Kafdrop topic messages too for a 2nd level of checking.



Step 15: Run **docker-compose down** to stop the containers from running when you are done with testing and trying out this project.

Some useful tips (Applicable if your computer only have Docker Instances generated from this project):

1) To perform a clean restart of running the Docker Instances for the project, we can

(a) Execute **docker-compose down** to stop the Docker container(s)
(b) Run **docker rm -f $(docker ps -a -q)** to delete all container(s)
(c) Run **docker volume rm $(docker volume ls -q)** to delete all volumes
(d) Perform **docker-compose up -d** to start and run the Docker container(s) again

2) Suppose you want to wipe out the Docker Instances from the computer,
You can perform steps (a) to (c) from above.