1a. Ambrose Liew Cheng Yuan, A0204750N

1b. https://github.com/MorningLit/CS3219

1c. and 1d.

To set up the Kafka cluster, download my GitHub Repository and change directory into the OTOT_Task_D folder. Once in, run 'docker-compose up -d' to run the containers in the background.

```
ambrc@LAPTOP-JULD6R3U MINGW64 ~/Downloads/nus/~y3s1/cs3219/assignments/otot/OTOT_Task_D (master)
$ docker-compose up -d
Creating network "otot_task_d_default" with the default driver
Creating otot_task_d_zookeeper-server_1 ... done
Creating otot_task_d_kafka-server2_1    ... done
Creating otot_task_d_kafka-server1_1    ... done
Creating otot_task_d_kafka-server3_1    ... done
```

Once done, the following containers can be seen

Next, run 'docker exec -it otot_task_d_kafka-server1_1 bash' to run a bash shell in the Kafka server container.

```
ambrc@LAPTOP-JULD6R3U MINGW64 ~/Downloads/nus/~y3s1/cs3219/assignments/otot/OTOT_Task_D (master)
$ docker exec -it otot_task_d_kafka-server1_1 bash
[appuser@70cc9a14fe22 ~]$
```

Next run
'kafka-topics --topic mytopic --create --zookeeper zookeeper-server:2181 --replication-factor 2 --partitions 3' to create our Kafka topic named mytopic, which allows us organise messages and to produce to and consume from.

```
[appuser@70cc9a14fe22 ~]$ kafka-topics --topic mytopic --create --zookeeper zookeeper-server:2181 --replication-factor 2 --partitions 3
Created topic mytopic.
```

Next run 'kafka-console-producer --topic mytopic --broker-list localhost:9092' to start the console producer client.

Now, start up a new terminal and run ‘docker exec -it otot_task_d_kafka-server1_1 bash’ to start another bash shell in the Kafka server container.

Run ‘kafka-console-consumer --topic mytopic --bootstrap-server localhost:9092 --from-beginning’ to run the console consumer client.

Now we can type any message from the console producer client to send and the console consumer client can listen and receive the messages. This demonstrates the successful implementation of a Pub-Sub messaging system.

```
[appuser@189c293210dd ~]$ kafka-console-producer --broker-list localhost:9092 --topic mytopic
>hello
>world!
```

```
[appuser@189c293210dd ~]$ kafka-console-consumer --bootstrap-server localhost:9092 --topic mytopic --from-beginning
hello
world!
```

Next onto the killing of the master node, we first need to run ‘kafka-topics --topic mytopic --zookeeper zookeeper-server:2181 --describe’ on a Kafka server to see the details of each topic and their leader.

```
Topic: mytopic  TopicId: CgxRax0LRvivuCxNLxwrzg PartitionCount: 3       ReplicationFactor: 2    Configs:
        Topic: mytopic  Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3,1
        Topic: mytopic  Partition: 1    Leader: 1       Replicas: 1,2   Isr: 1,2
        Topic: mytopic  Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

For this example, I will kill node 1 to see the demonstration of a node taking over as a master node.

Run ‘docker container kill otot_task_d_kafka-server1_1’ on a terminal to kill the node.

Then once successful, run ‘kafka-topics --topic mytopic --zookeeper zookeeper-server:2181 --describe’ on one of the Kafka servers to see the management happen.

```
Topic: mytopic  TopicId: CgxRax0LRvivuCxNLxwrzg PartitionCount: 3       ReplicationFactor: 2    Configs:
        Topic: mytopic  Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3
        Topic: mytopic  Partition: 1    Leader: 2       Replicas: 1,2   Isr: 2
        Topic: mytopic  Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

We can see the leader of partition 1 of mytopic has changed from the number from 1 to 2. Which means a node has successfully took over as the master node.