Project Milestone 2 - Data Ingestion Software-- Kafka Clusters

Esam Uddin - 100711116

2/14/2022

Given Lab 2 Repository: https://github.com/goergedaoud/SOFE4630U-tut2.git

Group Project Repository:

https://github.com/esam191/Intelligent-Transportation-System

Learning Objectives:

- Understand Event-Driven Architecture (EDA).
- Get familiar with Kafka concepts and APIs.
- Using Kafka within Google Cloud Platform (GCP).

Procedure:

1. **Watch the first 27 minutes from the following video that describes the concepts of EDA**

   https://youtu.be/OqN6x_vNsds

2. **Watch the following video that describes the main components of Kafka as a core of EDA**

   https://youtu.be/JalUUBKdcA0

3. **Answer the following questions:**

4. **What is EDA? What are its advantages and disadvantages?**

- EDA stands for Event-driven architectures and they are ideal for distributed application architectures.
- Event-driven architectures consist of event producers and event consumers

- The producers work as publishers and they don't know about the consumers that work as subscribers and they don't know about the producers. They both communicate through a broker.
- The advantages of event-driven architectures are that since the events are captured as they happen in the system, the event producers and consumers are able to share response info in real time.
- The disadvantages of event-driven architectures are that the events are loosely coupled and multiple duplicate messages due to a single event.

5. **In Kafka, what's meant by cluster, broker, topic, replica, partition, zookeeper, controller, leader, consumer, producer, and consumer group?**

   Cluster - consists of one or more brokers that run kafka
   Broker - allows producer processes to send messages into a topic and consumer processors to access that info
   Topic - the means of which producers send messages and consumers access that message within a particular topic; topics are divided into partitions
   Replica - multiple copies of data across brokers
   Partition - consists of records; helps topics split data into a specific topic across brokers
   Zookeeper - manages the brokers in the cluster
   Controller - handles the state for all brokers in the cluster
   Leader - manages the read/write requests for partition
   Consumer - send message records to a topic in the broker
   Producer - accesses records from a certain topic
   Consumer Group - contains partitions of a topic assigned by kafka

6. **Follow the following video to install Kafka into your local machine and create topics, consumers, and producers using Kafka's built-in tools.**

   https://youtu.be/Kx0o2jeMB0o

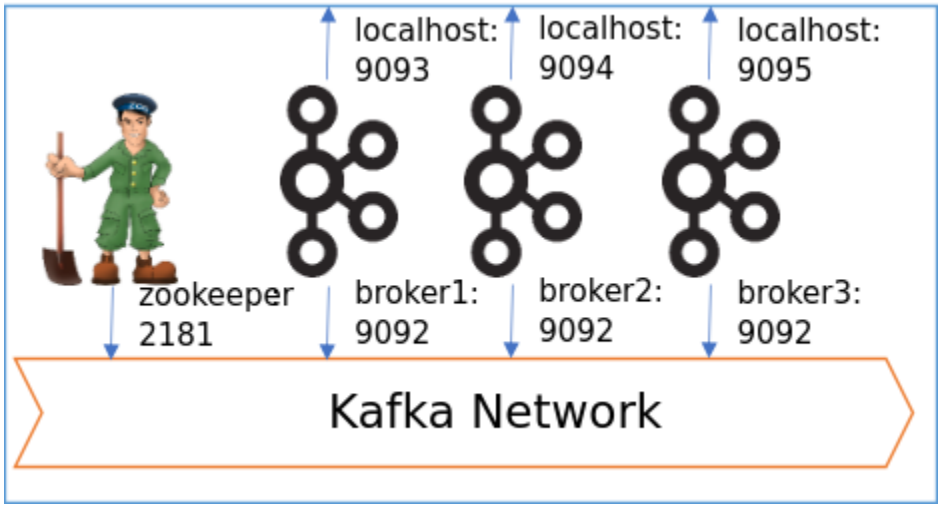
   Creating Topics:

```
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker1 kafka-topics --create --topic topic  --partitions 3
--replication-factor 3 --if-not-exists --bootstrap-server broker1:9092
Created topic topic.
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker1 kafka-topics --create --topic topic2 --partitions 3
--replication-factor 2 --if-not-exists --bootstrap-server broker1:9092,broker2:9092,broker3:9092
Created topic topic2.
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker1 kafka-topics --describe --bootstrap-server broker1:9
092
Topic: topic    TopicId: bc0NoTTmRvu3TK-YFGbxQA PartitionCount: 3       ReplicationFactor: 3    Configs:
        Topic: topic    Partition: 0    Leader: 3       Replicas: 3,2,1 Isr: 3,2,1
        Topic: topic    Partition: 1    Leader: 1       Replicas: 1,3,2 Isr: 1,3,2
        Topic: topic    Partition: 2    Leader: 2       Replicas: 2,1,3 Isr: 2,1,3
Topic: topic2   TopicId: cquykvDrR9OmJ0tg7PJmPQ PartitionCount: 3       ReplicationFactor: 2    Configs:
        Topic: topic2   Partition: 0    Leader: 2       Replicas: 2,3   Isr: 2,3
        Topic: topic2   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1
        Topic: topic2   Partition: 2    Leader: 1       Replicas: 1,2   Isr: 1,2
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker1 kafka-topics --list --bootstrap-server broker1:9092
topic
topic2
```

Creating Producers:

Creating Consumers:

```
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker2 kafka-console-consumer
--bootstrap-server broker1:9092 --topic topic --from-beginning
temp=20
value1
0.0
2.5
5.0
7.5
10.0
value2
^C
 --request-required-acks 1 --broker-list broker2:9092 --topic topic"
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker1 bash -c "seq 0 2.5 10 | kafka-console-producer
esam@esam-Precision-3520:~/Courses/Cloud Computing/SOFE4630U-tut2/v1$ sudo docker exec broker2 kafka-console-consumer
--bootstrap-server broker1:9092 --topic topic --from-beginning --max-messages 8                                          u
temp=20
value1
0.0
2.5
5.0
7.5
10.0
value2
Processed a total of 8 messages
```



7. **Follow the following video to generate NodeJS scripts for creating topics, consumers, and producers**

   https://youtu.be/R873BlNVUB4?t=2424

Note:

- **Use the docker application from the repository not shown in the video.**
- **To use the docker application from the repository, Kafka brokers' addresses will be localhost:9093,9094, and 9095.**
- **Follow only the NodeJS scripts, not the docker commands.**
- **When the partition is not specified, Kafka will decide the partition randomly or by hashing the key.**

8. **Using the python library Kafka-python, write three python scripts that**

   - **Create a topic**
   - **Produce messages on a topic. The message's key and the partition should be optional. Also, an error message should be displayed in the case of failure.**
   - **Repeatedly consume messages from a topic and display the consumed messages whenever available. The group id should be optional.**

9. **Prepare a video showing the codes that generated topics, produce messages, and consume them in both NodeJS and python. Your video should display the possible producer and consumer scenarios.**

   https://drive.google.com/drive/folders/1DV9t4g8MXCTUSoDLuQU6PMb35znbpJ0r?usp=sharing

10. **A problem in the used YAML file to create the docker images is that the data inside Kafka clusters are not persistent which means if the docker images are down, all its messages are lost. Update the YAML file for persistent data (hint: it's related to the volume options in Kafka brokers and zookeeper). Describe how this update solves the problem.**

    The following is the update that is added to the YAML file for persistent data :

    volumes:
        - /home/srv/kafka:/kafka/kafka-logs-1

    After adding the update above, the first time we run docker-compose up, the volume is generated and the same volume is reused the next time we run it. Kafka and Zookeeper need externally mounted volumes to persist data. This

update allows for persistent data so the data inside Kafka clusters is persistent. This means that even if the docker images/containers are down or restarted, all its messages still exist and they aren't lost.

11. **Follow the following video about Kafka in Confluent Cloud**

    https://youtu.be/vllYRz65tgA

    and use the shown CLI to create a topic, consumer, and producer. Also update your python code to create a consumer, and producer using Kafka in Confluent Cloud (hint: only the connection information of Kafka Cluster has to be updated). Record a video illustrating those tools and showing them in action.

https://drive.google.com/drive/folders/1DV9t4g8MXCTUSoDLuQU6PMb35znbpJ0r?usp=sharing

12. **Upload the used files, reports, and videos (links) into your repository.**

    GitHub Repository Link:

    https://github.com/esam191/Intelligent-Transportation-System

    Project milestone 2 directory:

    https://github.com/esam191/Intelligent-Transportation-System/tree/main/Project%20Milestone%202/Esam