Distributed Systems Project 2 (Kafka Integration)
Group 53

Vishwas Nalka
50419408
vishwasn@buffalo.edu

Vishal Raman
50376944
vraman2@buffalo.edu

This Project aims to build a one-stop application to view data from multiple social media platforms viz. Twitter, Reddit. For this project, the end-to-end work flow for Twitter and Reddit is implemented for a Publisher-Subscribe model, with multiple publishers, multiple subscribers using Kafka Cluster (multiple broker nodes).

Design:

1. Producer: This is a flask application deployed as a docker container. Data is fetched for each topic using Twitter and Reddit API, and the data is sent to the Kafka broker as messages to each kafka topics (e.g. for the topic cricket, data is fetched from both platforms and encapsulated as a single event message which is then sent using a **kafka producer API**. The consumer can then retrieve the data for the platforms it has subscribed to.)

2. Subscriber: A flask application using Python, which creates a basic UI for the user to enter their inputs and subscribe to topics. This application is deployed as a docker container. There is a container for every Subscriber, run using the same docker Image. Each subscriber requests for available topics from the Kafka Broker using a kafka **consumer API call**. Then, the Kafka Broker returns a list of topics. The subscriber then subscribes to multiple topics. For each topic we maintain a consumer group. And a consumer belonging to this group pulls the events data that it has subscriber to, from the broker, using the **kafka consumer API**. Then the events data is displayed on the UI. The Subscribers can unsubscribe a few topics as well.

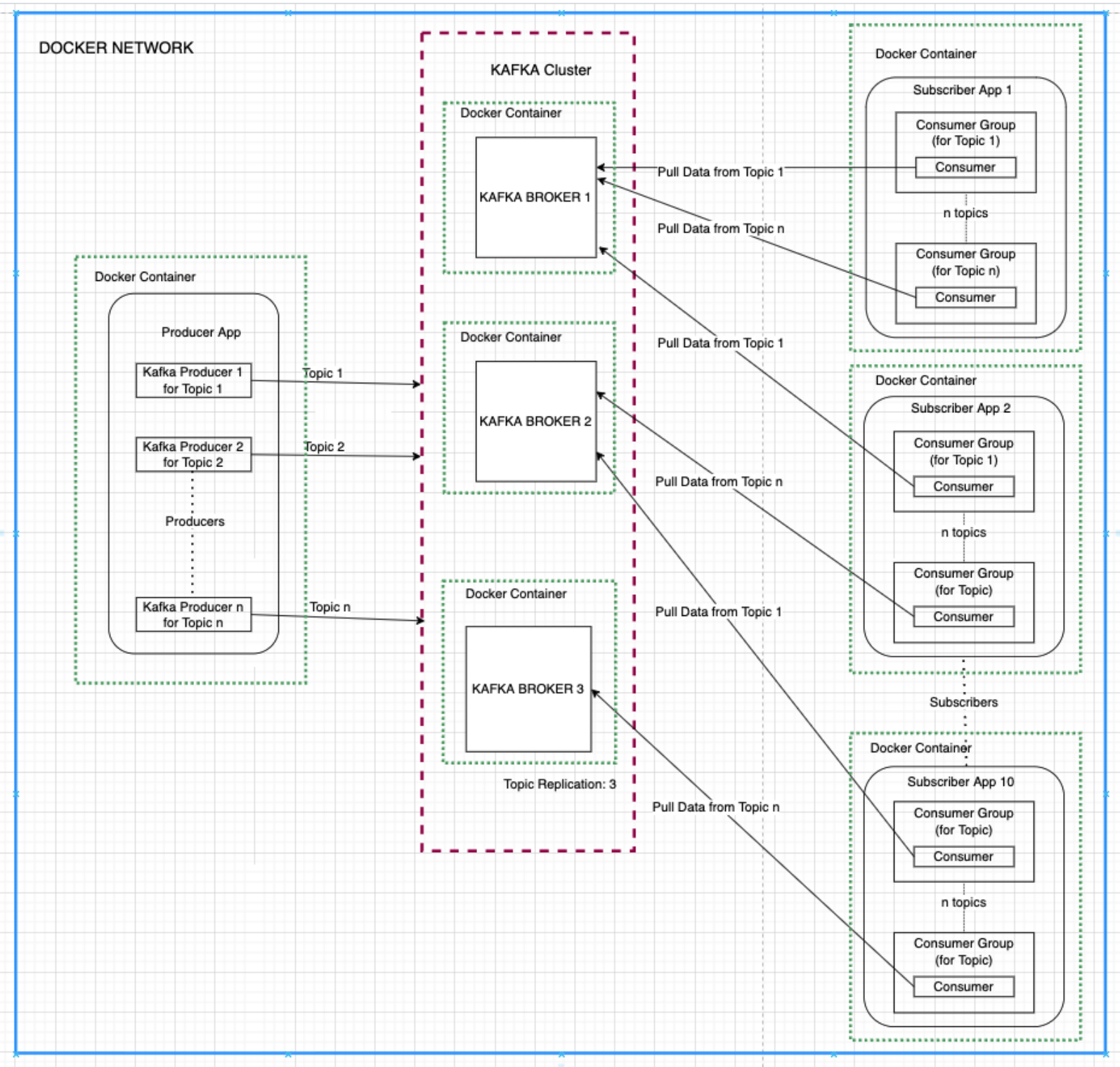
3. **Broker:** There are three Kafka brokers, and each Kafka broker stores the topics received from the Producer. The broker persists the event data and consumers can pull this data as and when they require it.

4. **Topics:** There can be any number of topics. At application startup, producer application starts sending the data for three topics. More topics can be advertised and published to the broker. The kafka topic replication and partition count is set to 3.

Dependencies and requirement:

- Docker (images used: wurstmeister/zookeeper, wurstmeister/kafka)
- Docker Compose
- Python3
- Flask
- Kafka-Python

Architecture Diagram:



Kafka Configuration & Integration Design:

1. Docker compose has been used to configure and start multiple containers in one go. Docker Networking concept is used to ensure all containers can identify and communicate with each other through ports.
2. Two containers of Zookeeper are spawned for the sake of redundancy. Even if one of the zookeeper containers fails due to memory constraints of our local system, the kafka cluster will still function.
3. Three Kafka broker nodes are configured in the docker-compose.yml file provided in the project folder. The following configurations of Kafka were set:
    a. KAFKA_NUM_PARTITIONS: number of default partitions for any kafka topic. **This is set to 3.**
    b. KAFKA_DEFAULT_REPLICATION_FACTOR: Default replication factor for the topic. **This is set to 3.**
    c. KAFKA_BROKER_ID: To configure the broker identifier of each broker node
    d. KAFKA_LISTENERS: To configure the listeners on to which the kafka listens
4. Each topic for which data is to be fetched is mapped to a Kafka topic. Both twitter and reddit data is combined into 1 JSON for that topic and sent to kafka broker as a serialized message. The message is produced every 3 seconds and sent over to the broker.
5. Every subscriber will have a container group for each topic. Kafka-python consumer is created within this group and the event data is pulled from the broker.
6. Docker-compose is used to spawn 3 broker nodes in one go and to establish a cluster. Docker-compose is also used to spawn 10 subscribers at one go.

Steps to instantiate the system:

1. Create a docker network which will be used by all the containers. Run the below command

- ***docker network create project2_network***

This creates a network with name project2_network

2. Deploy the Kafka Cluster: Run the docker-compose command on the .yml file in the root project folder:

- ***docker-compose -f docker-compose.yml up -d***

This will bring up a 3 broker nodes kafka cluster.

3. Go to the Producer folder and build a docker image using the given Dockerfile. This will bundle all the code and create a flask application image. Then run it using docker run and also attach it to the earlier created network. Run the below command.

- ***docker build . --tag producer***
- ***docker run --publish 8990:8990 --name producer --net project2_network producer***

To trigger the events data publish operation, send a CURL POST request to the ***http://localhost:8990/publish***. Now the Publisher Application will start publishing the event data for the **3 predefined topics ('cricket', 'sports', 'football').** (Note: Additional topics can be advertised. More explained below)

3. Go to the Subscriber folder and build a docker image using the given Dockerfile. This will bundle all the code and create a flask application image. Then run it using the provided .yml file. Run the below command.

- ***docker build . --tag subscriber***
- ***docker-compose -f docker-compose_subscribers.yml up -d***

This will start 10 subscribers, each in a separate container. All the above containers that are run in the same docker network.
Every subscriber is mapped to 8901, 8902, …. , 8910 of the localhost (note: the flask application in docker container will be running in 8991 in each container). Go to ***http://localhost:8901*** to view the subscriber UI.

Interaction with the application:

- Trigger a CURL POST request to the ***http://localhost:8990/publish***. Now the Publisher will start publishing the event data for the predefined topics.
- Every subscriber is mapped to 8901, 8902, …. , 8910 of the localhost (note: the flask application in docker container will be running in 8991 in each container). Go to ***http://localhost:8901*** to view the subscriber UI.
- Go to the subscriber UI, and subscribe to the topics along with the social media platform you want. Refresh the page every time to view updates.
- Now the subscriber will be able to see the events data for the subscribed topics.
- The subscriber can unsubscribe to a few topics as well. Then the event data will not be seen for the unsubscribed topics.

- New topics can be advertised as well, by sending a POST request to ***http://localhost:8990/advertise***. In the body of the API call, give the new topics to advertise.

```
{
        "topics": ["new_topic1", "new_topic2"]
}
```

Contribution by each team member:

- Consumer API code implementation: Vishal Raman
- Subscriber App Dockerization: Vishal Raman
- Producer API code implementation: Vishwas Nalka
- Producer App Dockerization: Vishwas Nalka
- Advertise Functionality: Vishwas Nalka, Vishal Raman
- Inter-networking using Docker Networks: Vishwas Nalka, Vishal Raman (Place everything in a single network)
- Kafka Cluster (3 broker nodes) setup: Vishwas Nalka, Vishal Raman (using docker-compose.yml):
- 10 Subscriber creation in one go: Vishwas Nalka, Vishal Raman (using docker-compose.yml)