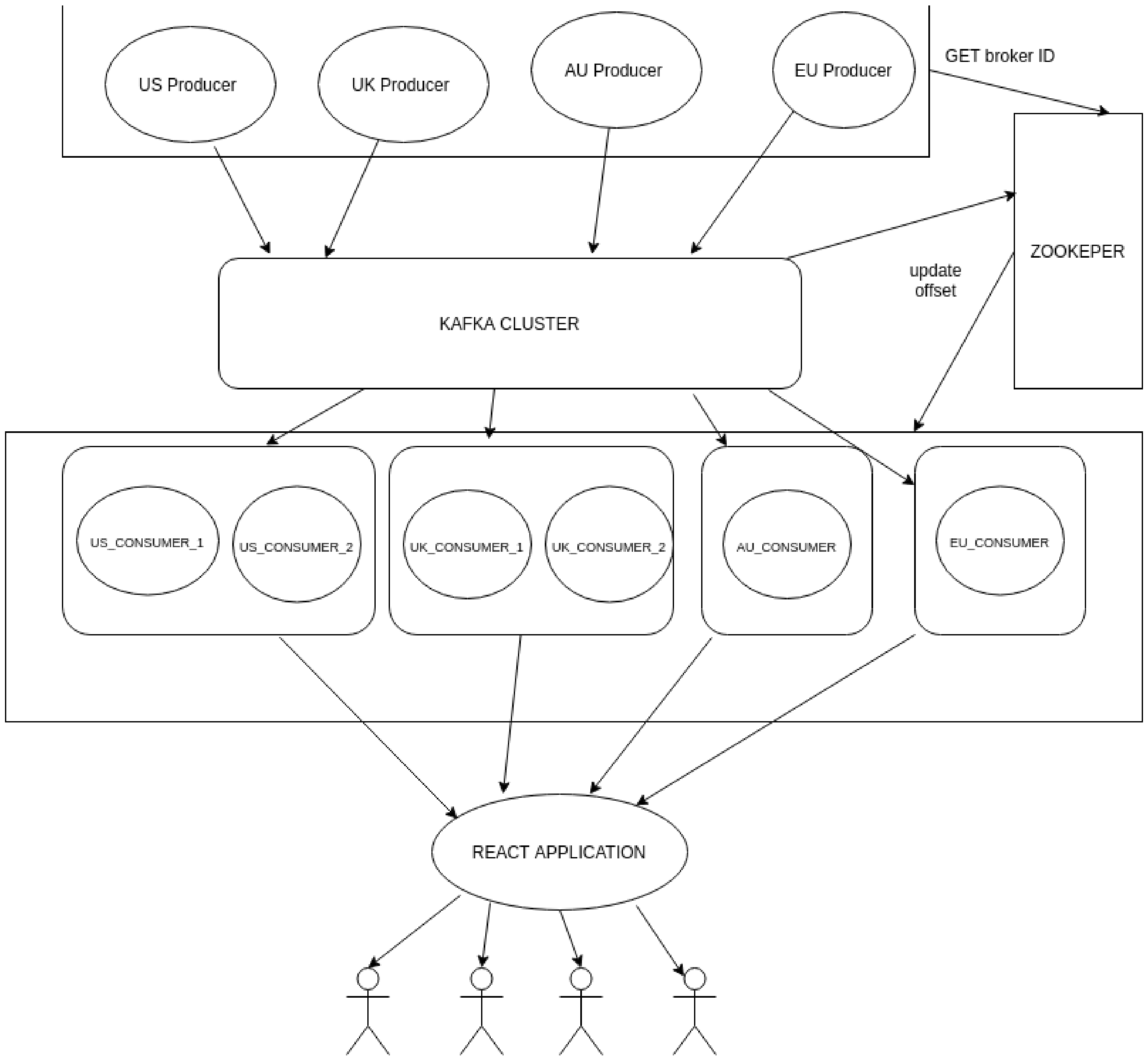
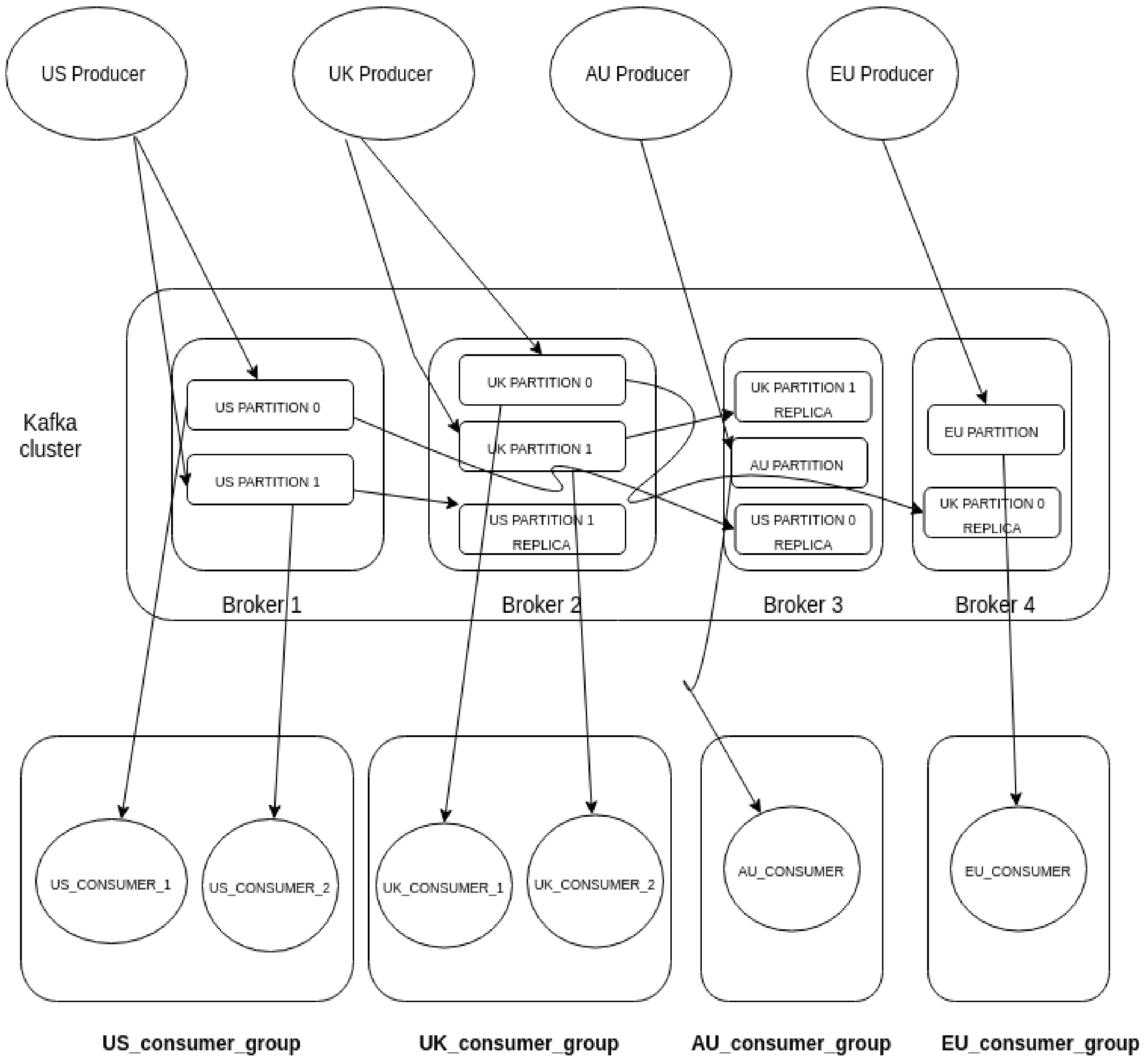
Architectural Overview



In the diagram, all the consumer groups are part of the consumer_server. The react application is connected to the consumer_server over ws protocol. Depending on what subscriptions the user has, the listeners on the client will be modified.

INTERACTION BETWEEN CONSUMERS AND PARTITIONS



In the diagram, the US broker has two partitions. The responsibility of consuming the two partitions will be shared among the two consumers of the US consumer group. The two partitions will also have a replica each in other brokers. This is achieved by setting a replication factor of 2 for the US topic. The EU and AU topics will not have any copies as replication factor is set to 1. The data will be written to the master partitions only and data will be synced among the replicas.

In this system, the US and UK producers both produce messages in a Round robin fashion to their respective partitions. For e.g, message 1 will be published to partition 0 of US topic and message 2 will be published to partition 1 of US topic. Same goes for the UK topic as well. But in the case of EU and AU producers, I have just used one partition without a replica as these events are not of vital importance considering the small amount of bookmaker tips.

Docker containers

- Consumer server
    - This container comprises the message consumers. There are a total of 4 consumer groups. The US and UK topic each has two partitions with a replica each. So I have used two consumers in the UK consumer group and US consumer group. In the case of EU and AU topics, there is only one consumer in each group.

Producers: These containers fetch api data and publish messages to the Kafa Cluster

- Producer_1
    - This producer is responsible for producing events from all UK bookmakers
- Producer_2
    - This producer is responsible for producing events from all US bookmakers
- Producer_3
    - This producer is responsible for producing events from all AU bookmakers
- Producer_4
    - This producer is responsible for producing events from all EU bookmakers
- Server
    - This server is responsible for accepting subscription and authentication requests and sending it to the mongo db for future retrieval.

- Sub1
  - This is a React application to provide an interface to the users. The users can use the multi select dropdown to choose the subscriptions that they are interested in. There are total of 4 subscription topics, AU, EU, US and UK
- Topic_creator
  - I have used Kafka JS library to create topic partitions and replicas. Alternatively the partitions can also be created by setting the Kafka configuration inside the docker compose yaml file.

Package Dependencies

- "axios": "^0.23.0",
- "kafkajs": "1.12.0",
- "socket.io-client": "^4.3.0"
- "socket.io": "^4.3.0",,
- "bcryptjs": "^2.4.3",
- "body-parser": "^1.19.0",
- "cors": "^2.8.5",
- "express": "^4.17.1",
- "express-session": "^1.17.2",
- "mongoose": "^5.9.29"
- "babel-preset-es2015": "^6.24.1",
- "bootstrap": "^5.1.1",
- "multiselect-react-dropdown": "^2.0.7",
- "react": "^16.6.3",
- "react-bootstrap": "^2.0.0",
- "react-dom": "^16.6.3",
- "react-redux": "^5.1.1",
- "react-router-dom": "^4.3.1",

How to Run?

Run **sudo docker-compose** build from the root of the project to build all the images. Run **sudo docker-compose up** from root to start all the containers using docker-compose