***FRAMEWORKS USED***

* Spring Boot (server-side)
* Apache Kafka – pub/sub central middleware
* HTML (client-side) and JavaScript

***HOW TO DEPLOY?***

* Open a terminal and cd into the project folder -> “***traveller-kafka***”
* Type the command -> “**docker-compose up -d**” -> all containers and images would be created in this step and all the producer, consumer and the kafka-cluster would spin up.
* 10 different consumers are hosted on:
  + Consumer 1 🡪 localhost:8080
  + Consumer 2 🡪 localhost:8081
  + Consumer 3 🡪 localhost:8082
  + Consumer 4 🡪 localhost:8083
  + Consumer 5 🡪 localhost:8084
  + Consumer 6 🡪 localhost:8085
  + Consumer 7 🡪 localhost:8086
  + Consumer 8 🡪 localhost:8087
  + Consumer 9 🡪 localhost:8088
  + Consumer 10 🡪 localhost:8089
* The producers are hosted on:
  + Producer 1 🡪 localhost:8050
  + Producer 2 🡪 localhost:8060
  + Producer 3 🡪 localhost:8070
* 3 Kafka broker nodes have been hosted and one zookeeper to monitor and maintain interaction between these broker nodes.
* This entire setup belongs to a docker-network named kafka-cluster

***CLIENT-SERVER INTERACTION***

* The Producers and Consumers are completely **independent** of each other.
* Producers and Consumer are individual spring boot applications.
* Whenever a producer publishes an event (for some topic) to the Kafka-cluster, all the consumers subscribed to the topic will receive the event.
* Since the producers and consumers have no dependency on each other, this conforms to the true publish subscribe pattern.

***ABOUT THE APPLICATION***

* This application is a complete representation of a distributed publish-subscribe system. Loose-coupling is achieved in the application by using **Kafka Consumer client** and **Kafka Producer** client which consumes and publishes records to and from the Kafka cluster.
* The application has been **completely** dockerized by using **17** docker containers. They have been linked using docker-compose and docker networking.
* The three topics that subscribers can choose from are -> India, Egypt, and Singapore.
* The three publishers involved in the system which poll topic data from an external API -> travelbriefing.org

***API’S IMPLEMENTED AND THE FLOW***

* **publish**(topic) -> This producer level endpoint will poll the topic data from the external API and publish the data as an event to the kafka-cluster. As soon as the event is registered by the kafka-cluster, all the consumers/subscribers who are subscribed to the topic will receive the event/message.
* **notify**() -> this endpoint gets all the topic messages for the consumer.
* **subscribe** (topics) – this consumer level endpoint stores topic subscription data for the consumer by calling the subscribe(topics) method provided by the Kafka Consumer client.
* **unsubscribe** (topics) – this consumer level endpoint unsubscribes topics from the consumer’s subscriptions by calling the unsubscribe() method provided by the Kafka Consumer client.
* **advertise**(t) – displays all the topics which could be at the user’s disposal to select from in the future.
* **deadvertise** – clears the list of topics advertised

SYSTEM DIAGRAM

Diagram

Description automatically generated