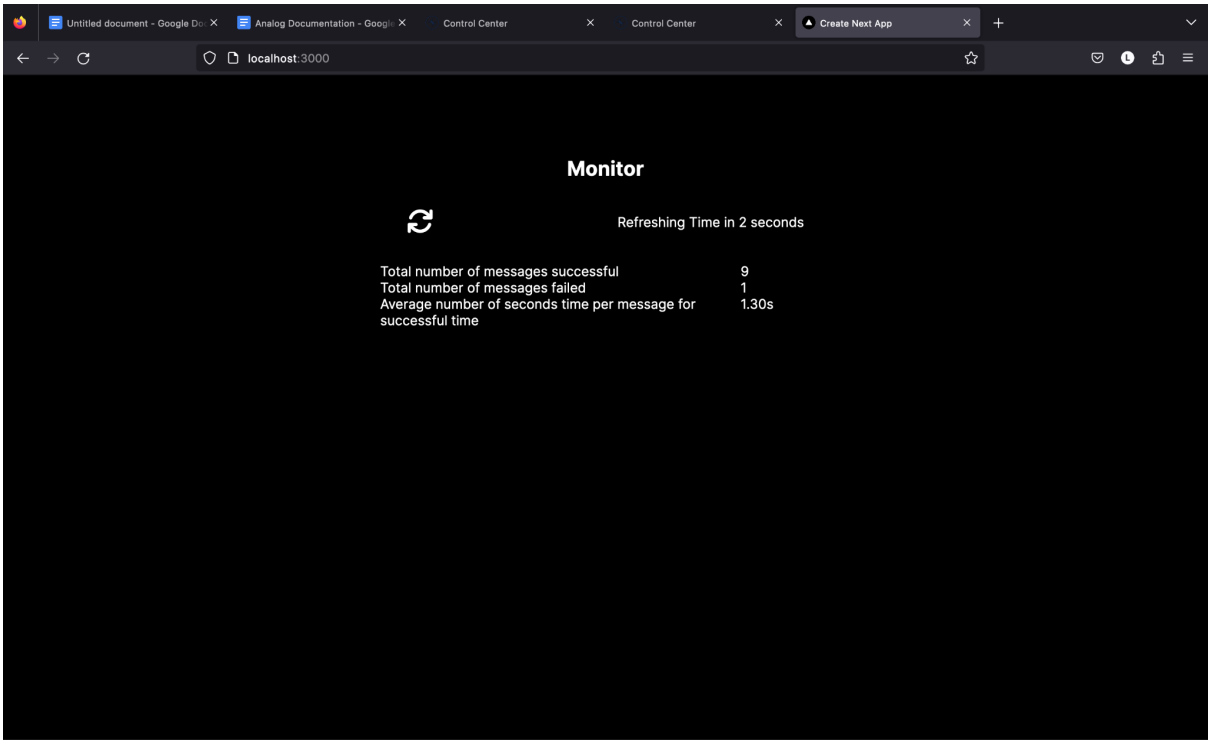
SMS Simulation Exercise

Monitor: (Monitor Folder - Next.js application)

To display the current status of the no of messages delivered.

The UI is built on Next.JS framework



Producer: (app.py Flask application - Message producer Folder)

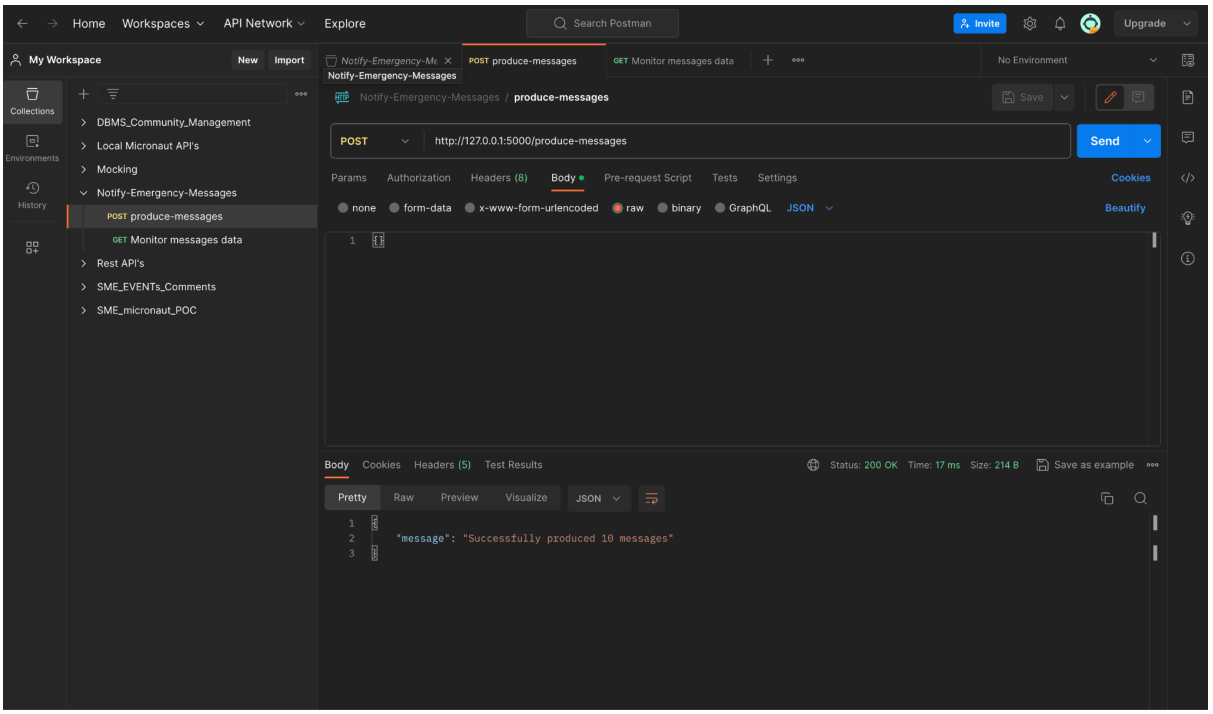
Creates the given no of messages and pushes on to the message queue on to the kafka server. The messages will be pushed on to the topic - **sms-messages.** The senders will be consuming the messages and the distribution of the messages will be done by the kafka server depending on

the no of partitions hence the no of partitions for the topic should be equal to the no of senders in the system.

Each sender will send the status update to the message producer back in another kafka topic - **sender-status-updates,** the main producer will be subscribing to this topic to listen for the messages sent from various senders. Upon listening to this update, the main producer will log the details and store it for the purpose of the monitor to show the details in real time. The producer starts the daemon thread to listen for the status updates from the senders.

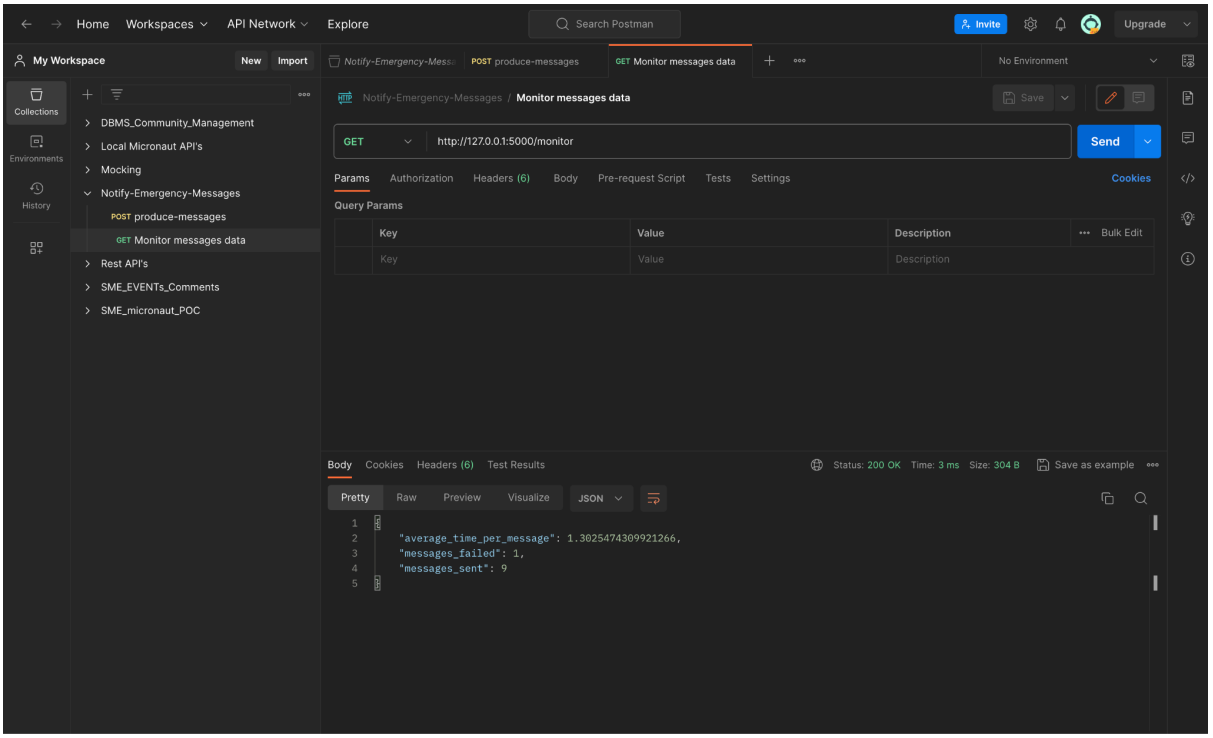
The messages are produced by the main producer with

http://127.0.0.1:5000/produce-messages



The monitor could read the data from the producer to show on to UI by consuming data from

http://127.0.0.1:5000/monitor



The main producer will be running as a flask application which connects to the kafka server. **MessageProducer**

Senders:(Sender.py)

Each and every sender will be started as an individual thread. Each thread will be running and consuming the messages from the main producer. The senders are started by executing **sender.py.**

Kafka Server:(confluent-7.5.1 Folder)

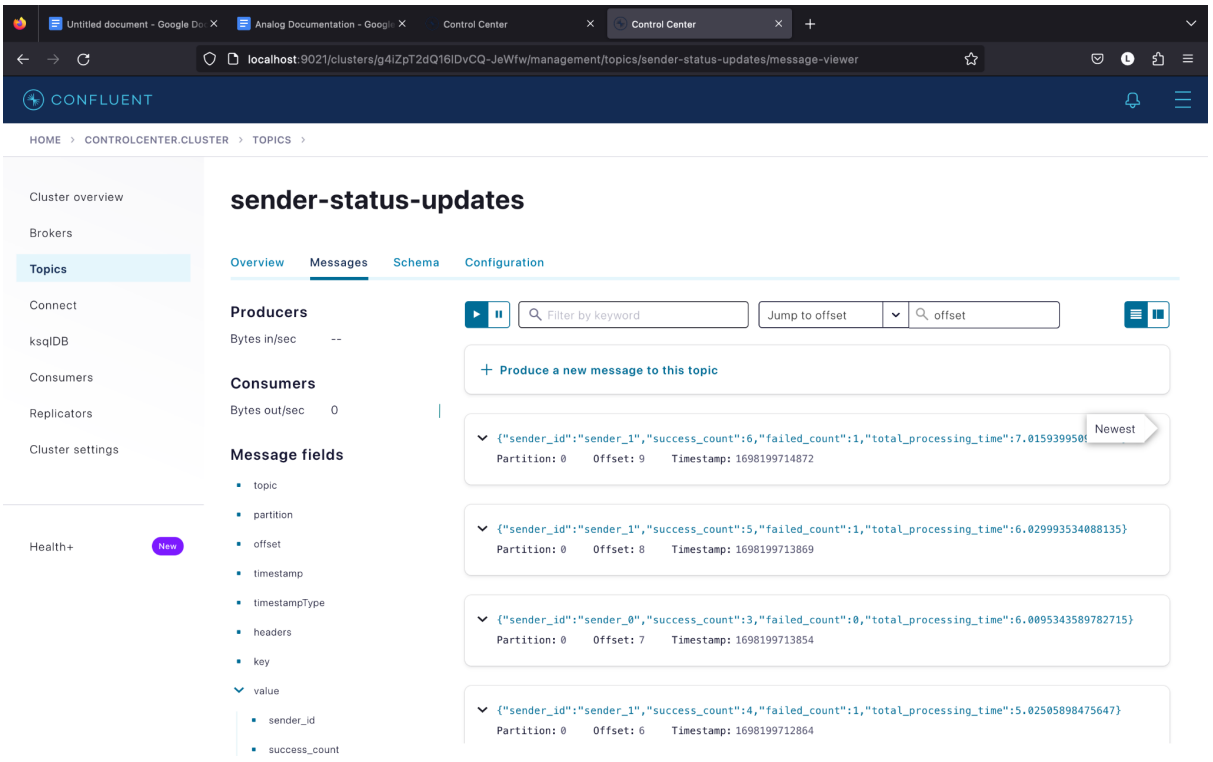
The kafka server would be up on the

To start the server:
“confluent local services start” - command

To stop the server:
“confluent local services stop” - command

http://localhost:9021/clusters/

Then we can check here for real time messages on different topics which are generated.



Senders are being started as individual threads and each thread will be acting as a consumer taking the messages and working up on delivering the messages. After delivering a message each sender will update the main-message

producer with a message reporting the status of the message by the sender.