Cloud Engineer Coding Test - Analog Devices

Problem Statement

The objective is to simulate sending a large number of SMS alerts, like for an emergency alert service. The simulation consists of three parts:

1. A producer that that generates a configurable number of messages (default 1000) to random phone number. Each message contains up to 100 random characters.
2. A sender, who pickups up messages from the producer and simulates sending messages by waiting a random period time distributed around a configurable mean. The sender also has a configurable failure rate.
3. A progress monitor that displays the following and updates it every N seconds (configurable):
   a. Number of messages sent so far
   b. Number of messages failed so far
   c. Average time per message so far

One instance each of producer and progress monitor will be started while a variable number of senders can be started with different mean processing time and error rate settings.

Flow of the program

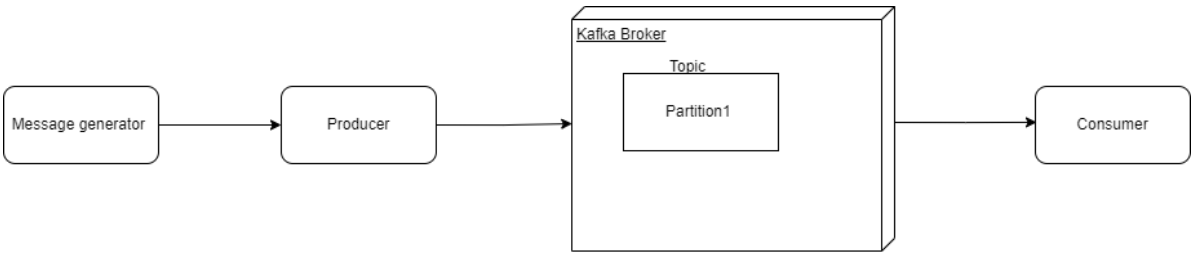
The objective is to stimulate sending large number of SMS to a random number.

Simulation takes place in three parts:

Producer - To send messages to a Kafka topic

Consumer/Sender - Receives the generated messages from the Kafka topic specified.

Progress Monitor - Once the consumer process, it keeps track of the number of messages send so far, number of messages failed so far and average number of messages send.



Once you run the producer.py file, it will call getdata() function from data.py and the corresponding messages will be sent to Kafka broker to the specified topic's partition.

As soon as you run your consumer.py file, it will wait to till there are no messages to read, once messages starts to get publish in a partition, the consumer will read the message and output it. Once the consumer instance is started, after every N seconds it will also output the total number of messages sent, failed and average of the numbers sent.

Note - Always make sure that the topic name for producer and consumer are same.

Why to use Kafka?

Because the problem statement mentions that multiple instances of consumer/sender can be started, and Kafka allows for data processing to be distributed across many consumer instances. And each consumer receives messages in order. Major advantages comes from the fact that it combines messaging queue and publish subscribe approaches.

Technologies Used

Python 3.8

Apache Kafka 3.3.1

kafka-python

Prerequisites

To run this program in your system locally you will need the following things installed in your system.

Install Python

```
sudo apt install python3
```

Install Java

```
sudo apt install openjdk-11-jre-headless -y
```

Install Apache Kafka

Download source file. To find latest version you can go to the official website: https://kafka.apache.org/downloads

```
wget https://downloads.apache.org/kafka/3.3.1/kafka-3.3.1-src.tgz
```

Extract the file using the following command:

```
tar -xvzf kafka-3.3.1-src.tgz
```

Now you will find Kafka installed in your system. The next step will be to update the server.properties file

Follow the steps:

1 Open the server.properties which you will find in bin folder of Kafka

```
vim server.properties
```

2. Uncomment the following like:

```
#advertised.listeners=PLAINTEXT://localhost:9092
```

Install kafka-python

```
pip install kafka-python
```

All the dependencies are installed

Understanding Kafka

Kafka is a popular open-source Publish/Subscribe messaging system that is used for building streaming analytics platforms and data integration pipelines. Kafka is both a queue for parallelizing tasks and a messaging-oriented middleware for service integration. The Kafka message broker (cluster) ingests and stores streams of messages (records) from event producers, which are later distributed to consumer services asynchronously when requested.

Producers publish events to the Kafka instance or instances since a Kafka cluster can be either single or multi-node. Kafka stores the messages in a topic. The messages in the topic are organized in an immutable sequence (Python tuple object) based on when the messages were created. Downstream applications can then subscribe to the topics that they are interested in. Then as we create Consumers, they read the message from the topic.

- **Topic**: A named resource to which a particular stream of messages is stored and published.
- **Producer**: A client application that creates and publishes records/messages to a Kafka topic(s).
- **Consumer**: A client application that subscribes to a topic(s) to receive data from it.
- **Message**: The combination of data and associated metadata that a producer application writes to a topic and is eventually consumed by consumers

Executing program

1. Start Kafka on your local system

   1.1 Start Zookeeper

   Open a new terminal and start your zookeeper

   ```
   bin/zookeeper-server-start.sh config/zookeeper.properties
   ```

1.2 Start Kafka Server

```
JMX_PORT=8004 bin/kafka-server-start.sh config/server.properties
```

2. Create topic where you want to publish the messages

   ```
   bin/kafka-topics.sh --create --topic sms1 --bootstrap-server localhost:9092
   ```

   Successfully created topic named sms

3. Start your producer and consumer

   In this project, we will use single producer, single topic and multiple consumer

   3.1 Start your consumer

   In a new terminal run the following command

   ```
   python consumer.py
   ```

3.2 Start your producer

In another terminal, run the producer

```
python producer.py
```

After this you will see the consumer will print the messages along with that after every N second will show you the progress monitor.

Understanding input.txt file

You can modify input.txt file according to your input requirements.

First you give the number the messages which you want to send. Default is 1000

Then set the processing time and error rate for your consumer/sender

Then set N, where N is the number of seconds after which you want to see the progress.