Python engineer code challenge

This is a Python engineer tech challenge for Quix. Please read the exercise in full before beginning work. If you have questions or need clarification, you are free to reach out to us at any time.

Rules

At Quix our team values the ability to think through a problem and solve it with a fine balance of simplicity and innovation. Write code that is easy to read and understand while following best practices.

Timing

- You will have a week to complete and submit your work.

Delivery

Please return your solution as a compressed zip file containing your git repo.

Review

- After submitting your work, an engineer of ours will review your work.
- Should you pass the technical challenge, we will set up a video interview to discuss your solution, so please prepare to explain your work.

How this challenge will be evaluated

- Your ability to develop a working solution
- There is more than one correct solution. Focus on solving the task in a way that reflects your strengths.
- Your decisions and thinking are important to us. Comments and documentation will be reviewed and should provide additional context and clarity for your solution.
- Source control and work methodology are important so please use git. Meaningful commit messages and organization will help us understand your solution better.
- This code challenge is designed to test your ability to work with new technologies and concepts.

The task

In this code challenge, we ask you to build a simple Python service that processes data in a Kafka topic.

We will run a data generator (code provided) to send messages to a local Kafka at a specified rate.
(We show you to set Kafka up below)

The message format of the generator is the following:

```
{ "Invoice":"489574", "StockCode":"21491",
"Description":"SET OF THREE VINTAGE GIFT WRAPS",
"Quantity":2, "Price":1.95, "Customer ID":13097.0,
"Country":"United Kingdom",
"InvoiceDate":1674744306.978004 }
```

Important properties are:

- **InvoiceDate** - in epoch format in seconds from 1970
- **StockCode** - we will aggregate data using StockCode as the key
- **Price**
- **Quantity**

> Calculate the total amount of goods sold per StockCode in the last 10 minutes in a streaming fashion by subscribing to the `data` topic and output message to the `agg` topic for each incoming message. Use a rolling window approach to calculate the aggregated value.

Imagine incoming messages as rows presented in the table. We converted the epoch column into time format for ease of explanation.

InvoiceDate	StockCode	Price	Quantity
10:32	1	10	2
10:33	2	20	1
10:37	1	10	2
10:45	2	20	1
10:46	2	20	3
10:47	1	10	2
10:48	2	20	1
10:49	1	10	3
10:50	2	20	1
10:51	2	20	3

We group it by StockCode:

InvoiceDate	StockCode	Price	Quantity
10:32	1	10	2
10:37	1	10	2
10:47	1	10	2
10:49	1	10	3

InvoiceDate	StockCode	Price	Quantity

10:33	2	20	1
10:45	2	20	1
10:46	2	20	3
10:48	2	20	1
10:50	2	20	1
10:51	2	20	3

And then we calculate the rolling window per StockCode as follows:

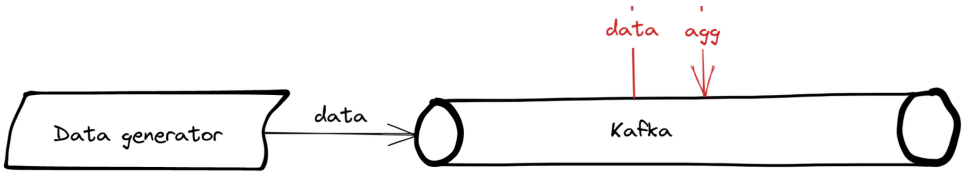
InvoiceDate	StockCode	Price	Quantity	Rolling sum
10:32	1	10	2	(10*2) = 20
10:37	1	10	2	20 + 20 = 40
10:47	1	10	2	20 + 20 = 40
10:49	1	10	3	20 + 30 = 50

💡 For every row, we calculate the sum of the total price for the current and previous rows received in the last 10 minutes. That is why the 10:49 row includes only 10:49 and 10:47 rows in aggregation.

Output message example:

```
{ "StockCode":"21491", "Epoch":1674744306.978004,
"10minutesSum":312312312.75 }
```

The architecture diagram:



Guide

To help you with the task, we have prepared the following guide:

1) Run local or any other Apache Kafka.

You can download the Apache Kafka binary from the Apache Kafka Download page. See Kafka Quick start guide.

💡 You need Java 8+ to run Kafka.

Here is an example of how to install and run Kafka in a Unix style OS:

Download and extract Kafka and start the zookeeper:

```
curl https://downloads.apache.org/kafka/3.3.2/
kafka_2.13-3.3.2.tgz --output kafka.tgz tar -xf kafka.tgz
cd kafka_2.13-3.3.2/ bin/zookeeper-server-start.sh config/
zookeeper.properties
```

In another window start the server

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

2) Create topics in the Kafka cluster

Create the `data` and `agg` topics:

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

```
agg    bootstrap server localhost:9092
```

Create two topics in the broker

3) Data generator

Install the required libraries:

```
pip install kafka-python pip install urllib3 pip install
pandas
```

Run the following Python script to continuously send data to Kafka:

```
import pandas as pd import json import datetime as dt from
time import sleep from kafka import KafkaProducer import
os import time file = "online_retail_II.csv" # Initialize
Kafka Producer Client producer = KafkaProducer(
bootstrap_servers=['localhost:9092'] ) interval = 0.1
print('Initialized Kafka producer at
{}'.format(dt.datetime.utcnow())) # Set a basic counter as
the message key counter = 0 while True: print("Starting
file restream...") for index, chunk in
pd.read_csv(file,encoding='latin1').iterrows(): # For each
chunk, convert the invoice date into the correct time
format chunk["InvoiceDate"] = time.time() key =
"invoice".encode() # Convert the data frame chunk into a
dictionary including the index chunkd = chunk.to_dict()
print(chunkd) # Encode the dictionary into a JSON Byte
Array data = json.dumps(chunkd,
default=str).encode('utf-8') # Send the data to Kafka
producer.send(topic="data", key=key, value=data) counter =
counter + 1 # Sleep to simulate a real time interval
sleep(interval) print(f'Sent record to topic at time
{dt.datetime.utcnow()}')
```

`online_retail_II.csv` file is provided in the email.

online_retail_II.csv 3829.6KB

4) Main part of the code challenge - create an

aggregator

Create a Python script with the following requirements:

- Subscribe to the `data` topic and aggregate the total amount of goods traded in the last 10 minutes per **StockCode,** using rolling window aggregation
- Output aggregated values to the output topic `agg` with the following structure:

```
{ "StockCode":"21491", "Epoch":1674744306.978004,
"10minutesSum":312312312.75 }
```

- Do it in a scalable and resilient way
  - Consider the consumer group concept and change `producer.py` to enable this service to scale horizontally and run in multiple instances
- Consider the application state and its recovery in case of restart
- Consider working with Kafka checkpointing and state together in recovery