Kafka Python Client

Estimated time needed: **30** minutes

Objectives

After completing this reading, you will be able to:

- List the common Apache Kafka clients
- Use `kafka-python` to interact with Kafka server in Python

Apache Kafka Clients

Kafka has a distributed client-server architecture. For the server side, Kafka is a cluster with many associated servers called broker, acting as the event broker to receive, store, and distribute events. It also has some servers that run "Kafka Connect" to import and export data as event streams. All the brokers until versions prior to 2.8 relied on another distributed system called ZooKeeper for management and to ensure all brokers work in an efficient and collaborative way.

However, Kafka Raft, or `KRaft`, is now used to eliminate Kafka's reliance on ZooKeeper for metadata management. It is a consensus protocol that streamlines Kafka's architecture by consolidating metadata responsibilities within Kafka itself using Kafka Controllers. `Producers` send or publish data to the `Topic` and the `consumers` subscribe to the topic to receive data. Kafka uses a TCP-based network communication protocol to exchange data between clients and servers.

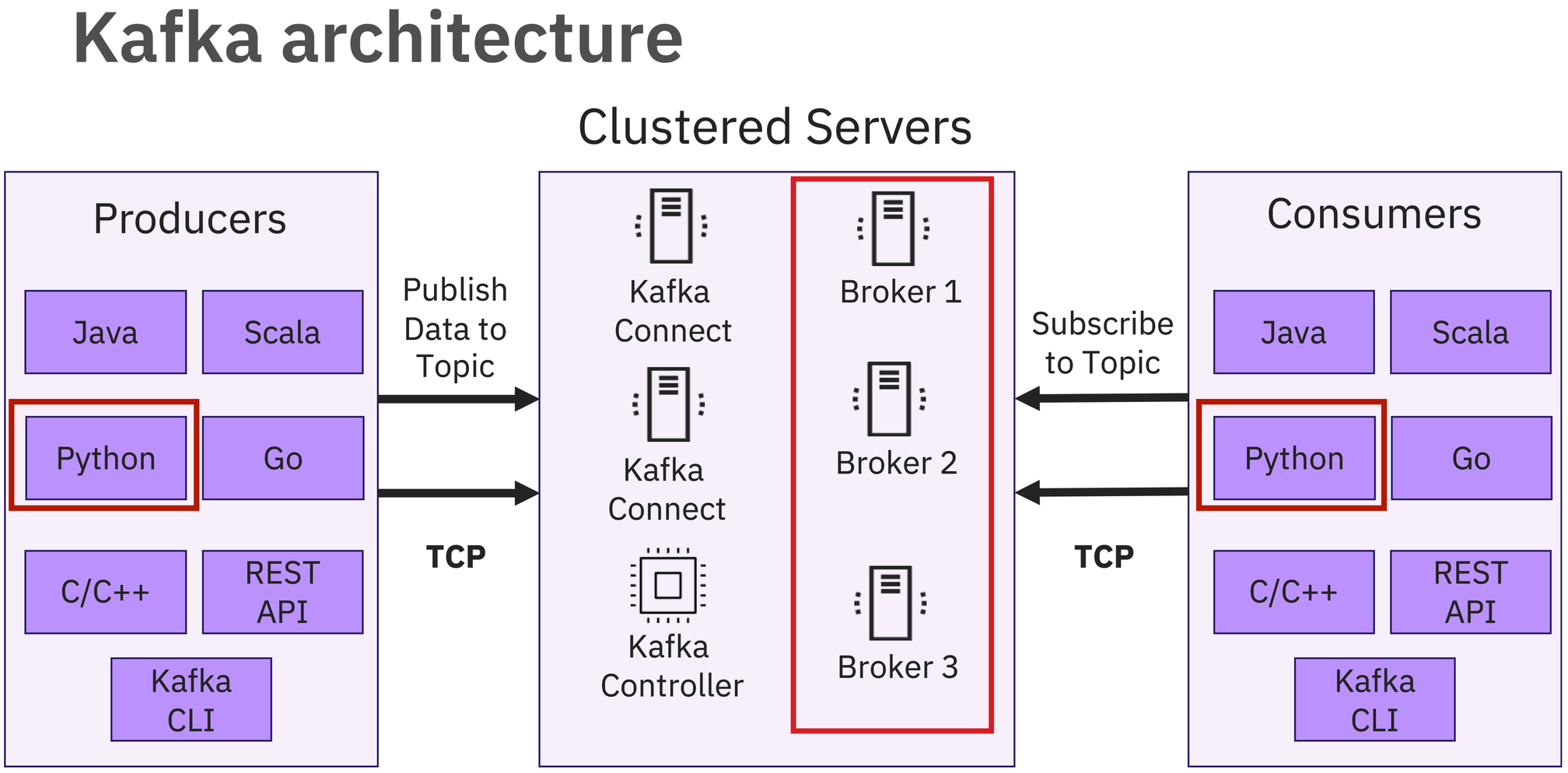
For the client side, Kafka provides different types of clients, such as:

- Kafka CLI, which is a collection of shell scripts to communicate with a Kafka server
- Many high-level programming APIs such as Python, Java, and Scala
- REST APIs
- Specific third-party clients made by the Kafka community

You can choose different clients based on your requirements.

Kafka Python

Let's focus on the **Kafka Python client** called `kafka-python`.



Note: Code snippets provided in this reading are just for your reference but not the complete working code.

The "kafka-python" package

`kafka-python` is a Python client for the Apache Kafka distributed stream processing system, which aims to provide similar functionalities as the main Kafka Java client. With `kafka-python`, you can easily interact with your Kafka server such as managing topics, publish, and consume messages in Python programming language.

You must install `kafka-python` using `pip3 installer` to use it with a Python client.

```
pip3 install kafka-python
```

Next, let's review use cases for the main functions provided by the `kafka-python` package.

"KafkaAdminClient" class

The main purpose of `KafkaAdminClient` class is to enable fundamental administrative management operations on kafka server such as creating/deleting topic, retrieving, and updating topic configurations and so on.

Let's check some code examples:

1. To use `KafkaAdminClient`, you first need to define and create a `KafkaAdminClient` object.

   ```
   admin_client = KafkaAdminClient(bootstrap_servers="localhost:9092", client_id='test')
   ```

   - `bootstrap_servers="localhost:9092"` argument specifies the host/IP and port that the consumer should contact to bootstrap initial cluster metadata
   - `client_id` specifies an id of current admin client

2. The most common use of the `admin_client` is managing topics, such as creating and deleting topics. To create topics, you must first define an empty topic list:

   ```
   topic_list = []
   ```

3. Then, you use the `NewTopic` class to create a topic with name, partition, and replication factors. For example, name equals `bankbranch`, partition nums equals 2, and replication factor equals 1.

   ```
   new_topic = NewTopic(name="bankbranch", num_partitions= 2, replication_factor=1)
   topic_list.append(new_topic)
   ```

4. You can use `create_topics(...)` method to create topics.

   ```
   admin_client.create_topics(new_topics=topic_list)
   ```

   Note: The `create` topic operation used above is equivalent to using `kafka-topics.sh --topic` in Kafka CLI client.

Describe a topic

1. After the topics are created, you can check its configuration details using the `describe_configs()` method.

   ```
   configs = admin_client.describe_configs(
       config_resources=[ConfigResource(ConfigResourceType.TOPIC, "bankbranch")])
   ```

   Note: The `describe` topic operation used above is equivalent to using `kafka-topics.sh --describe` in Kafka CLI client.

KafkaProducer

Having created the new `bankbranch` topic, you can start producing messages.

For `kafka-python`, you will use `KafkaProducer` class to produce messages. Since many real-world message values are in the JSON format, let's look at how to publish JSON messages as an example.

1. First, let's define and create a `KafkaProducer`.

   ```
   producer = KafkaProducer(value_serializer=lambda v: json.dumps(v).encode('utf-8'))
   ```

   Since Kafka produces and consumes messages in raw bytes, you need to encode our JSON messages and serialize them into bytes. For the `value_serializer` argument, you will define a lambda function to take a Python dict/list object and serialize it into bytes.

2. Then, with the `KafkaProducer` created, you can use it to produce two ATM transaction messages in JSON format as follows:

   ```
   producer.send("bankbranch", {'atmid':1, 'transid':100})
   producer.send("bankbranch", {'atmid':2, 'transid':101})
   ```

   The first argument specifies the topic `bankbranch` to be sent and the second argument represents the message value in a Python dict format and will be serialized into bytes.

   Note: The above producing message operation is equivalent to using `kafka-console-producer.sh --topic` in Kafka CLI client.

KafkaConsumer

In the previous step, you published two JSON messages. Now, you can use the `KafkaConsumer` class to consume the messages.

1. Define and create a `KafkaConsumer` subscribing to the topic `bankbranch`:

   ```
   consumer = KafkaConsumer('bankbranch')
   ```

2. Once the consumer is created, it will receive all available messages from the topic `bankbranch`. Then, you can iterate and print them with the following code snippet:

   ```
   for msg in consumer:
       print(msg.value.decode("utf-8"))
   ```

   Note: The above consuming message operation is equivalent to using `kafka-console-consumer.sh --topic` in Kafka CLI client.

Authors

Lavanya T S
Yan Luo