Kafka Python Client

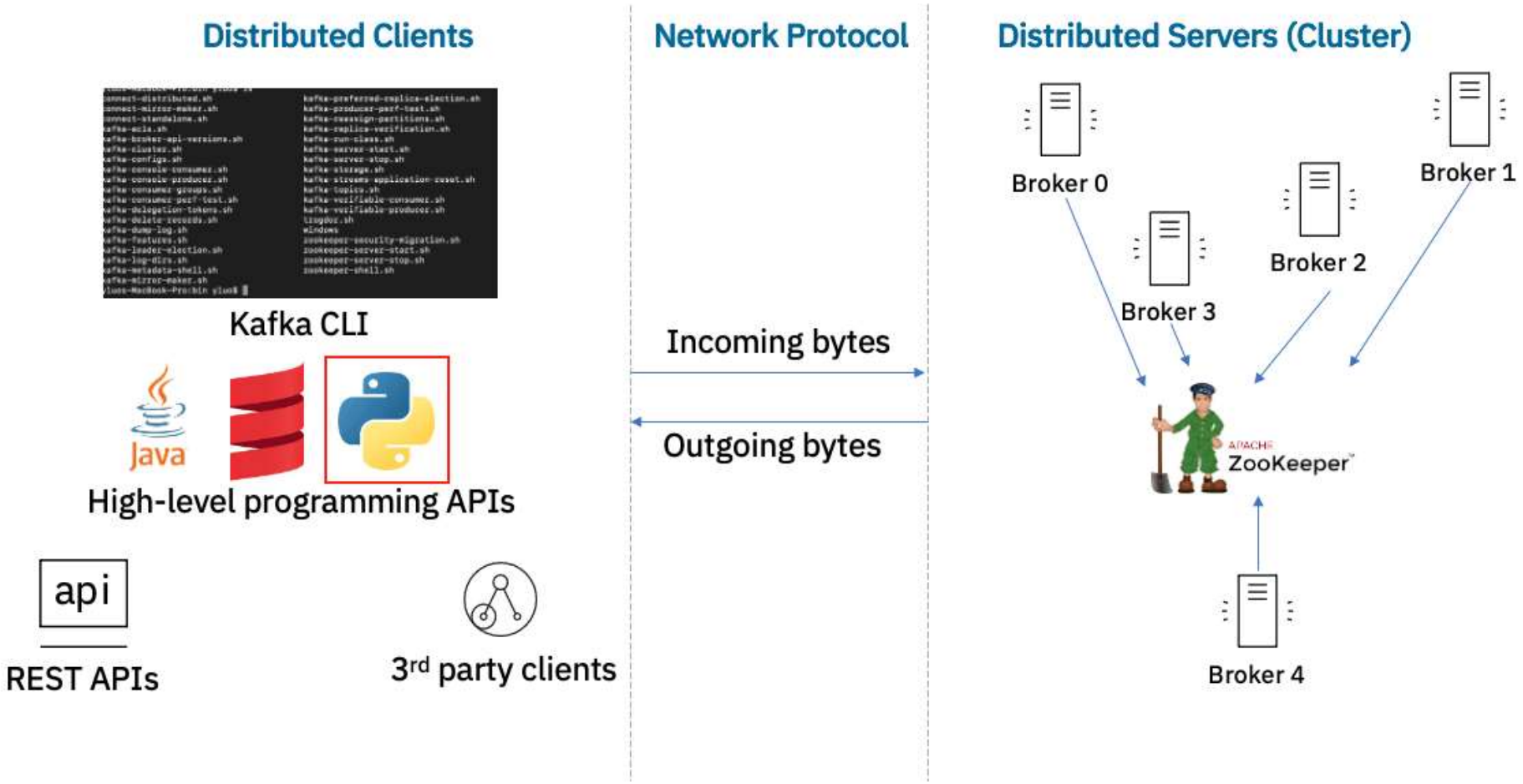
Estimated time needed: **30** minutes

Objectives

After reading this article, 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. All those brokers are managed by another distributed system called ZooKeeper to ensure all brokers work in an efficient and collaborative way.

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 3rd party clients made by the Kafka community

You can choose different clients based on your requirements. In this reading, we will be focusing on a Kafka Python client called `kafka-python`

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

`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.

Install `kafka-python`

Install `kafka-python` is similar to other regular Python packages:

```
1. 1
```

```
1. pip install kafka-python
```

Copied!

Next, let's see the use cases of 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 concrete code examples:

Create a `KafkaAdminClient` object

To use `KafkaAdminClient`, we first need to define and create a `KafkaAdminClient` object:

```
1. 1
```

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

Copied!

- `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

Create new topics

Next, the most common usage of `admin_client` is managing topics such as creating and deleting topics.

To create new topics, we first need to define an empty topic list:

```
1
```

```
topic_list = []
```

Copied!

Then we use the `NewTopic` class to create a topic with name equals `bankbranch`, partition nums equals to 2, and replication factor equals to 1.

```
1
2
```

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

Copied!

At last, we can use `create_topics(...)` method to create new topics:

```
1
```

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

Copied!

Above `create` topic operation is equivalent to using `kafka-topics.sh --topic` in Kafka CLI client:

```
1
```

```
"kafka-topics.sh --bootstrap-server localhost:9092 --create --topic bankbranch  --partitions 2 --replication_factor 1"
```

Copied!

Describe a topic

Once new topics are created, we can easily check its configuration details using `describe_configs()` method

```
1
2
```

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

Copied!

Above `describe` topic operation is equivalent to using `kafka-topics.sh --describe` in Kafka CLI client:

```
1
```

```
kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bankbranch
```

Copied!

KafkaProducer

Now we have a new `bankbranch` topic created, we can start produce messages to the topic.

For `kafka-python`, we will use `KafkaProducer` class to produce messages.
Since many real-world message values are in the format of JSON, we will show you how to publish JSON messages as an example.

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

```
1. 1
```

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

Copied!

Since Kafka produces and consumes messages in raw bytes, we need to encode our JSON messages and serialize them into bytes.

For the `value_serializer` argument, we define a lambda function to take a Python dict/list object and serialize it into bytes.

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

```
1. 1
```

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

Copied!

```
1. 1
```

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

Copied!

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.

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

```
1. 1
```

```
1. kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankbranch
```

Copied!

KafkaConsumer

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

We just need to define and create a `KafkaConsumer` subscribing to the topic `bankbranch`:

```
1. 1

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

Copied!

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

```
1. 1
2. 2

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

Copied!

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

```
1. 1

1. kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch
```

Copied!

Summary

In this reading, you have learned how to use `kafka-python` package to perform some main operations such as create or describe topic, produce and consume messages with a Kafka server.

Authors

Yan Luo

Other Contributors

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-10-26	1.1	Appalabhaktula Hema	Updated logo and instruction
2021-10-31	1.0	Yan Luo	Created initial version of the lab