Apache Kafka and Python

Introduction

In this tutorial, you will build Python client applications which produce and consume messages from an Apache Kafka® cluster.

The lab will walk you through setting up a local Kafka cluster if you do not already have access to one.

Prerequisites

This guide assumes that you already have Python installed.

(If you're using Python 2.7, you'll also need to install Pip and VirtualEnv separately.)

Later in this lab, you will connect to an existing Kafka cluster.

Create Project

Create a new directory anywhere you'd like for this project:

```
mkdir kafka-python-getting-started && cd kafka-python-getting-started
```

Create and activate a Python virtual environment to give yourself a clean, isolated workspace:

```
virtualenv env

source env/bin/activate
```

Python 3.x

Install the Kafka library:

```
pip install confluent-kafka
```

Python 2.7

First install librdkafka

Then install the python libraries:

```
pip install confluent-kafka configparser
```

Kafka Setup

We are going to need a Kafka Cluster for our client application to operate with. Let's create a Kafka cluster now.

Paste the following file into a `docker-compose.yml` file:

```
---
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.3.0
    hostname: zookeeper
    container_name: zookeeper
    environment:
```

```
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-kafka:7.3.0
    container_name: broker
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://broker:29092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
```

Now start the Kafka broker with the new `docker compose` command.

```
docker compose up -d
```

Configuration

Paste the following configuration data into a file named `getting_started.ini`:

```
[default]
bootstrap.servers=localhost:9092

[consumer]
group.id=python_example_group_1

# 'auto.offset.reset=earliest' to start reading from the beginning of
# the topic if no committed offsets exist.
auto.offset.reset=earliest
```

Create Topic

Events in Kafka are organized and durably stored in named topics. Topics have parameters that determine the performance and durability guarantees of the events that flow through them.

Create a new topic, `purchases`, which we will use to produce and consume events.

We'll use the `kafka-topics` command located inside the local running Kafka broker:

```
docker compose exec broker \
  kafka-topics --create \
    --topic purchases \
    --bootstrap-server localhost:9092 \
```

```
    --replication-factor 1 \
    --partitions 1
```

Build Producer

Paste the following Python code into a file located at `producer.py`:

```
#!/usr/bin/env python

import sys
from random import choice
from argparse import ArgumentParser, FileType
from configparser import ConfigParser
from confluent_kafka import Producer

if __name__ == '__main__':
    # Parse the command line.
    parser = ArgumentParser()
    parser.add_argument('config_file', type=FileType('r'))
    args = parser.parse_args()

    # Parse the configuration.
    # See https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md
    config_parser = ConfigParser()
    config_parser.read_file(args.config_file)
    config = dict(config_parser['default'])

    # Create Producer instance
    producer = Producer(config)

    # Optional per-message delivery callback (triggered by poll() or flush())
    # when a message has been successfully delivered or permanently
    # failed delivery (after retries).
    def delivery_callback(err, msg):
        if err:
            print('ERROR: Message failed delivery: {}'.format(err))
        else:
            print("Produced event to topic {topic}: key = {key:12} value =
{value:12}".format(
                topic=msg.topic(), key=msg.key().decode('utf-8'),
value=msg.value().decode('utf-8')))

    # Produce data by selecting random values from these lists.
    topic = "purchases"
    user_ids = ['eabara', 'jsmith', 'sgarcia', 'jbernard', 'htanaka', 'awalther']
    products = ['book', 'alarm clock', 't-shirts', 'gift card', 'batteries']

    count = 0
    for _ in range(10):

        user_id = choice(user_ids)
        product = choice(products)
```

```
        producer.produce(topic, product, user_id, callback=delivery_callback)
        count += 1

    # Block until the messages are sent.
    producer.poll(10000)
    producer.flush()
```

Build Consumer

Paste the following Python code into a file located at `consumer.py`:

```
#!/usr/bin/env python

import sys
from argparse import ArgumentParser, FileType
from configparser import ConfigParser
from confluent_kafka import Consumer, OFFSET_BEGINNING

if __name__ == '__main__':
    # Parse the command line.
    parser = ArgumentParser()
    parser.add_argument('config_file', type=FileType('r'))
    parser.add_argument('--reset', action='store_true')
    args = parser.parse_args()

    # Parse the configuration.
    # See https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md
    config_parser = ConfigParser()
    config_parser.read_file(args.config_file)
    config = dict(config_parser['default'])
    config.update(config_parser['consumer'])

    # Create Consumer instance
    consumer = Consumer(config)

    # Set up a callback to handle the '--reset' flag.
    def reset_offset(consumer, partitions):
        if args.reset:
            for p in partitions:
                p.offset = OFFSET_BEGINNING
            consumer.assign(partitions)

    # Subscribe to topic
    topic = "purchases"
    consumer.subscribe([topic], on_assign=reset_offset)

    # Poll for new messages from Kafka and print them.
    try:
        while True:
            msg = consumer.poll(1.0)
            if msg is None:
                # Initial message consumption may take up to
```

```
                # `session.timeout.ms` for the consumer group to
                # rebalance and start consuming
                print("Waiting...")
            elif msg.error():
                print("ERROR: %s".format(msg.error()))
            else:
                # Extract the (optional) key and value, and print.

                print("Consumed event from topic {topic}: key = {key:12} value =
{value:12}".format(
                    topic=msg.topic(), key=msg.key().decode('utf-8'),
value=msg.value().decode('utf-8')))
    except KeyboardInterrupt:
        pass
    finally:
        # Leave group and commit final offsets
        consumer.close()
```

Produce Events

Make the producer script executable, and run it:

```
chmod u+x producer.py

./producer.py getting_started.ini
```

You should see output that resembles:

```
Produced event to topic purchases: key = jsmith value = batteries
Produced event to topic purchases: key = jsmith value = book
Produced event to topic purchases: key = jbernard value = book
Produced event to topic purchases: key = eabara value = alarm clock
Produced event to topic purchases: key = htanaka value = t-shirts
Produced event to topic purchases: key = jsmith value = book
Produced event to topic purchases: key = jbernard value = book
Produced event to topic purchases: key = awalther value = batteries
Produced event to topic purchases: key = eabara value = alarm clock
Produced event to topic purchases: key = htanaka value = batteries
```

Consume Events

Make the consumer script executable and run it:

```
chmod u+x consumer.py

./consumer.py getting_started.ini
```

You should see output that resembles:

```
Consumed event from topic purchases: key = sgarcia value = t-shirts
Consumed event from topic purchases: key = htanaka value = alarm clock
Consumed event from topic purchases: key = awalther value = book
Consumed event from topic purchases: key = sgarcia value = gift card
```

```
Consumed event from topic purchases: key = eabara value = t-shirts
Consumed event from topic purchases: key = eabara value = t-shirts
Consumed event from topic purchases: key = jsmith value = t-shirts
Consumed event from topic purchases: key = htanaka value = batteries
Consumed event from topic purchases: key = htanaka value = book
Consumed event from topic purchases: key = sgarcia value = book
Waiting...
Waiting...
Waiting...
```

The consumer will wait indefinitely for new events. You can kill the process off (with `Ctrl+C`), or experiment by starting a separate terminal window and re-running the producer.