# Aiven Kafka Quickstart with Python

## Introduction

Aiven is a managed service platform for consuming most common open-source databases and data services like Kafka, PostgreSQL, ElasticSearch, etc.

It takes away a burden of setting up and managing necessary infrastructure and in a few clicks can migrate your infrastructure from one region to another or even between various cloud providers like AWS, GCP, Azure and a few less known. With Aiven you can be setup and ready to go in a matter of a few minutes.

In this article I am going to show how to setup Kafka Service, develop python scripts for producer and consumer and also deploy service monitoring solution on top of Kafka.

So, where do you start?

## Setting up Aiven Account

First of all, create an account:

<https://console.aiven.io/signup.html>

and you will have $300 credit to play with it for some time without worrying about a need to pay. You do not even have to enter credit card as part of registration process.

## Setting up Kafka Service

Once you are logged into your newly created account, please to go “Services” in the menu on the left and click **“+ Create a new service”** button.

Choose Kafka service Version 3.0:

Graphical user interface, application

Description automatically generated

And scroll down. In here you can choose a cloud provider your infrastructure to be hosted at, region and payment plan. When you choose different options panel on the right immediately shows you cost projection based on your choices:

Graphical user interface, application

Description automatically generated

If you are looking just to play with the service for the lowest possible cost, I suggest choosing:

**Service Plan:** Startup-2

**Service Cloud Provider:** Digital Ocean

**Service Cloud Region:** <the one that is closer to your geo location>

For this demo I will go with:

Graphical user interface, text, application, email

Description automatically generated

Once you press create button you should get a new service in progress of creation (Status: Rebuilding):

Graphical user interface, application

Description automatically generated

After clicking on it, you will immediately have all connection credentials that you would need for your Python script:

Graphical user interface, text, application, email

Description automatically generated

Please create a folder on your disk (for instance “Aiven”) and download there “Access Key”, “Access Certificate” and “CA Certificate” provided. We would need them later.

I would save them with the default names suggested:

Graphical user interface, application

Description automatically generated

Provisioning of Kafka Service should take just a couple of minutes. Once status changes to “Running” we need to do the last bit - create a topic. I am going to use “json-data-stream”:

Graphical user interface, text, application, email

Description automatically generated

Once topic is created, it will show up in the list below:

Graphical user interface, application

Description automatically generated

## Writing Kafka Producer script

Next, let me create a new python script “aiven-kafka-producer.py”.

Of course, you would also need to have python on our machine and install the following lib

[kafka-python](https://pypi.org/project/kafka-python/).

On Mac installation of “kafka-python” is as simple as running:

**pip install kafka-python**

With that in place we can start developing Kafka Producer.

In my demo I want to simulate producing some IoT events with UUID as a key of the message and JSON object as a payload of the message. Each message would also have timestamp in ISO 8601 format. Messages will need to be produced with 1 sec delay.

To do all that, we would need to import a few packages:

from time import sleep

from json import dumps

import uuid

from random import randrange

from datetime import datetime

from kafka import KafkaProducer

The most important one is the last, which allows to build Kafka Producer. All the rest are needed just to implement the logic of my application.

Let’s create a producer. Given that our Kafka is secured with SSL and messages that we would be sending to it have to be JSON formatted, the code would look like:

producer = KafkaProducer(bootstrap\_servers="<aiven-kafka-host>:<port>",

security\_protocol="SSL",

ssl\_cafile="ca.cer",

ssl\_certfile="service.cert",

ssl\_keyfile="service.key",

key\_serializer=lambda x: dumps(x).encode('utf-8'),

value\_serializer=lambda x: dumps(x).encode('utf-8')

)

Please update **bootstrap\_servers** parameter with actual link to the Kafka Service in Aiven. In my case: “**kafka-2c20c13e-mfomenko-028c.aivencloud.com:22025**”.

Next, we need to generate a message and send it to Kafka Service. For that I will use the following code:

for e in range(5):

msgKey = str(uuid.uuid4())

msgData = {"motorId": "Fulton-A32", "eventTs": datetime.now().isoformat(), "sensorDataEvent": {"pressure": 20+randrange(10), "temperature": 40+randrange(5)}}

producer.send('json-data-stream', key=msgKey, value=msgData)

print("Sent a message with msgKey={} msgData={}".format(dumps(msgKey),dumps(msgData)))

sleep(1)

**msgKey** variable is used for random UUID v.4 value I generate using respective library.

**msgData** variable is used for JSON structure with IoT event attributes. I use **randrange()** function to make numbers fluctuate a bit between messages and **datetime.now().isoformat()** function to generate event timestamp in ISO 8601 format.

**json-data-stream** in my example is a Kafka topic name where we will be pushing data.

Between messages there will be one second delay thanks to **sleep(1)**. Every message is also printed to the console, so it was clear what was I generated.

All above we wrap into a loop with 5 iterations just to test on a small number of messages initially.

Finally, we finish the code with:

producer.flush()

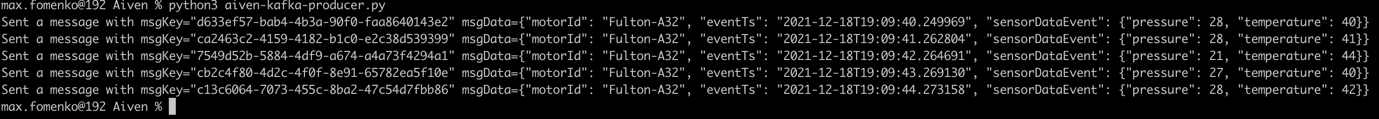
This make sures all messages have been sent and are not stuck in any Kafka’s buffer waiting for more messages before flushing content to Kafka Brokers.

This is the end of producer logic.

Now, all we need to do is to run the program:

**python3 aiven-kafka-producer.py**

and we should see something like below:



## Writing Kafka Consumer script

To check that messages landed in Kafka topic successfully, I would write a similar Kafka Consumer python script: “aiven-kafka-consumer.py”.

We again start with importing libs:

from json import loads

from json import dumps

from kafka import KafkaConsumer

the main one is the last. It allows to implement Kafka Consumer.

Other two are just needed for serialize/deserialise actions around JSON.

Majority of properties are similar to Kafka Producer:

consumer = KafkaConsumer(

"json-data-stream",

auto\_offset\_reset="earliest",

bootstrap\_servers="kafka-2c20c13e-mfomenko-028c.aivencloud.com:22025",

client\_id="consumer-1",

group\_id="consumer-group-1",

security\_protocol="SSL",

ssl\_cafile="ca.cer",

ssl\_certfile="service.cert",

ssl\_keyfile="service.key",

key\_deserializer=lambda x: loads(x.decode('utf-8')),

value\_deserializer=lambda x: loads(x.decode('utf-8'))

)

Only **client\_id** and **group\_id** are new and allow to configure customer’s ID and consumer’s Group if needed. For demo app those values do not really matter.

After creating consumer object, we are ready to read messages:

for \_ in range(2):

raw\_msgs = consumer.poll(timeout\_ms=1000)

for tp, msgs in raw\_msgs.items():

for msg in msgs:

print("Received a message with: msgKey={}, msgData={}".format(dumps(msg.key), dumps(msg.value)))

above code consumes messages and prints them to the console. We have to repeat **consumer.poll()** twice because first call will just assign partitions for our consumer without actually returning anything.

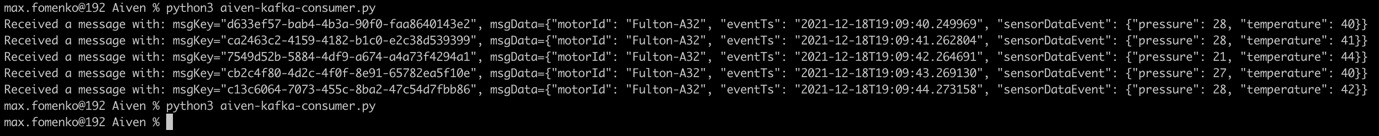
Lastly, as we do not want same messages to be fetched by the consumer again next time it polls, we need to commit a new offset to Kafka Broker:

consumer.commit()

after reading messages.

Now let’s try to run the program:

**python3 aiven-kafka-consumer.py**



As we can see, the messages that are printed are those that we previously pushed to Kafka topic. Furthermore, if we re-run consumer program without changing **group\_id** value, we no longer get these messages consumed as offsets have been successfully moved forward for this consumer.

## Observability (InfluxDB + Grafana)

Now when we have our Kafka Service up and running and even some tiny python application using it, it is important to monitor how Kafka Service performs in some easy to consume format – ideally a dashboard. This is also something Aiven can very easily provide by integrating “Kafka Service” with “InfluxDB Service” and “Grafana Service”.

First of all, let’s create two additional services we need:

Graphical user interface, application

Description automatically generated

For this demo I would go again for the cheapest option hosted at the same cloud provider and region as my Kafka Service:

Graphical user interface, text, application, email

Description automatically generated

While service is still provisioning:

Graphical user interface, text, application

Description automatically generated

You can already go to InfluxDB Service->Overview->Manage Integrations:

Graphical user interface, application

Description automatically generated

In the next dialog choose Kafka Service you want to collect metric from:

Graphical user interface, text, application

Description automatically generated

And press “Enable”. That’s it! Once integration is activated, you should be seeing it in Overview tab of the service:

Graphical user interface, text, application

Description automatically generated

Metrics from Kafka are now pushed to the time series DB.

Next step is to provision Grafana Service to visualize metrics data on a Dashboard:

Graphical user interface, application

Description automatically generated

Similarly, to previous services, I would again go for the cheapest option in the same Cloud Provider and Region:

Graphical user interface, text, application

Description automatically generated

While service is still provisioning:

Graphical user interface, text, application, chat or text message

Description automatically generated

You can already go to Grafana Service->Overview->Manage Integrations:

Graphical user interface, application

Description automatically generated

In the next dialog choose Kafka Service you want to collect metric from:

Graphical user interface, text, application, email

Description automatically generated

And press “Enable”. That’s it! Once integration is activated, you should be seeing it in Overview tab of the service:

Graphical user interface, text, application

Description automatically generated

Let’s now connect to Grafana instance and look at already pre-configured Dashboard.

The details we need for connecting are available in the Grafana Service->Overview:

Graphical user interface, application

Description automatically generated

Go by service URI and use user/password that you have:

Graphical user interface, application

Description automatically generated

Once you are there, go to search Dashboards:

Graphical user interface, application

Description automatically generated

And you will see pre-configured one for Kafka Service:

Graphical user interface, application

Description automatically generated

Just open it to see all available metrics:

Graphical user interface

Description automatically generated with low confidence

Among other metrics there are two that we are going to use to see that metrics from Kafka Service are indeed reflected in the dashboard. As you can see, currently Bytes in/out are all zero:

A screenshot of a computer

Description automatically generated with medium confidence

I am going to update “aiven-kafka-producer.py” to push 100 messages into Kafka topic and will then consume them using “aiven-kafka-consumer.py”. After doing that, and refreshing Grafana Dashboard:

Graphical user interface, website

Description automatically generated

We indeed can see fresh metrics appearing in the dashboard:

Graphical user interface, application

Description automatically generated

This is the end of my quick demo.

## Summary

In this demo we have:

1. Provisioned Aiven Kafka Service
2. Provisioned Aiven InfluxDB Service
3. Provisioned Aiven Grafana Service
4. Connected to Kafka Service from a Python scripts (producer and consumer) to push and pull some test messages
5. Integrated Aiven Kafka Service with Aiven InfluxDB Service to push metrics to a time series DB.
6. Integrated Aiven Grafana Service with Aiven Influx DB Service to visualize Kafka Service metrics in a dashboard that was pre-built for me by Aiven.

All together it allowed us dramatically bootstrap Kafka application development. We have spent only about one hour to setup a comprehensive deployment.