Aiven Kafka Quickstart with Python

by Yossi Drori | on 05 July 2022 | Technical How-To

In this blog post, I show you how to configure Kafka pipeline by using **Aiven** services and Python code

Inorder to check the pipeline, I will show how to build a client producer and to stream a stock market mockup data using Python

On the last step I will guide on how to Monitor the pipeline

Solution overview

The following architecture diagram-Figure 1-presents an overview of the solution.

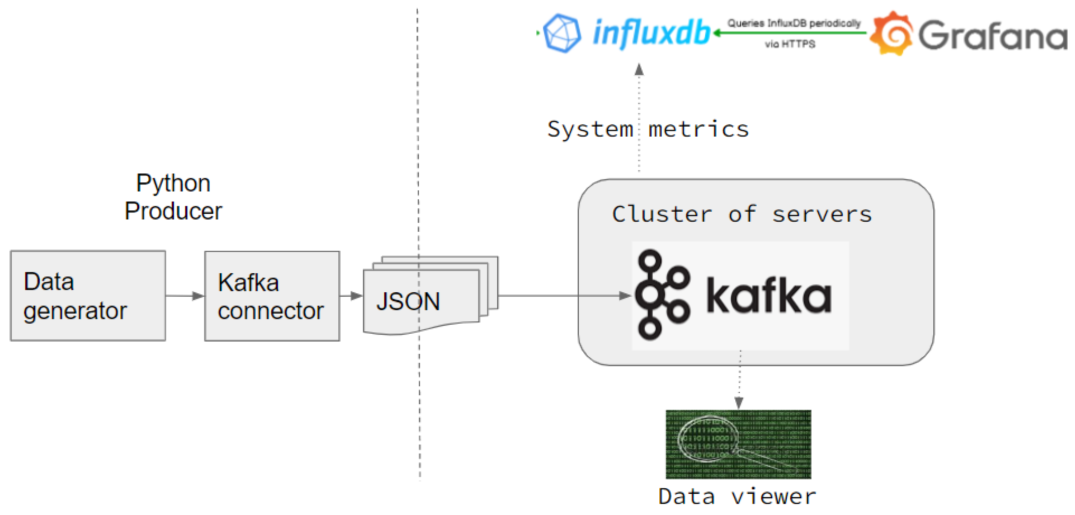


Figure 1: Solution architecture diagram

- In this example we can see 2 main components:

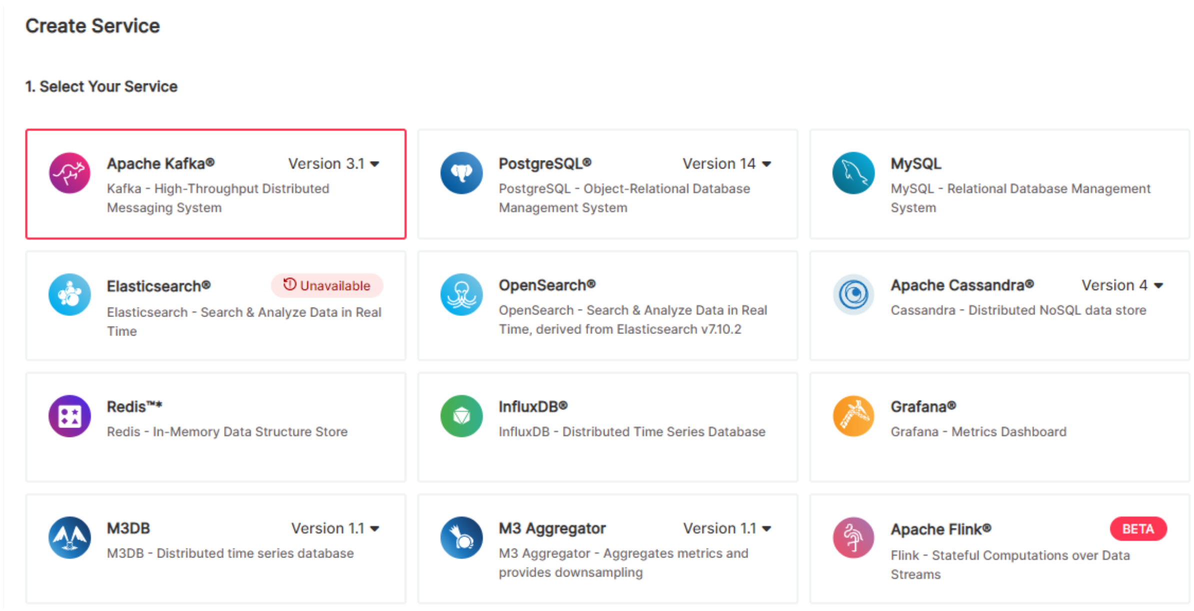
A. **The producer** - is responsible for generating data and sending it to the Kafka cluster
B. **The Kafka cluster** - is responsible for receiving the data

Prerequisites

- Registering to Aiven
- Python environment
  - Installing faker
  - Installing kafka-python
- Clone the github project: Python Fake Data Producer for Apache Kafka

Implement the solution

Setting up a Aiven Kafka service



Create Service

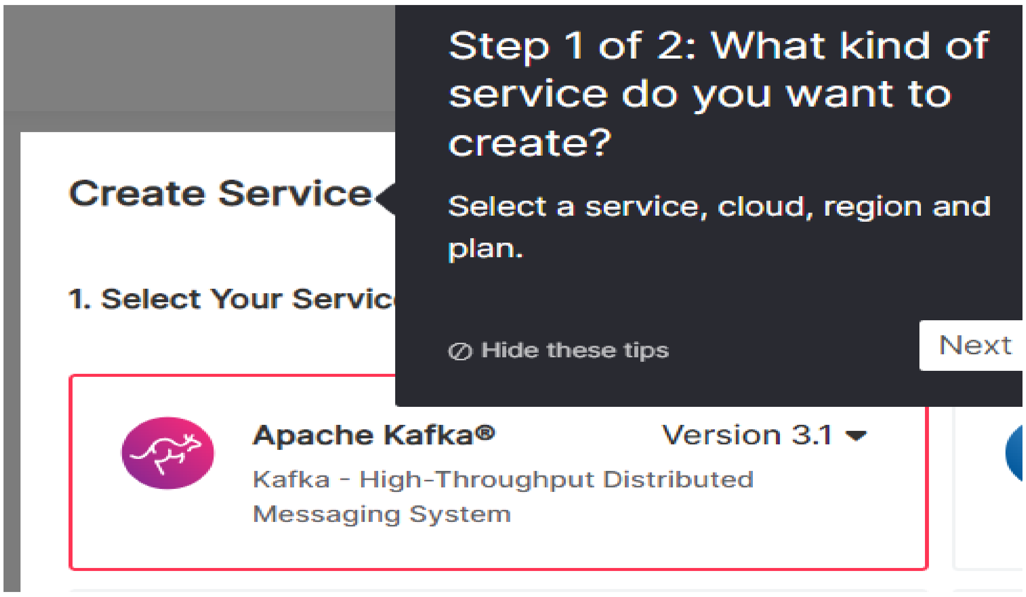
1. Select Your Service

Apache Kafka® Version 3.1 — Kafka - High-Throughput Distributed Messaging System	PostgreSQL® Version 14 — PostgreSQL - Object-Relational Database Management System	MySQL — MySQL - Relational Database Management System
Elasticsearch® Unavailable — Elasticsearch - Search & Analyze Data in Real Time	OpenSearch® — OpenSearch - Search & Analyze Data in Real Time, derived from Elasticsearch v7.10.2	Apache Cassandra® Version 4 — Cassandra - Distributed NoSQL data store
Redis™* — Redis - In-Memory Data Structure Store	InfluxDB® — InfluxDB - Distributed Time Series Database	Grafana® — Grafana - Metrics Dashboard
M3DB Version 1.1 — M3DB - Distributed time series database	M3 Aggregator Version 1.1 — M3 Aggregator - Aggregates metrics and provides downsampling	Apache Flink® BETA — Flink - Stateful Computations over Data Streams

Figure 2: Choose the relevant service

From the list for all services we will first choose "Apache Kafka"

Kafka configuration



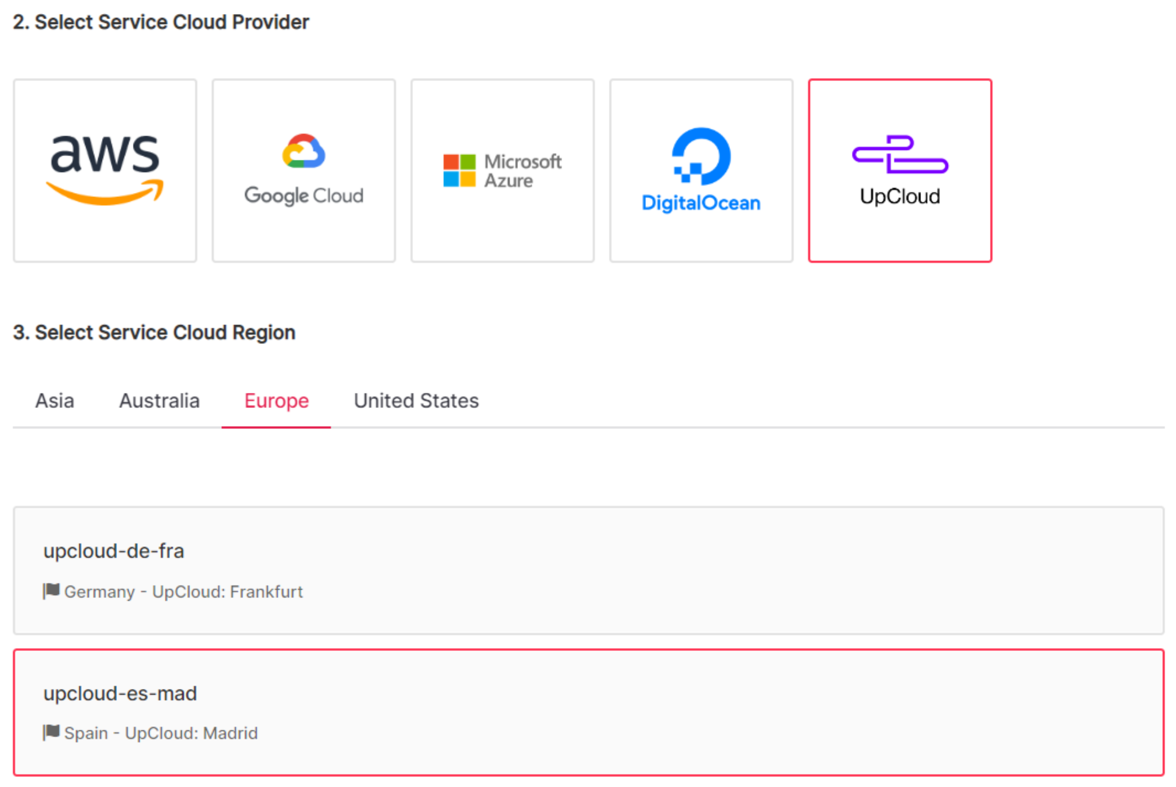
Step 1 of 2: What kind of service do you want to create?

Select a service, cloud, region and plan.

Figure 3: Kafka configuration tool tips

After selecting the desired service (Kafka in our case)
Most informative tool tips will guide you through on how to set it the best for your use case!

Selecting: Cloud service + Region


2. Select Service Cloud Provider

aws | Google Cloud | Microsoft Azure | DigitalOcean | UpCloud

3. Select Service Cloud Region

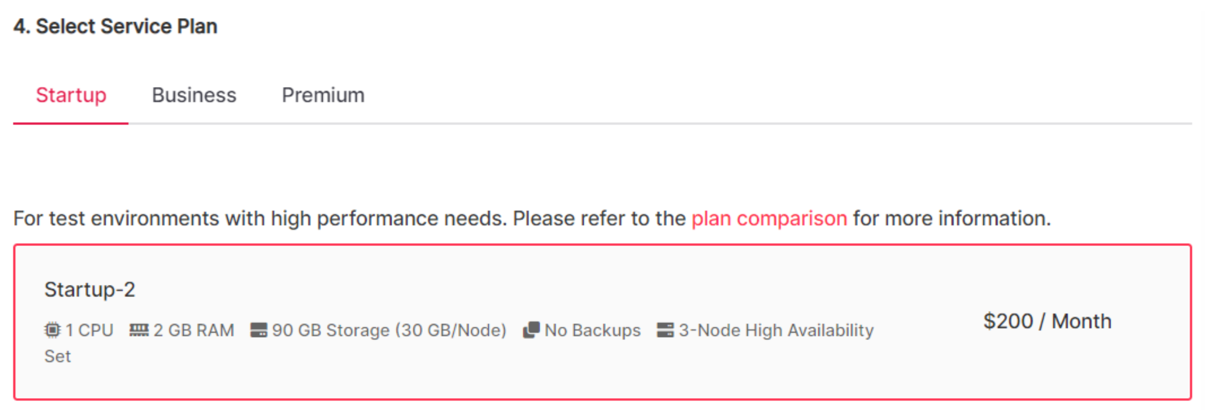
Asia | Australia | Europe | United States

upcloud-de-fra
Germany - UpCloud: Frankfurt

upcloud-es-mad
Spain - UpCloud: Madrid

Figure 4: select Cloud service + Region

Flexible service plans


4. Select Service Plan

Startup | Business | Premium

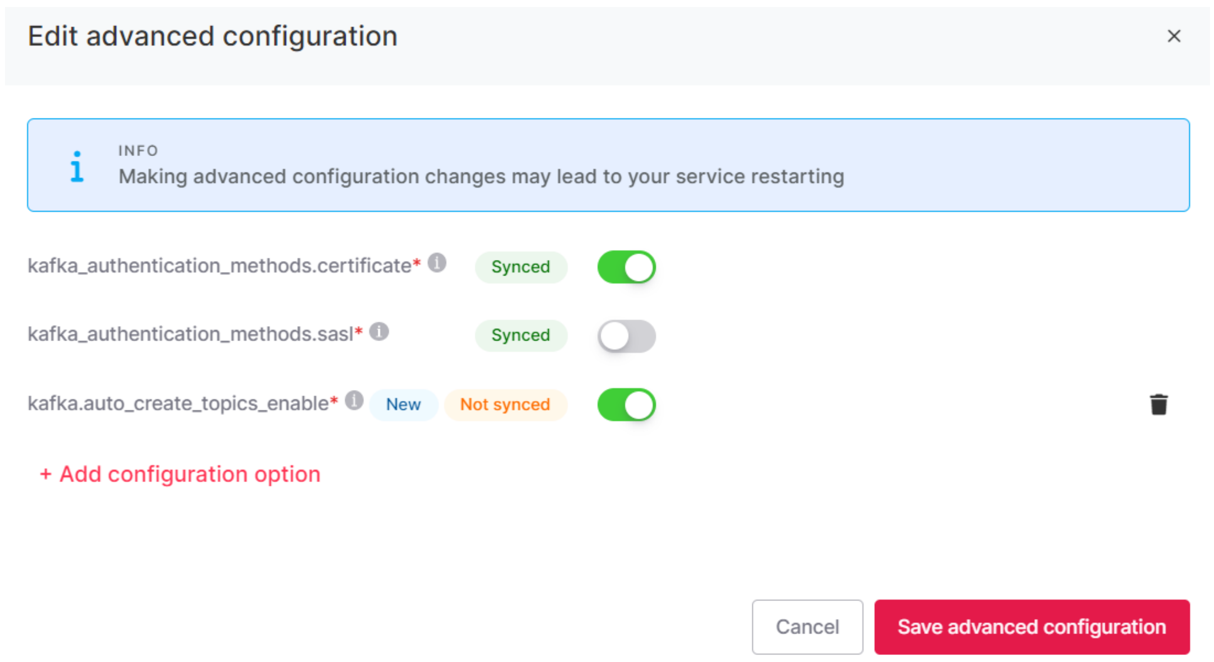
For test environments with high performance needs. Please refer to the plan comparison for more information.

Startup-2
1 CPU 2 GB RAM 90 GB Storage (30 GB/Node) No Backups 3-Node High Availability Set
$200 / Month

Figure 5: select a plan

In my case I’m using “Startup” since it’s just a functionality test which requires light resources For “heavy lifting” you will consider “Business” / “Premium”.

Advanced configurations

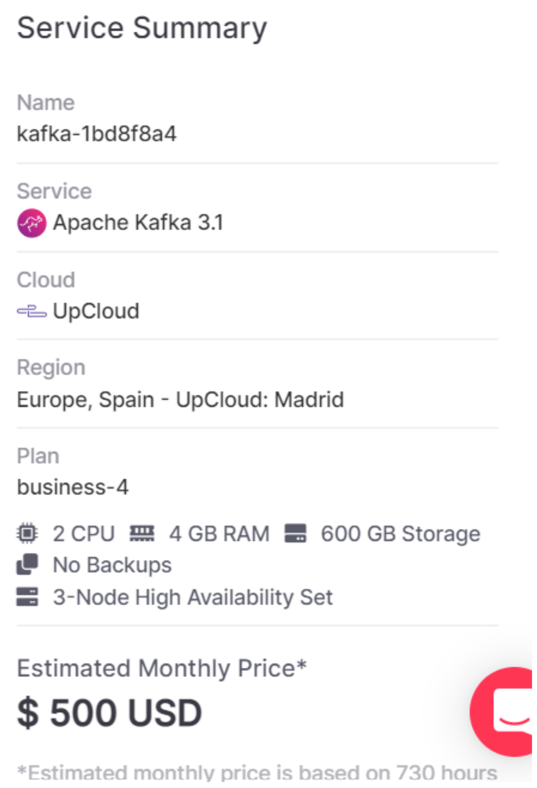

Edit advanced configuration

INFO
Making advanced configuration changes may lead to your service restarting

kafka_authentication_methods.certificate*		Synced	On
kafka_authentication_methods.sasl*		Synced	Off
kafka.auto_create_topics_enable*	New	Not synced	On

Figure 6: Advanced configurations

Service setup summary


Service Summary

Name
kafka-1bd8f8a4

Service
Apache Kafka 3.1

Cloud
UpCloud

Region
Europe, Spain - UpCloud: Madrid

Plan
business-4

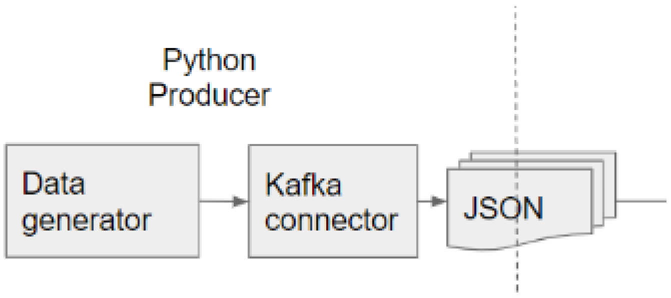
2 CPU 4 GB RAM 600 GB Storage
No Backups
3-Node High Availability Set

Estimated Monthly Price*
$ 500 USD

*Estimated monthly price is based on 730 hours

Figure 6: Service setup summary

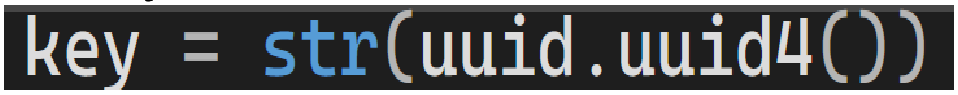
Data generator and producer



See - Figure 1

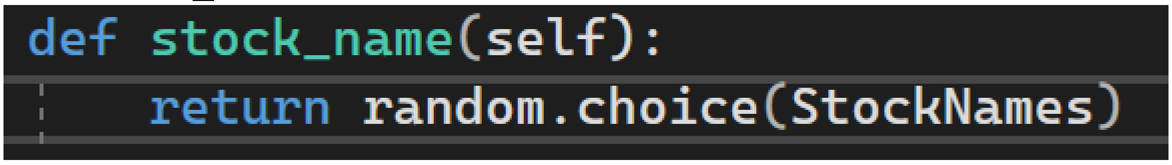
Data generator
Inorder to generate the mockup stock tickets data I have used [Faker Python lib]
In my example I'm generating stock tickets mockup data
The file [stockproducer.py] is implementing the code for the mockup data:

- **Key**



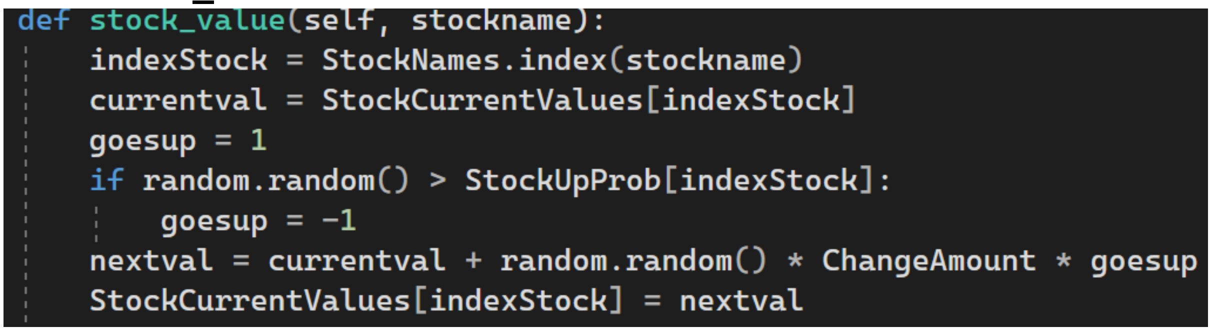
```
key = str(uuid.uuid4())
```

- **stock_name**



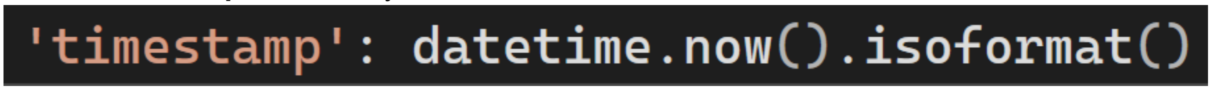
```
def stock_name(self):
    return random.choice(StockNames)
```

- **stock_value**



```
def stock_value(self, stockname):
    indexStock = StockNames.index(stockname)
    currentval = StockCurrentValues[indexStock]
    goesup = 1
    if random.random() > StockUpProb[indexStock]:
        goesup = -1
    nextval = currentval + random.random() * ChangeAmount * goesup
    StockCurrentValues[indexStock] = nextval
```

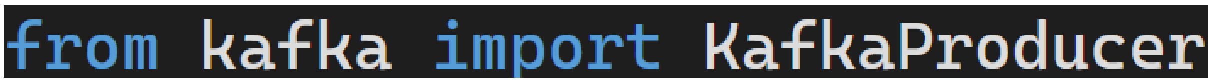
- **timestamp -** Is actually the current time in ISO 8601 format



```
'timestamp': datetime.now().isoformat()
```

An example for generated event:
__key__: cd4fab54-6f6e-4f30-ac93-7e1941079357
__message__:
{
'stock_name': 'Indiana Jeans'
,'stock_value': 26.568693883965526
,'timestamp': '2022-07-05T23:23:58.236269'
}

Sending the data into Kafka cluster
On the main.py I've implemented the interaction with the Kafka cluster by using the "KafkaProducer"



```
from kafka import KafkaProducer
```

To run the Python producer

python main.py
--cert-folder *Stock*
The folder with the credential files

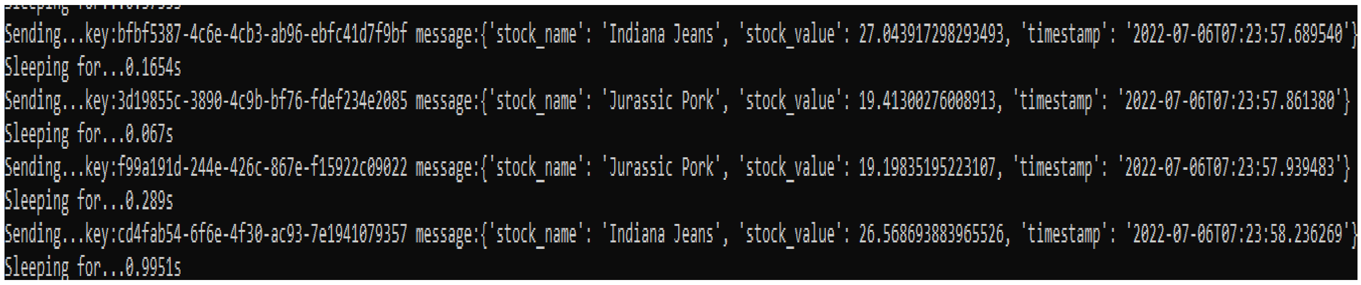
--host *kafka-stock-tickets-drori-a69e.aivencloud.com*
The Kafka service URL

--port *28003*

--topic-name *stock-tickets*
The Kafka topic name (it will be created dynamically on the Kafka side)

--nr-messages *1000*
Number of messages to generate

--max-waiting-time *1*
Maximum time between messages



```
Sending...key:bfbf5387-4c6e-4cb3-ab96-ebfc41d7f9bf message:{'stock_name': 'Indiana Jeans', 'stock_value': 27.043917298293493, 'timestamp': '2022-07-06T07:23:57.689540'}
Sleeping for...0.1654s
Sending...key:3d19855c-3890-4c9b-bf76-fdef234e2085 message:{'stock_name': 'Jurassic Pork', 'stock_value': 19.41300276008913, 'timestamp': '2022-07-06T07:23:57.861380'}
Sleeping for...0.067s
Sending...key:f99a191d-244e-426c-867e-f15922c09022 message:{'stock_name': 'Jurassic Pork', 'stock_value': 19.19835195223107, 'timestamp': '2022-07-06T07:23:57.939483'}
Sleeping for...0.289s
Sending...key:cd4fab54-6f6e-4f30-ac93-7e1941079357 message:{'stock_name': 'Indiana Jeans', 'stock_value': 26.568693883965526, 'timestamp': '2022-07-06T07:23:58.236269'}
Sleeping for...0.9951s
```

Observability and Monitoring

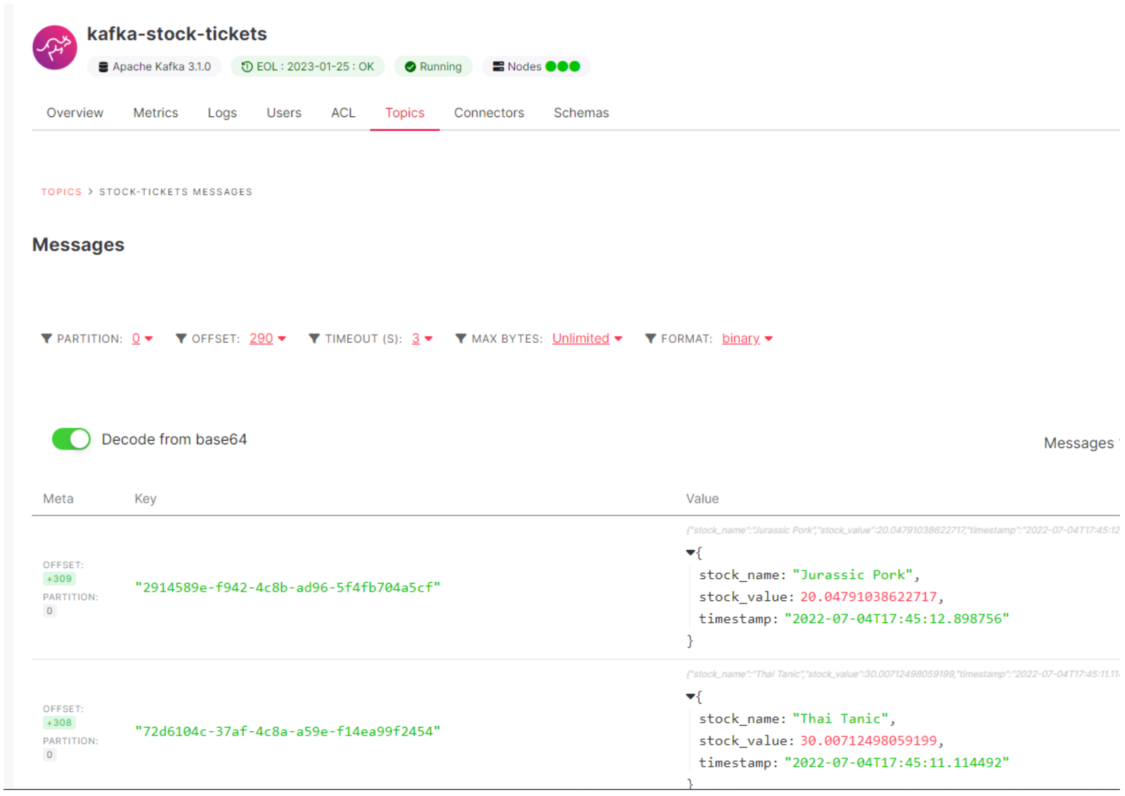
Browsing to the kafka topic.
The generated data includes stock tickets values

When browsing to the topic we can see the JSON steamed data

We can see 2 sections: “**key**” and “**value**”

Key: UUID

VALUE: {“stock_name”, “stock_value”, “timestamp”}


kafka-stock-tickets

Apache Kafka 3.1.0 | EOL : 2023-01-25 : OK | Running | Nodes ●●●

Overview | Metrics | Logs | Users | ACL | Topics | Connectors | Schemas

TOPICS > STOCK-TICKETS MESSAGES

Messages

PARTITION: 0 | OFFSET: 290 | TIMEOUT (S): 3 | MAX BYTES: Unlimited | FORMAT: binary

Decode from base64

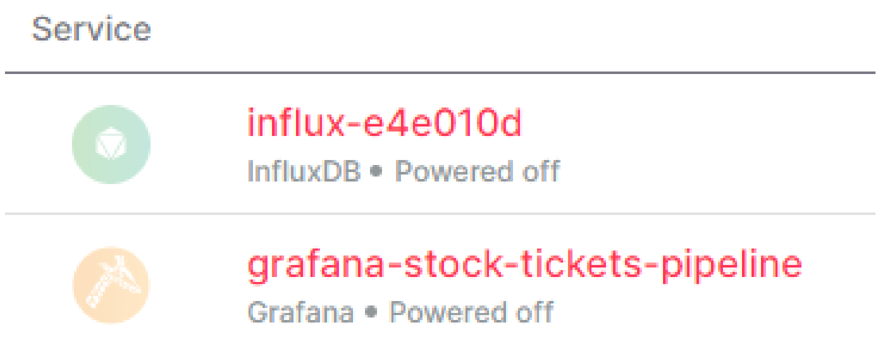
Messages

Meta	Key	Value
OFFSET: +309 PARTITION: 0	"2914589e-f942-4c8b-ad96-5f4fb704a5cf"	{"stock_name":"Jurassic Pork","stock_value":20.04791038622717,"timestamp":"2022-07-04T17:45:12 { stock_name: "Jurassic Pork", stock_value: 20.04791038622717, timestamp: "2022-07-04T17:45:12.898756" }
OFFSET: +308 PARTITION: 0	"72d6104c-37af-4c8a-a59e-f14ea99f2454"	{"stock_name":"Thai Tanic","stock_value":30.00712498059199,"timestamp":"2022-07-04T17:45:11.11 { stock_name: "Thai Tanic", stock_value: 30.00712498059199, timestamp: "2022-07-04T17:45:11.114492" }

Figure 7: Browsing to the data in the pipeline

This view is available from: Kafka service view => Topics => Topic => Fetch Messages

Monitoring system matrix


Service

influx-e4e010d
InfluxDB • Powered off

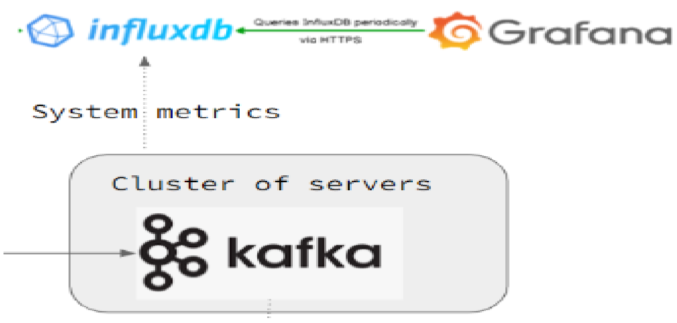
grafana-stock-tickets-pipeline
Grafana • Powered off

Figure 8: Adding services to enable system monitoring

Inorder to have a system monitoring I've created 2 additional services:

1. **Influx-db -** A database that connects to the Kafka cluster and collects the system matrices

2. **Grafana -** A visualizer that is connected to the **Influx-db** and can present all the system matrices in a dashboard



See - Figure 1

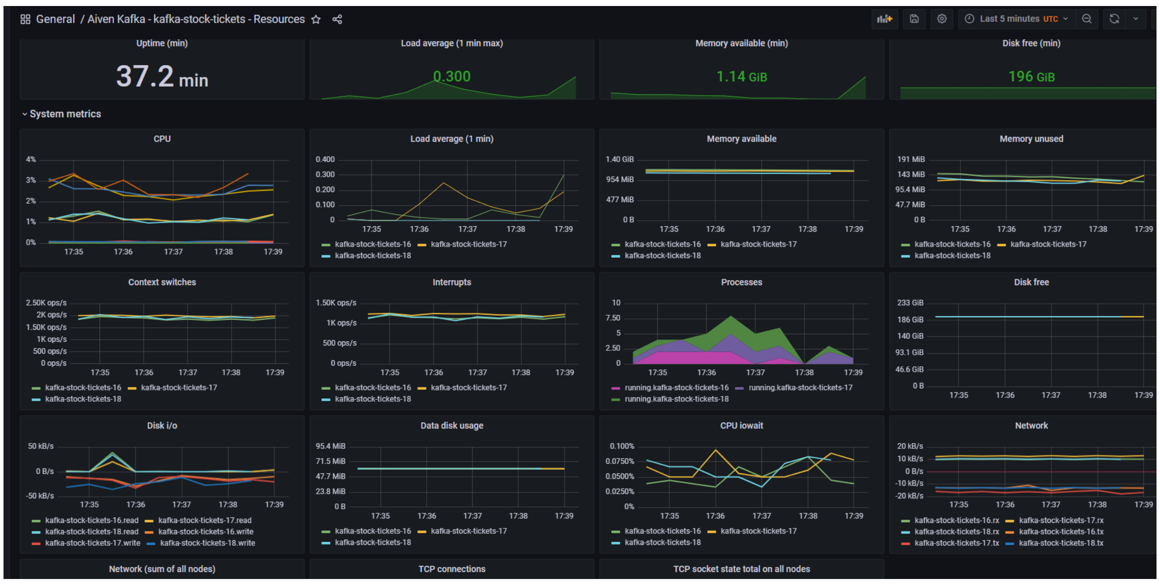


Figure 8: Using Grafana to monitor the system matrices

Wrapping up

As I have shown in this document, by following a simple set of instructions you can easily set up an event driven pipeline.

Working with Aiven console is very simple and have a lot of tool tips to guide you through

Since the service is a “Pay as you go” don’t forget that you can stop the services when you don’t need it up and running.

For any question or feedback please contact me or our support team

Email: drori.yossi@gmail.com