**Building a Data Mesh using Aiven**

**Part *I) Introduction*:**

The Data Mesh is the future. At the heart of the Data Mesh is Apache Kafka. Currently 70% of Fortune 500 Companies are utilizing Apache Kafka. While Kafka can be extremely powerful, the setup and orchestration can be complex and mind boggling. With Aiven, we take care of the infrastructure and maintenance for you so that way organizations can focus on software development.

In this tutorial, I will walk you through on how you can build your own Data Mesh using the Aiven Console. From setting up Kafka, generating sample data to integrating monitoring and observability, you will learn the offerings Aiven can provide.

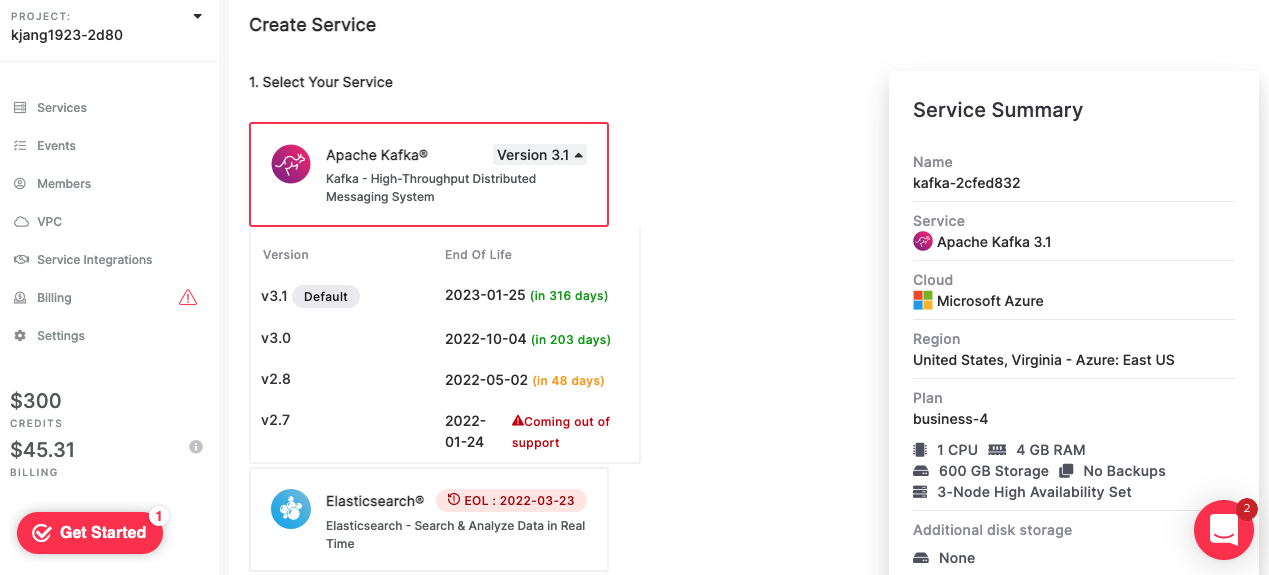
Feel free to clone the Readme to the tutorial here.

https://github.com/aivenkafka/EventStreamingApplication

WARNING: This blog is not a substitute for the official Kafka Documentation which can be found [here](https://kafka.apache.org/documentation/). Kafka itself can be complex and this basic demo is used to illustrate an event driven architecture using the Aiven Console.

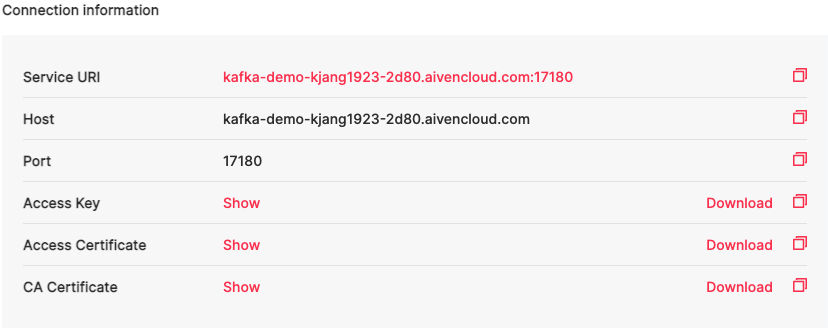
1) Setting up Kafka Server with SSL

A. Go to the Aiven [homepage](https://console.aiven.io/). In the homepage go to the top right and click on "Create a new Service". When selecting your service, choose Apache Kafka. With Aiven you have the option to pick the version of Kafka best suited for your needs.



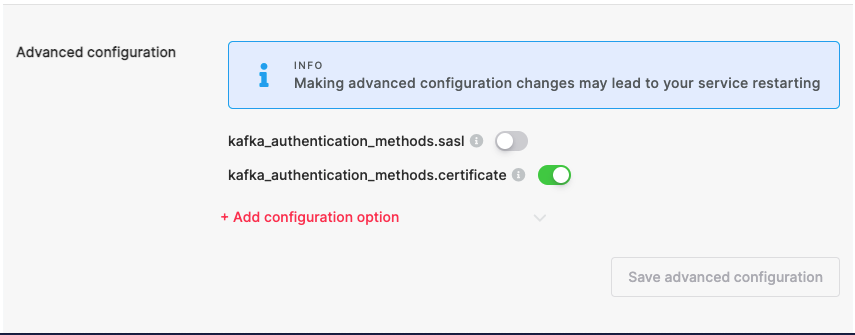
As you select your settings, select the Cloud Provider, Region, Plan and Service Name that best suites you. It is important to note that your Provider Service Name cannot be changed after.

B. Once your Kafka Server is created observe the connection information.



Your Kafka Server is the Service URI. Since we will also setup SSL for our Kafka server, please download the "Access Key", "Access Certificate" and "CA Certificate"

**Tip**: Based upon the server settings, we will only show SSL for simplicity of the demo but SASL authorization is also available to complement SSL. Additionally, the Aiven Console can also restrict IPs but by default, it is open to all IP Addresses with a valid SSL Certificate.

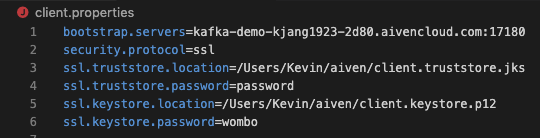


C. Later on in the demo, we will create a Java Producer application that will send data to Kafka. You will need to use the certificates and keys we downloaded earlier to generate a truststore and a keystore. The files we downloaded are the "ca.pem", "service.cert" and the "service.key".

D. Full instructions are found [here](https://github.com/aivenkafka/EventStreamingApplication/blob/master/SSL_Setup.sh).

Warning: The passwords to the truststore and keystore are very simple. This should not be in a Production environment.

2. Since our Kafka Producer is in Java, we will create a properties file to account for the details.



**Part II)** **Begin the Kafka Producer**

Introduction: The business requirement for the Kafka Producer include the following

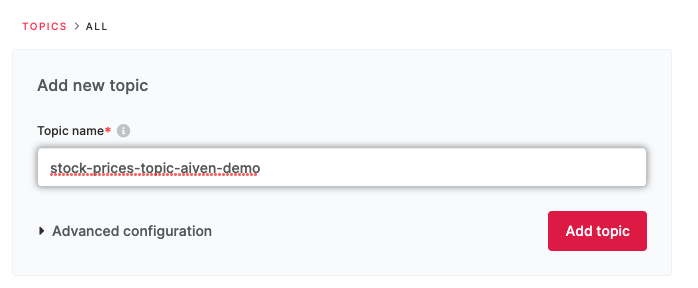
1. The Kafka Producer will now send data to Kafka. Please clone/download the repo found in this url found in [here](https://github.com/aivenkafka/EventStreamingApplication). Using your favorite code editor (IntelliJ or Eclipse), ensure that Maven is installed in [here](https://maven.apache.org/install.html). For this demo, we will be using IntelliJ.

A. Move your "client.properties", "client.truststore.jks" and "client.keystore.p12" file into the "resources" folder.

B. Once this is moved, let's create the Kafka topic. The Kafka topic will be called "*stock-prices-topic-aiven-demo*"

Note: Our business requirements include the Key to have a unique

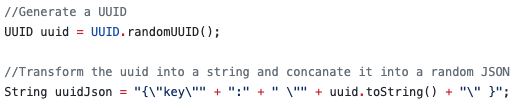
C. You can create the topic. Click the "Add Topic" button.



Tip: We did not need to hardcode or manually place our Kafka Server Settings in the Java Code since all of it was done in the "client.properties" file. Any setting relating to a Kafka Client can be added in the properties file. Documentation of Kafka Producer or Consumer settings can be found in the Kafka Documentation. [Producer Settings](https://kafka.apache.org/documentation/#producerconfigs) and [Consumer Settings](https://kafka.apache.org/documentation/#consumerconfigs).

2. Before starting the demo and clicking the run button, let's dive in deeper on the code found [here](https://github.com/aivenkafka/EventStreamingApplication/blob/master/src/main/java/KafkaClients/KafkaProducerApplication.java).

Requirement 1: The Key needs to generate a UUID. The message needs to be valid JSON



*This can be found on lines 73 through 76.*

Note: From lines 59 through lines 94, a function generating the Producer Record will be created. This will be invoked in line 48 when sending the Kafka message in the valid JSON Requirement 2: The value needs to be valid JSON. Lines 60-61 of the code can accomplish this.

Requirement 2: The message needs to contain a timestamp in ISO 8601 a random event. Our event is a Stock Bid Price of 4 Companies. These symbols will be random.

../Screen%20Shot%202022-03-13%20at%209.51.23%20PM.png



*Below, a producer record is generated with the associated name of the topic. The key is also generated as well. Time to run the code. Click the Green Triangle button.*

Requirement 3: The message needs to be seen in the Aiven Console.

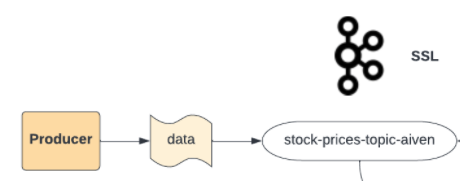


***{"key":"daa3c5e0-d3f3-4d58-bfcc-d8586777eeb2"}***

*{"Symbol":"FB","AskingPrice":223,"TimeOfBid":"2022-03-14T01:34:34.557Z"}*

Tip: These messages can be validated in "jsonlint.com"

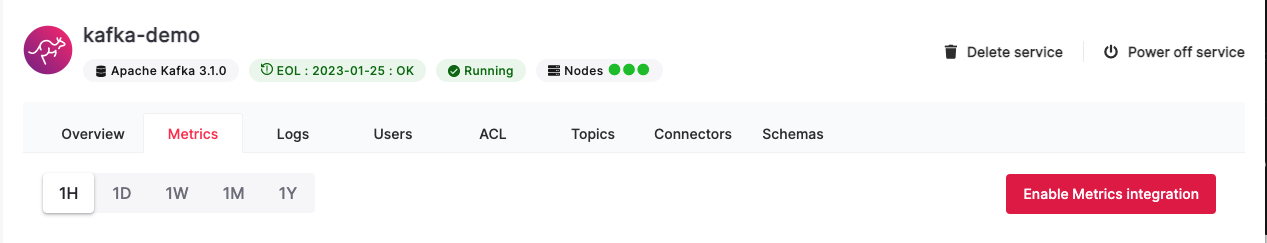
So far we created a Kafka Producer. The Producer is currently sending data to secured Kafka server using SSL/TLS Authentication. In the next part, we will create a Kafka Consumer that will send data to InfluxDB and then Grafana.



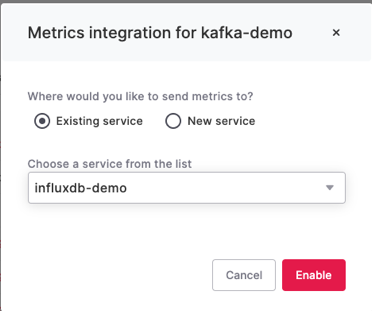
Part III) Show Observability using Influx DB & Grafana

Introduction: There are many stats to monitor in Kafka. Fortunately, by sending the metrics to InfluxDB and then visualized in Grafana, Kafka metrics can be monitored for you. Go to the Aiven Console and create a new InfluxDB Service and a Grafana as well. The name of the InfluxDB Service is called "influx-demo" and the name of the Grafana Service is called "grafana-188d50d5". Wait for these services to be created.

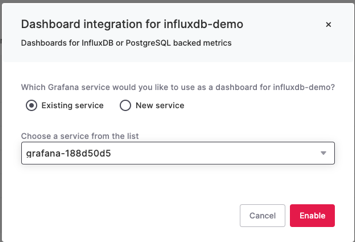
1. Start off by clicking "Enable Metrics Integration"



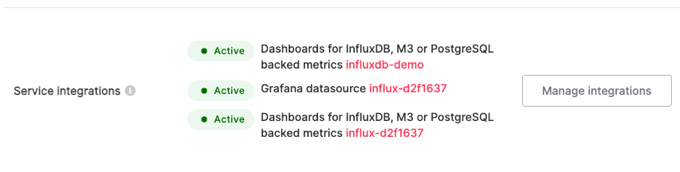
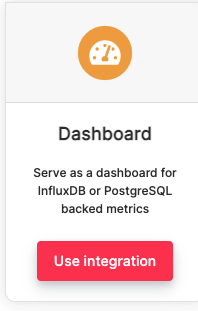
2. Choose the InfluxDB in the Service List.



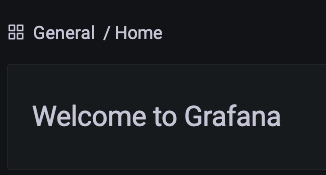
Once this is set, if you go view your InfluxDB Service, you will be prompted to the Dashboard integration. Choose the Grafana Service. Click "Enable".



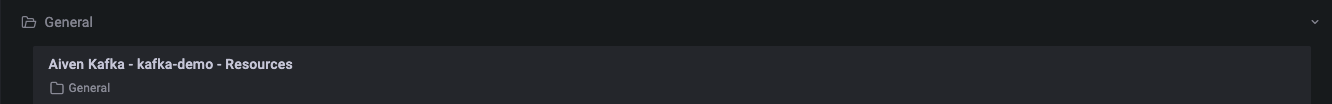
If you are not prompted the window, no worries, just go back to your Grafana Service and click "Manage Integrations". Go to the Dashboard and click "Use Integration"

 => 

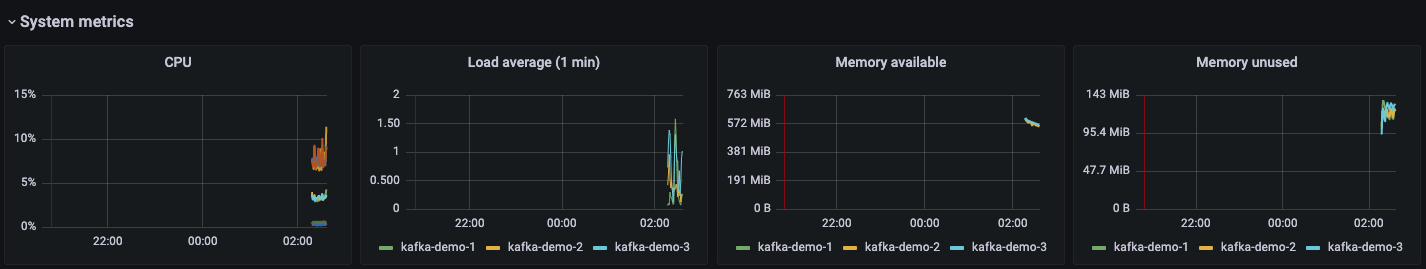
4. Logon to your Grafana Instance. The link will be displayed in the Grafana Console. When you login to the Grafana Console, go to the "General/Home Menu". Click on the tab.



A. When you go in, there will be a folder with a Dashboard pre-built for you. These metrics will have all of the metrics of your Kafka Server.



B. Once you click on that, you can now see the metrics.



Introduction: Producers send data to Kafka while Consumers read data from Kafka. When Consumers read data, they consume it from a Kafka topic and send the data to a target system. In this demo we will create a Kafka Consumer using Telegraf. The Consumer will read the data and send the data into Influx DB.

A. Telegraf can be installed [here](https://github.com/influxdata/telegraf).

B. InfluxDB is offered on Aiven which can be setup through our Console.

C. Information on InfluxDB can be found [here](https://aiven.io/influxdb).

Verdict: So far we have created a Kafka Producer that sends data to Kafka. Data is in a valid JSON that shows stock prices. Upon sending the data successfully to Kafka, we also enabled observability. Data is sent to InfluxDB and visualized with Grafana. With Aiven the setup is simple so organizations can focus on developing applications while Aiven maintains the infrastructure. Teamwork makes the dream work! Additional features like Telegraf are not in this blog but are part of the ReadMe. Kafka Streams and Kafka Connect are also found in the Git Repo as well.