

Aiven Kafka Quickstart With Spring Boot



In this exercise we will create a webhook (REST API) to publish stock price updates as Kafka Messages. This is to simulate real world scenarios where the webhook can be triggered by a financial data provider such as Polygon.io everytime there is an update to the share price.

Publishing these data to Kafka enables downstream services to consume and react to the price change event and reporting in near real time at scale.

Source code for this exercise available at: https://github.com/e17711/kafka_aiven.git

1. Create Aiven Kafka Service

Create a new Kafka service in the Aiven dashboard, and wait for the status to be up and running.

Service	Nodes
 emil-kafka-exercise Apache Kafka • Running	

For this tutorial we will be using SSL authentication (default). Please follow the steps to generate Java key-store and trust-store for Aiven Kafka in : <https://developer.aiven.io/docs/products/kafka/howto/keystore-truststore.html>

The next step will be to create a topic called 'stockTicker' in Aiven-Kafka. Go to Kafka Service > Topics, enter topic name, and click "Add Topic" button

Overview

Metrics

Logs

Users

ACL

Topics

Connectors

Schemas

TOPICS / ALL

Add new topic

Topic name* ⓘ

stockTicker

► Advanced configuration

Add topic

Ensure "Apache Kafka REST API" is enabled under overview, to allow us to see messages in stockTicker topic on Aiven Kafka Dashboard. The switch is in the Kafka > Overview page.

Apache Kafka REST API (Karapace)	HTTP REST based interface to the Apache Kafka cluster. See the  Karapace project on Github for documentation.	
-------------------------------------	--	---

2. Initiate Spring Boot Project

Create a Spring Boot Project in your Favorite IDE, example VS Code or Spring Tool Suite. Spring initializr can be used to bootstrap the project creation (<https://start.spring.io/>).

Ensure the following dependency is selected

- **spring-boot-starter-web**
- **spring-kafka**

Check the generated pom.xml file, it should contains two dependencies like below screenshot

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

3. Update application.properties

Spring-kafka use the following properties to connect and define the serializer class for Kafka producer:

- **spring.kafka.bootstrap-servers** : this is your kafka broker address and port
- **spring.kafka.producer.key-serializer** : the Serializer class to use for Key of the kafka message.
- **spring.kafka.producer.value-serializer** : the Serializer class to use for the Value of the kafka message.
- **spring.kafka.ssl.trust-store-location** : location of trust store file (.jks)
- **spring.kafka.ssl.trust-store-password** : password to access the trust store
- **spring.kafka.ssl.key-store-location** : location of the key store (.p12)
- **spring.kafka.ssl.key-store-password** : password to access the key store

In this exercise Spring kafka JsonSerializer will be used as the key and value serializer. This option will ensure that the producer only publishes message keys and values in JSON compatible format.

Below is the screenshot of the spring boot application.properties file:

```
#Kafka server address and port
spring.kafka.bootstrap-servers=emil-kafka-exercise-sulistya-e019.aivencloud.com:25950

#key will be UUID in String
#spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.key-serializer= org.springframework.kafka.support.serializer.JsonSerializer

#to serialize java object as JSON
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer

#Kafka SSL config
spring.kafka.ssl.trust-store-location=file:///home/emil/Downloads/aiven-kafka/client.truststore.jks
spring.kafka.ssl.trust-store-password=abcd1234
spring.kafka.ssl.key-store-location=file:///home/emil/Downloads/aiven-kafka/client.keystore.p12
spring.kafka.ssl.key-store-password=abcd1234
spring.kafka.properties.security.protocol=SSL
```

4. Create model class in Java

To simulate stock price update events, we will create a Ticker class with the following parameters: Symbol, Buy Price, Sell Price, and Timestamp. Refer to the screenshot below:

```
// Model class to represent a stock ticker buy and sell price at a point of time
public class Ticker {

    private String symbol;
    private String buy;
    private String sell;
    private String timestamp;
```

Getter/Setter will be created for each parameter.

5. Create Kafka producer using Spring Kafka Template

Spring-kafka provides KafkaTemplate for conveniently creating Producer and Consumer from the configuration in application.properties file with minimal coding.

Code below use KafkaTemplate to initialize a Kafka producer from application.properties:

```
@Service
public class KafkaProducer {
    //We are just going to hardcode Kafka Topic to be used to publish message for this demo
    private static final String kafkaTopic="stockTicker";

    /*Initiate Kafka Template with spring-kafka dependency injection based on application.properties */
    @Autowired
    private KafkaTemplate<String, Ticker> kafkaTemplate;

    /*Convenient method to publish a message to kafka broker*/
    public void sendMessage(Ticker ticker) {

        // Using Random UUID as key for the Kafka Event
        String key=UUID.randomUUID().toString();
        this.kafkaTemplate.send(kafkaTopic, key, ticker);
    }
}
```

6. Publish message to Aiven Kafka

Spring REST controller will be used to simulate data creation through webhook events.

Create a REST controller class like the screenshot below:

```
@RestController
@RequestMapping(value = "/exercise")
public class ExerciseRestController {

    //Inject the KafkaProducer service
    @Autowired
    private KafkaProducer kafkaProducer;

    //get Timestamp in ISO8601 format
    private String getTimestamp(){
        return ZonedDateTime.now( ZoneOffset.UTC ).format( DateTimeFormatter.ISO_INSTANT );
    }

    //Publish Kafka Even everytime a GET request is received on URI /exercise/publish
    @GetMapping(value = "/publish")
    public void sendMessageToKafkaTopic(@RequestParam("symbol") String symbol,
                                       @RequestParam("buy") String buy,
                                       @RequestParam("sell") String sell) {
        //Initiate Ticker object, from request parameter
        Ticker ticker = new Ticker();

        ticker.setSymbol(symbol);
        ticker.setBuy(buy);
        ticker.setSell(sell);
        ticker.setTimestamp(this.getTimestamp());

        //Publish Ticker Object to Kafka broker in JSON format
        this.kafkaProducer.sendMessage(ticker);
    }
}
```

The sample Spring boot application will start a Tomcat server on port 8080. to host our REST API that publishes messages to Aiven Kafka. Message publishing to Kafka can be triggered by running the following curl command, or opening the URL on a web browser.

```
curl "http://localhost:8080/exercise/publish?symbol=AMZN&buy=1000&sell=1100"
```

Feel free to change the request parameters.

Once the curl command is completed successfully, message will be published into Aiven Kafka. To check the message on Aiven-Kafka dashboard, go to Kafka > Topic > stockTicker and click “Fetch Message”.

The messages received by Kafka broker on stockTicker topic will be displayed like the below screenshot.

Messages

▼ PARTITION: **Show all** ▼ ▼ OFFSET: **0** ▼ ▼ TIMEOUT (S): **3** ▼ ▼ MAX BYTES: **Unlimited** ▼ ▼ FORMAT: **json** ▼

Fetch Messages Produce Message

Messages 1-3 of 3 · Page 1

Meta	Key	Value
OFFSET: +2 PARTITION: 0	"05682a0b-cd31-4f8f-9b81-320f51770782"	<pre>{ "buy": "1000", "sell": "1100", "symbol": "AMZN", "timestamp": "2022-02-20T13:23:01.725246Z" }</pre>
OFFSET: +1 PARTITION: 0	"89b6eda4-df07-46bb-91c3-d8cf69412a1a"	<pre>{ "buy": "169", "sell": "170", "symbol": "AAPL", "timestamp": "2022-02-20T13:22:58.932687Z" }</pre> <p>▶ { ... } 4 items</p>
OFFSET: 0 PARTITION: 0	"3c97b42c-7574-45ea-ae6d-63f0836bfc0b"	<pre>{ "buy": "169", "sell": "170", "symbol": "RIVN", "timestamp": "2022-02-20T13:22:25.313739Z" }</pre> <p>▶ { ... } 4 items</p>

7. Observability and Monitoring with Grafana

Now that Aiven-Kafka service is running and receiving messages successfully from the producer in the Spring Boot application. Grafana can be enabled to monitor important metrics like CPU, Memory, storage usage, and more.

First step to enable metrics collection is by enabling Influx DB (The metric database) service in Aiven Dashboard


 **influx-2e798dc9**
InfluxDB • Running

Once Influx DB is up and running, service integration can be configured by clicking on the “Manage Integration” button on the Influx DB service overview page.


On the “Manage Integration” page enable Dashboard integration for a new Grafana Service, and Metrics integration for existing Apache Kafka Service. If done correctly the two service integration, like the screenshot below, will be listed as active.

Service integrations for influx-2e798dc9

Enabled service integrations

**emil-kafka-exercise**
Receive service metrics from service • active

Delete

**grafana-150578c0**
Provide a datasource for Grafana service • active

Delete

To view the Grafana dashboard with Aiven-Kafka service metrics, click on the Service URI link on the Grafana service overview page, and login with the username and password provided on the same page.

Overview Metrics Logs Backups

Connection information

Service URI

<https://grafana-150578c0-sulistya-e019.aivencloud.com>

On the grafana dashboard, click the dashboard icon and select “Aiven-Kafka Resources” dashboard to view metrics related to the Aiven Kafka service.

