Table of Contents

[Setup Environment 2](#_Toc74820882)

[Stream data from Twitter. 3](#_Toc74820883)

[Create a Mock to simulate twitter streams 6](#_Toc74820884)

[Create a Common Configuration Data 11](#_Toc74820885)

[Setup Kafka 16](#_Toc74820886)

[Install Docker MacOS 19](#_Toc74820887)

[Create Kafka Model 20](#_Toc74820888)

[Prepare the component 20](#_Toc74820889)

[Creating Kafka Admin Part1 24](#_Toc74820890)

[Create class RetryConfigData 26](#_Toc74820891)

[Create KafkaConfigData 27](#_Toc74820892)

[Create class RetryConfig in common-config 27](#_Toc74820893)

[Create file KafkaAdminClient 29](#_Toc74820894)

[Create file KafkaAdminConfig 30](#_Toc74820895)

[Create kafka-producer 32](#_Toc74820896)

[Apply Dependencies 32](#_Toc74820897)

[Change ai-sandbox pom.xml 32](#_Toc74820898)

[Change Twitter-to-kafka-service pom.xml 35](#_Toc74820899)

[Change Kafka-admin pom.xml 36](#_Toc74820900)

[Change Kafka-model pom.xml 37](#_Toc74820901)

[Change Kafka-producer pom.xml 38](#_Toc74820902)

[Change application.yml file 39](#_Toc74820903)

[Create KafkaProducerConfigData 40](#_Toc74820904)

[Create KafkaProducerConfig 40](#_Toc74820905)

[Create KafkaProducer service folder 42](#_Toc74820906)

[Integrate Kafka Module 44](#_Toc74820907)

[Change KafkaAdminClient 44](#_Toc74820908)

[Create KafkaClientException 47](#_Toc74820909)

[Create StreamInitializer 48](#_Toc74820910)

[Create KafkaStreamInitializer 48](#_Toc74820911)

# Setup Environment

Download Maven

[Maven – Download Apache Maven](https://maven.apache.org/download.cgi)

Setup maven in MacOS

[How to install Maven on macOS - Mkyong.com](https://mkyong.com/maven/install-maven-on-mac-osx/)

How to use bash files in MacOs

[Use zsh as the default shell on your Mac - Apple Support](https://support.apple.com/en-us/HT208050)

Download Sprint Assistant for Intellij

cat .zshrc

export MAVEN\_HOME=~/apache-maven-3.8.1

export PATH=$PATH:$MAVEN\_HOME/bin

PS1="$"

# Stream data from Twitter.

Text

Description automatically generated

Step 1

* Collect the information from twitter4j.
* Setup developer twitter keys, secrets.

Text

Description automatically generatedGraphical user interface, application

Description automatically generated

debug=true  
oauth.consumerKey=xxxxxxxxxxx  
oauth.consumerSecret=xxxxxxxxxxx  
oauth.accessToken=xxxxxxxxxxxxxxxx  
oauth.accessTokenSecret=xxxxxxxxxxxxxx

Create a listener to start streaming Twitter data.

We can check the examples on twitter4j site,

in the streaming part. We could actually implement StatusListener interface directly.

But I don't want to implement all the methods, so I will simply extend StatusAdapter class,

and then override only onStatus method.

Create package listener

Create class TwitterKafkaStatusListener.

Extend the StatusAdapter class from twitter4j.

Implement the method onStatus to write some logging inside this method for now.

Create a logger here using LoggerFactory class of slf4j and log a message in info level.

We will print the text property of the Status class here, where this Status class is again a

class from twitter4j library and it represents a twitter object.

Annotate the class with Component annotation to make it a Spring managed bean.

Note that, all classes that annotated with Component annotation will be scanned and loaded as Spring bean at runtime.

That are also more specific annotations like Controller for Apis, Service for business layer, Repository for data layer and Configuration for config classes.

In the end, all these annotations will be scanned and loaded by Spring at runtime.

Now I'm going to create an interface called StreamRunner.

Create a package runner,

and then create this stream interface with a single start method,

which will throw a TwitterException and has a void return type.

Then we will implement this interface, with TwitterKafkaStreamRunner class, which will be under impl package.

Let's first implement StreamRunner and override start method here.

And then annotate this class with Component annotation to make it a Spring managed bean.

All right, let's inject the twitter listener and the config classes we created previously, using constructor injection

here.

Let's also create a global TwitterStream variable, which is a twitter4j library class again,

and then we will start implementing start method, by first creating a stream, using TwitterStreamFactory

getInstance method.

And then we simply add the listener we just injected into this Twitter stream.

Then first, I will extract the keywords as String array from the configuration object using toArray method

on the twitter keywords list object.

And then I will create a FilterQuery from twitter4j library with the keywords array.

Then we will simply add the FilterQuery to the Twitter stream.

Here, let's also create a logger using LoggerFactory to log some messages, and then write a simple info

message here,

specifying that, we started filtering Twitter stream for the configured keywords.

We can actually create a new method for this adding filter task.

So I will refactor here and create a method, addFilter, and extract this part as a new method.

I will also create a shutdown method and mark with PreDestroy annotation, and then close the Twitter stream

here.

Let's also add a bit of logging after checking stream is not null.

This method will be called before the bean destroyed.

That is before the application shutdown, so that we will be sure that, stream connection will be closed

prior to application close.

All right, finally, let's inject the runner we created, into the main class, and call the start method in

the run method.

Let's first inject the StreamRunner to Application class by adding the StreamRunner into the current constructor.

Then we will call the start method, and this way we will trigger the streaming logic on application start,

and it will listen twitter messages continuously for us.

All right, let's now run mvn install command to see we have a successful build, and to install dependencies.

Whenever you add new dependencies you can run mvn install command to install the dependencies

from Maven repository.

We can now run the application and we should see the tweets in the console if everything works correctly.

# Create a Mock to simulate twitter streams

* **Change twitter.properties adding variable to control the process**

enable-mock-tweets: false

mock-min-tweet-length: 5

mock-max-tweet-length: 15

mock-sleep-ms: 10000

* **Change TwitterToKafkaServiceConfigData**

To read the new variables from twitter.properties

package com.microservices.demo.twitter.to.kafka.service.config;  
  
import lombok.Data;  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.context.annotation.Configuration;  
  
import java.util.List;  
  
@Data  
@Configuration  
@ConfigurationProperties(prefix = "twitter-to-kafka-service")  
public class TwitterToKafkaServiceConfigData {  
 private List<String> twitterKeywords;  
 private String welcomeMessage;  
 private Boolean enableMockTweets;  
 private Long mockSleepMs;  
 private Integer mockMinTweetLength;  
 private Integer mockMaxTweetLength;  
}

* Create TwitterToKafkaServiceException

package com.ai.sandbox.twitter.to.kafka.service.exception;  
  
public class TwitterToKafkaServiceException extends RuntimeException {  
  
 public TwitterToKafkaServiceException() {  
 super();  
 }  
  
 public TwitterToKafkaServiceException(String message) {  
 super(message);  
 }  
  
 public TwitterToKafkaServiceException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

* Create MockKafkaStreamRunner

package com.ai.sandbox.twitter.to.kafka.service.runner.impl;  
  
import com.ai.sandbox.twitter.to.kafka.service.config.TwitterToKafkaServiceConfigData;  
import com.ai.sandbox.twitter.to.kafka.service.exception.TwitterToKafkaServiceException;  
import com.ai.sandbox.twitter.to.kafka.service.listener.TwitterKafkaStatusListener;  
import com.ai.sandbox.twitter.to.kafka.service.runner.StreamRunner;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;  
import org.springframework.stereotype.Component;  
import twitter4j.Status;  
import twitter4j.TwitterException;  
import twitter4j.TwitterObjectFactory;  
  
import java.time.ZonedDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Arrays;  
import java.util.Locale;  
import java.util.Random;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ThreadLocalRandom;  
  
@Component  
@ConditionalOnProperty(name = "twitter-to-kafka-service.enable-mock-tweets", havingValue = "true")  
public class MockKafkaStreamRunner implements StreamRunner {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(MockKafkaStreamRunner.class);  
  
 private final TwitterToKafkaServiceConfigData twitterToKafkaServiceConfigData;  
  
 private final TwitterKafkaStatusListener twitterKafkaStatusListener;  
  
 private static final Random *RANDOM* = new Random();  
  
 private static final String[] *WORDS* = new String[]{  
 "Lorem",  
 "ipsum",  
 "dolor",  
 "sit",  
 "amet",  
 "consectetuer",  
 "adipiscing",  
 "elit",  
 "Maecenas",  
 "porttitor",  
 "congue",  
 "massa",  
 "Fusce",  
 "posuere",  
 "magna",  
 "sed",  
 "pulvinar",  
 "ultricies",  
 "purus",  
 "lectus",  
 "malesuada",  
 "libero"  
 };  
  
 private static final String *tweetAsRawJson* = "{" +  
 "\"created\_at\":\"{0}\"," +  
 "\"id\":\"{1}\"," +  
 "\"text\":\"{2}\"," +  
 "\"user\":{\"id\":\"{3}\"}" +  
 "}";  
  
 private static final String *TWITTER\_STATUS\_DATE\_FORMAT* = "EEE MMM dd HH:mm:ss zzz yyyy";  
  
 public MockKafkaStreamRunner(TwitterToKafkaServiceConfigData configData,  
 TwitterKafkaStatusListener statusListener) {  
 this.twitterToKafkaServiceConfigData = configData;  
 this.twitterKafkaStatusListener = statusListener;  
 }  
  
 @Override  
 public void start() throws TwitterException {  
 final String[] keywords = twitterToKafkaServiceConfigData.getTwitterKeywords().toArray(new String[0]);  
 final int minTweetLength = twitterToKafkaServiceConfigData.getMockMinTweetLength();  
 final int maxTweetLength = twitterToKafkaServiceConfigData.getMockMaxTweetLength();  
 long sleepTimeMs = twitterToKafkaServiceConfigData.getMockSleepMs();  
 *LOG*.info("Starting mock filtering twitter streams for keywords {}", Arrays.*toString*(keywords));  
 simulateTwitterStream(keywords, minTweetLength, maxTweetLength, sleepTimeMs);  
 }  
  
 private void simulateTwitterStream(String[] keywords, int minTweetLength, int maxTweetLength, long sleepTimeMs) {  
 Executors.*newSingleThreadExecutor*().submit(() -> {  
 try {  
 while (true) {  
 String formattedTweetAsRawJson = getFormattedTweet(keywords, minTweetLength, maxTweetLength);  
 Status status = TwitterObjectFactory.*createStatus*(formattedTweetAsRawJson);  
 twitterKafkaStatusListener.onStatus(status);  
 sleep(sleepTimeMs);  
 }  
 } catch (TwitterException e) {  
 *LOG*.error("Error creating twitter status!", e);  
 }  
 });  
 }  
  
 private void sleep(long sleepTimeMs) {  
 try {  
 Thread.*sleep*(sleepTimeMs);  
 } catch (InterruptedException e) {  
 throw new TwitterToKafkaServiceException("Error while sleeping for waiting new status to create!!");  
 }  
 }  
  
 private String getFormattedTweet(String[] keywords, int minTweetLength, int maxTweetLength) {  
 String[] params = new String[]{  
 ZonedDateTime.*now*().format(DateTimeFormatter.*ofPattern*(*TWITTER\_STATUS\_DATE\_FORMAT*, Locale.*ENGLISH*)),  
 String.*valueOf*(ThreadLocalRandom.*current*().nextLong(Long.*MAX\_VALUE*)),  
 getRandomTweetContent(keywords, minTweetLength, maxTweetLength),  
 String.*valueOf*(ThreadLocalRandom.*current*().nextLong(Long.*MAX\_VALUE*))  
 };  
 return formatTweetAsJsonWithParams(params);  
 }  
  
 private String formatTweetAsJsonWithParams(String[] params) {  
 String tweet = *tweetAsRawJson*;  
  
 for (int i = 0; i < params.length; i++) {  
 tweet = tweet.replace("{" + i + "}", params[i]);  
 }  
 return tweet;  
 }  
  
 private String getRandomTweetContent(String[] keywords, int minTweetLength, int maxTweetLength) {  
 StringBuilder tweet = new StringBuilder();  
 int tweetLength = *RANDOM*.nextInt(maxTweetLength - minTweetLength + 1) + minTweetLength;  
 return constructRandomTweet(keywords, tweet, tweetLength);  
 }  
  
 private String constructRandomTweet(String[] keywords, StringBuilder tweet, int tweetLength) {  
 for (int i = 0; i < tweetLength; i++) {  
 tweet.append(*WORDS*[*RANDOM*.nextInt(*WORDS*.length)]).append(" ");  
 if (i == tweetLength / 2) {  
 tweet.append(keywords[*RANDOM*.nextInt(keywords.length)]).append(" ");  
 }  
 }  
 return tweet.toString().trim();  
 }  
  
}

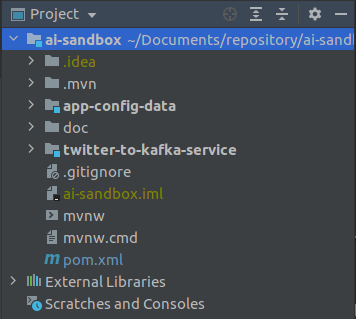
* **Change TwitterKafkaStreamRunner**

Added the ConditionalProperty to allow condiotnal execution based in the proper enable-mock-tweets

@ConditionalOnProperty(name = "twitter-to-kafka-service.enable-mock-tweets", havingValue = "false", matchIfMissing = true)  
public class TwitterKafkaStreamRunner implements StreamRunner {

# Create a Common Configuration Data

Create the app-config-data



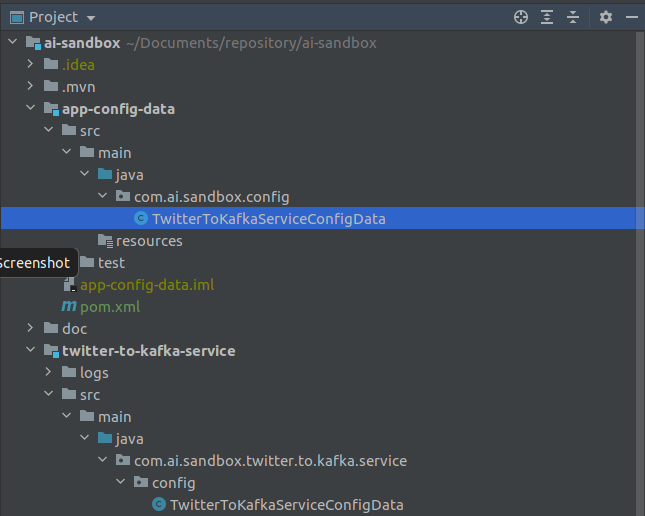
Change the pom.xml adding the standart dependencies

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <scope>provided</scope>  
 </dependency>  
</dependencies>

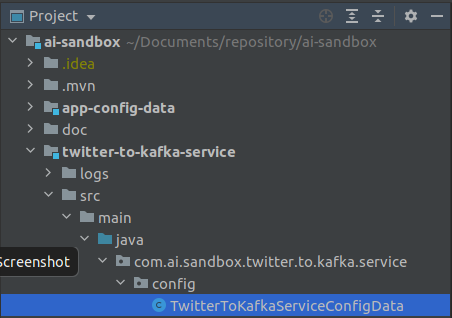
Create a new package

com.ai.sandbox.config

move from twitter-to-kafka-service microservice config to this new microservice package config



Delete the config file from twitter-to-kafka-service



Change pom.xml file of twitter-to-kafka-service adding the app-config-data dependency to allow the TwitterToKafkaServiceConfigData see the new configuration file

<dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <scope>compile</scope>  
</dependency>

Remove projec Lombok out of th pom.xml because the app-config-data pom.xml will have the lombok

this is the file result

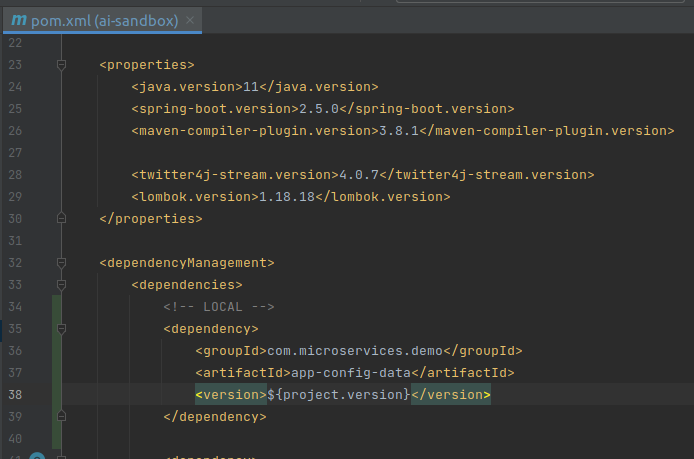
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>twitter-to-kafka-service</artifactId>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.twitter4j</groupId>  
 <artifactId>twitter4j-stream</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <scope>compile</scope>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

Change TwitterToKafkaServiceApplication to point to import the new config file

import com.ai.sandbox.config.TwitterToKafkaServiceConfigData;

Attention: Intellij can do this changes following the directions, once you delete the config , the dependencies will turn red and then the IDE will tell you what to do.

Change the ai-sandbox pom.xml adding the new module



Change the files MockKafkaStreamRunner, TwitterKafkaStreamRunner to point to the new config file, then build the project.

# Setup Kafka

Create the folder docker-compose containg the files .env, kafka.yml, common.yml

.env

COMPOSE\_PATH\_SEPARATOR=:  
COMPOSE\_FILE=common.yml:kafka\_cluster.yml:services.yml  
KAFKA\_VERSION=5.0.4  
ELASTIC\_VERSION=7.9.1  
SERVICE\_VERSION=0.0.1-SNAPSHOT  
GLOBAL\_NETWORK=application  
GROUP\_ID=com.ai.sandbox

common.yml

version: '3.7'  
  
networks:  
 application:  
 driver: bridge

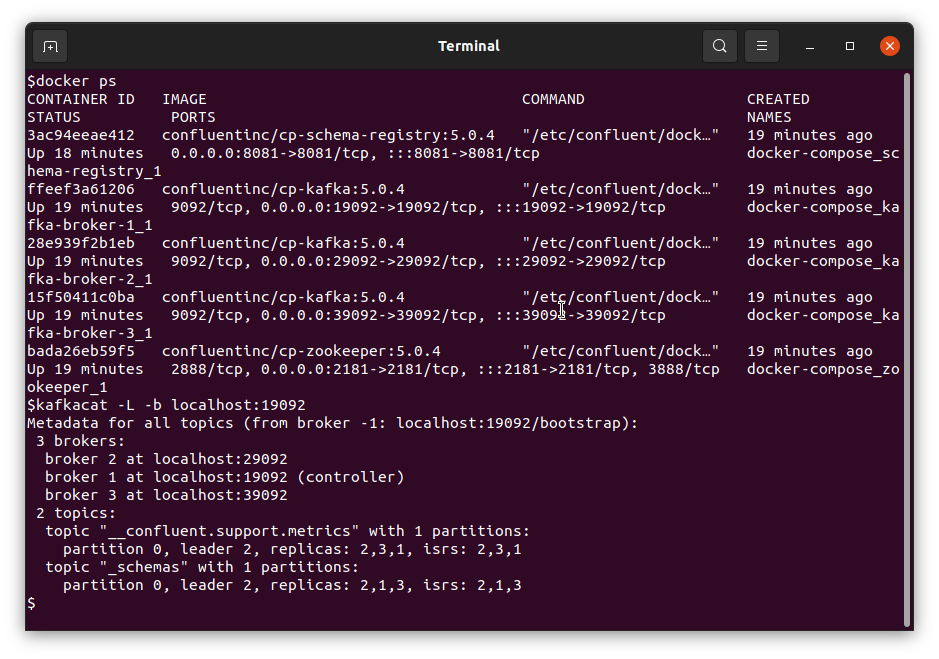
kafka\_cluster.yml

version: '3.7'  
services:  
 zookeeper:  
 image: confluentinc/cp-zookeeper:${KAFKA\_VERSION:-latest}  
 hostname: zookeeper  
 ports:  
 - "2181:2181"  
 environment:  
 ZOOKEEPER\_SERVER\_ID: 1  
 ZOOKEEPER\_CLIENT\_PORT: 2181  
 ZOOKEEPER\_TICK\_TIME: 2000  
 ZOOKEEPER\_INIT\_LIMIT: 5  
 ZOOKEEPER\_SYNC\_LIMIT: 2  
 ZOOKEEPER\_SERVERS: zookeeper:2888:3888  
 networks:  
 - ${GLOBAL\_NETWORK:-kafka}  
 schema-registry:  
 image: confluentinc/cp-schema-registry:${KAFKA\_VERSION}  
 hostname: schema-registry  
 depends\_on:  
 - zookeeper  
 - kafka-broker-1  
 - kafka-broker-2  
 - kafka-broker-3  
 ports:  
 - "8081:8081"  
 environment:  
 SCHEMA\_REGISTRY\_HOST\_NAME: schema-registry  
 SCHEMA\_REGISTRY\_KAFKASTORE\_CONNECTION\_URL: 'zookeeper:2181'  
 SCHEMA\_REGISTRY\_LISTENERS: http://schema-registry:8081  
 SCHEMA\_REGISTRY\_DEBUG: 'true'  
 networks:  
 - ${GLOBAL\_NETWORK:-kafka}  
 kafka-broker-1:  
 image: confluentinc/cp-kafka:${KAFKA\_VERSION}  
 hostname: kafka-broker-1  
 ports:  
 - "19092:19092"  
 depends\_on:  
 - zookeeper  
 environment:  
 KAFKA\_BROKER\_ID: 1  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka-broker-1:9092,LISTENER\_LOCAL://localhost:19092  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,LISTENER\_LOCAL:PLAINTEXT  
 KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT  
 KAFKA\_COMPRESSION\_TYPE: producer  
 networks:  
 - ${GLOBAL\_NETWORK:-kafka}  
 kafka-broker-2:  
 image: confluentinc/cp-kafka:${KAFKA\_VERSION}  
 hostname: kafka-broker-2  
 ports:  
 - "29092:29092"  
 depends\_on:  
 - zookeeper  
 environment:  
 KAFKA\_BROKER\_ID: 2  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka-broker-2:9092,LISTENER\_LOCAL://localhost:29092  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,LISTENER\_LOCAL:PLAINTEXT  
 KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT  
 KAFKA\_COMPRESSION\_TYPE: producer  
 networks:  
 - ${GLOBAL\_NETWORK:-kafka}  
 kafka-broker-3:  
 image: confluentinc/cp-kafka:${KAFKA\_VERSION}  
 hostname: kafka-broker-3  
 ports:  
 - "39092:39092"  
 depends\_on:  
 - zookeeper  
 environment:  
 KAFKA\_BROKER\_ID: 3  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka-broker-3:9092,LISTENER\_LOCAL://localhost:39092  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,LISTENER\_LOCAL:PLAINTEXT  
 KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT  
 KAFKA\_COMPRESSION\_TYPE: producer  
 networks:  
 - ${GLOBAL\_NETWORK:-kafka}

Testing the docker containers for kafka

run docker ps to see the brokers

run kafkacat -L -b localhost:19092



How to use kafkacat

https://docs.confluent.io/platform/current/app-development/kafkacat-usage.html

# Install Docker MacOS

Install docker-compose

docker-compose -f common.yml -f kafka\_cluster.yml up

docker-compose -f common.yml -f kafka\_cluster.yml down

Change permission

sudo chown -R $(whoami) /usr/local/lib/pkgconfig

chmod u+w /usr/local/lib/pkgconfig

Install kafkacat

brew install kafkacat

Test Kafka

kafkacat -L -b localhost:19092

# Create Kafka Model

The Kafka model will be created for kafka communication, and it will use Avro schema type. Avro is an open data serialization system that helps with data exchange between different systems, and it is efficient by serialization and there is no need to keep names as in Json, and it also has a direct mapping to and from Json.

## Prepare the component

**Create the folders**

src/main/resources/avro

**Install Apache Avro IDL Schema Plugin**

Graphical user interface, text

Description automatically generated

**Install maven Avro dependency**

<https://mvnrepository.com/artifact/org.apache.avro/avro/1.10.2>

Documentation

<https://avro.apache.org/index.html>

Change main ai-sandbox pom.xml adding the new dependency

<properties>

:  
 <avro.version>1.10.2</avro.version>  
 </properties>

<dependency>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro</artifactId>  
 <version>${avro.version}</version>  
</dependency>

Change kafka-model pom.xml file adding the avro definitions

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 <relativePath>../../pom.xml</relativePath>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>kafka-model</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro</artifactId>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro-maven-plugin</artifactId>  
 <version>${avro.version}</version>  
 <configuration>  
 <stringType>String</stringType>  
 </configuration>  
 <executions>  
 <execution>  
 <phase>generate-sources</phase>  
 <goals>  
 <goal>schema</goal>  
 </goals>  
 <configuration>  
 <sourceDirectory>${project.basedir}/src/main/resources/avro/</sourceDirectory>  
 <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Create the new file the twitter.avsc under avro folder**

{"namespace": "com.ai.sandbox.kafka.avro.model",

"type": "record",

"name": "TwitterAvroModel",

"fields": [

{"name": "userId", "type": "long"},

{"name": "id", "type": "long"},

{"name": "text", "type": ["null", "string"]},

{"name": "createdAt", "type": ["null", "long"], "logicalType": ["null", "date"]}

]

}

Open the terminal in the kafka-model folder and run the mvn to build the source code

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with medium confidence

This should be the results, the files in the red boxes were auto generated.

Graphical user interface, text

Description automatically generated

# Creating Kafka Admin Part1

Change kafka-admin pom.xml adding the app-config-data dependency

<dependencies>  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 </dependency>

<dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 </dependency>  
  
</dependencies>

Change ai-sandbox pom.xml file adding the spring kafka dependency

https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka/2.7.2

<properties>

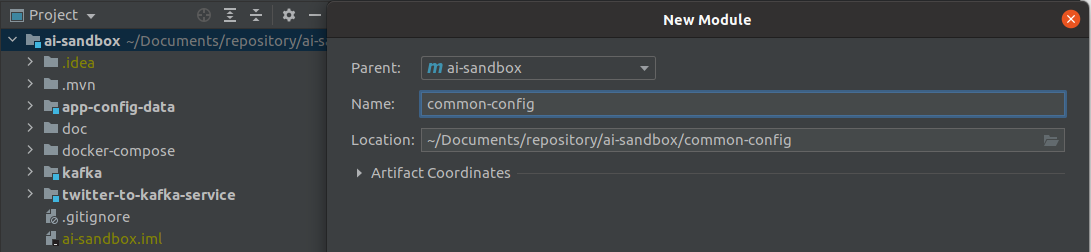
:  
 <spring-kafka.version>2.7.2</spring-kafka.version>  
</properties>

<dependencies>

:

<dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 <version>${spring-kafka.version}</version>  
 </dependency>  
   
</dependencies>

Create a common-config



https://mvnrepository.com/artifact/org.springframework.retry/spring-retry/1.3.1

Change common-config pom.xml file

<dependencies>  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.retry</groupId>  
 <artifactId>spring-retry</artifactId>  
 </dependency>  
</dependencies>

Change ai-sandbox pom.xml file

<properties>

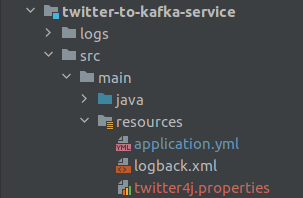
:

<spring-retry.version>1.3.1</spring-retry.version>  
</properties>

<dependencies>

:  
 <dependency>  
 <groupId>org.springframework.retry</groupId>  
 <artifactId>spring-retry</artifactId>  
 <version>${spring-retry.version}</version>  
 </dependency>  
  
</dependencies>

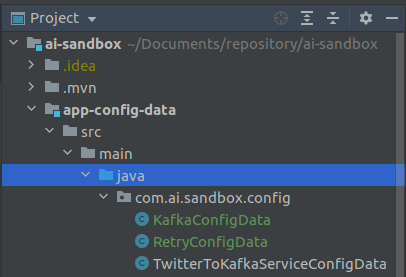
Change application.yml in twitter-to-kafka-service



retry-config:  
 initial-interval-ms: 1000  
 max-interval-ms: 10000  
 multiplier: 2.0  
 maxAttempts: 3  
 sleep-time-ms: 2000

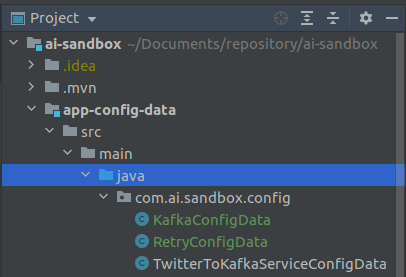
kafka-config:  
 bootstrap-servers: localhost:19092, localhost:29092, localhost:39092  
 schema-registry-url-key: schema.registry.url  
 schema-registry-url: http://localhost:8081  
 topic-name: twitter-topic  
 topic-names-to-create:  
 - twitter-topic  
 num-of-partitions: 3  
 replication-factor: 3

## Create class RetryConfigData



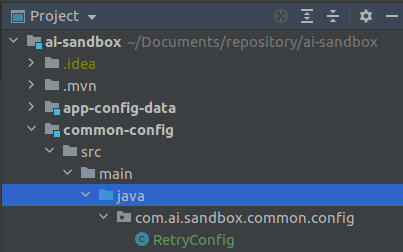
package com.ai.sandbox.config;  
  
import lombok.Data;  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.context.annotation.Configuration;  
  
@Data  
@Configuration  
@ConfigurationProperties(prefix = "retry-config")  
public class RetryConfigData {  
 private Long initialIntervalMs;  
 private Long maxIntervalMs;  
 private Double multiplier;  
 private Integer maxAttempts;  
 private Long sleepTimeMs;  
}

## Create KafkaConfigData



package com.ai.sandbox.config;  
  
import lombok.Data;  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.context.annotation.Configuration;  
import java.util.List;  
  
@Data  
@Configuration  
@ConfigurationProperties(prefix = "kafka-config")  
public class KafkaConfigData {  
 private String bootstrapServers;  
 private String schemaRegistryUrlKey;  
 private String schemaRegistryUrl;  
 private String topicName;  
 private List<String> topicNamesToCreate;  
 private Integer numOfPartitions;  
 private Short replicationFactor;  
}

## Create class RetryConfig in common-config



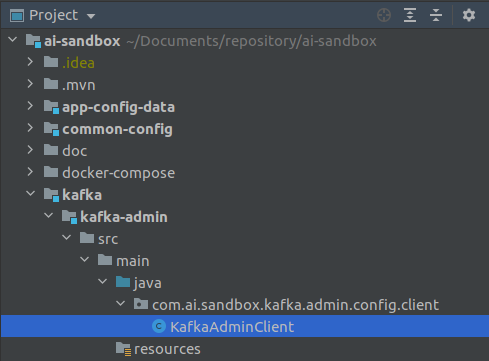
package com.ai.sandbox.common.config;  
  
import com.ai.sandbox.config.RetryConfigData;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.retry.backoff.ExponentialBackOffPolicy;  
import org.springframework.retry.policy.SimpleRetryPolicy;  
import org.springframework.retry.support.RetryTemplate;  
  
@Configuration  
public class RetryConfig {  
  
 private final RetryConfigData retryConfigData;  
  
 public RetryConfig(RetryConfigData configData) {  
 this.retryConfigData = configData;  
 }  
  
 @Bean  
 public RetryTemplate retryTemplate() {  
 RetryTemplate retryTemplate = new RetryTemplate();  
  
 ExponentialBackOffPolicy exponentialBackOffPolicy = new ExponentialBackOffPolicy();  
 exponentialBackOffPolicy.setInitialInterval(retryConfigData.getInitialIntervalMs());  
 exponentialBackOffPolicy.setMaxInterval(retryConfigData.getMaxIntervalMs());  
 exponentialBackOffPolicy.setMultiplier(retryConfigData.getMultiplier());  
  
 retryTemplate.setBackOffPolicy(exponentialBackOffPolicy);  
  
 SimpleRetryPolicy simpleRetryPolicy = new SimpleRetryPolicy();  
 simpleRetryPolicy.setMaxAttempts(retryConfigData.getMaxAttempts());  
  
 retryTemplate.setRetryPolicy(simpleRetryPolicy);  
  
 return retryTemplate;  
 }  
}

Change ai-sandbox pom.xml

<modules>  
 <module>twitter-to-kafka-service</module>  
 <module>app-config-data</module>  
 <module>kafka</module>  
 <module>kafka/kafka-model</module>  
 <module>kafka/kafka-admin</module>  
 <module>kafka/kafka-producer</module>  
 <module>common-config</module>  
</modules>

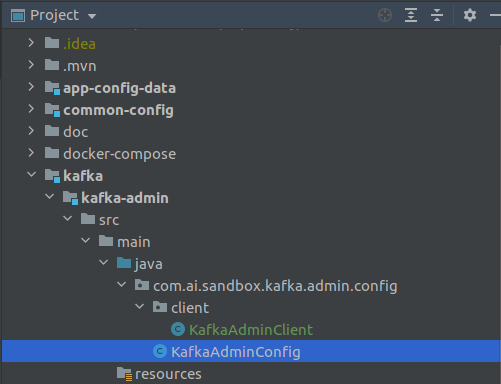
<dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>common-config</artifactId>  
 <version>${project.version}</version>  
</dependency>

## Create file KafkaAdminClient



package com.ai.sandbox.kafka.admin.config.client;  
  
import com.ai.sandbox.config.KafkaConfigData;  
import com.ai.sandbox.config.RetryConfigData;  
import org.apache.kafka.clients.admin.AdminClient;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.retry.support.RetryTemplate;  
import org.springframework.stereotype.Component;  
  
@Component  
public class KafkaAdminClient {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(KafkaAdminClient.class);  
  
 private KafkaConfigData kafkaConfigData;  
  
 private RetryConfigData retryConfigData;  
  
 private AdminClient adminClient;  
  
 private RetryTemplate retryTemplate;  
  
 public KafkaAdminClient(KafkaConfigData config,  
 RetryConfigData retryConfigData,  
 AdminClient client,  
 RetryTemplate template) {  
 this.kafkaConfigData = config;  
 this.retryConfigData = retryConfigData;  
 this.adminClient = client;  
 this.retryTemplate = template;  
 }  
}

## Create file KafkaAdminConfig



package com.ai.sandbox.kafka.admin.config;  
  
import com.ai.sandbox.config.KafkaConfigData;  
import org.apache.kafka.clients.CommonClientConfigs;  
import org.apache.kafka.clients.admin.AdminClient;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.retry.annotation.EnableRetry;  
  
import java.util.Map;  
  
@EnableRetry  
@Configuration  
public class KafkaAdminConfig {  
  
 private final KafkaConfigData kafkaConfigData;  
  
 public KafkaAdminConfig(KafkaConfigData configData) {  
 this.kafkaConfigData = configData;  
 }  
  
 @Bean  
 public AdminClient adminClient() {  
 return AdminClient.*create*(Map.*of*(CommonClientConfigs.*BOOTSTRAP\_SERVERS\_CONFIG*,  
 kafkaConfigData.getBootstrapServers()));  
 }  
}

# Create kafka-producer

First Change the pom.xml file for ai-sandbox and the other pom.xml file from the module

## Apply Dependencies

### Change ai-sandbox pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <modules>  
 <module>twitter-to-kafka-service</module>  
 <module>app-config-data</module>  
 <module>kafka</module>  
 <module>kafka/kafka-model</module>  
 <module>kafka/kafka-admin</module>  
 <module>kafka/kafka-producer</module>  
 <module>common-config</module>  
 </modules>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.5.0</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>ai-sandbox</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>ai-sandbox</name>  
 <description>AI Sandbox project</description>  
  
 <packaging>pom</packaging>  
  
 <properties>  
 <java.version>11</java.version>  
 <spring-boot.version>2.5.0</spring-boot.version>  
 <maven-compiler-plugin.version>3.8.1</maven-compiler-plugin.version>  
 <twitter4j-stream.version>4.0.7</twitter4j-stream.version>  
 <lombok.version>1.18.18</lombok.version>  
 <avro.version>1.10.2</avro.version>  
 <spring-kafka.version>2.7.2</spring-kafka.version>  
 <spring-retry.version>1.3.1</spring-retry.version>  
 <kafka-avro-serializer.version>6.2.0</kafka-avro-serializer.version>  
 </properties>  
  
 <repositories>  
 <repository>  
 <id>confluent</id>  
 <url>https://packages.confluent.io/maven/</url>  
 </repository>  
 </repositories>  
  
 <dependencyManagement>  
 <dependencies>  
 <!-- LOCAL -->  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 <version>${project.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>common-config</artifactId>  
 <version>${project.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-model</artifactId>  
 <version>${project.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-admin</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-producer</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 <version>${spring-boot.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <version>${spring-boot.version}</version>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 <version>${spring-kafka.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.retry</groupId>  
 <artifactId>spring-retry</artifactId>  
 <version>${spring-retry.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-webflux</artifactId>  
 <version>${spring-boot.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.twitter4j</groupId>  
 <artifactId>twitter4j-stream</artifactId>  
 <version>${twitter4j-stream.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${lombok.version}</version>  
 <scope>provided</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro</artifactId>  
 <version>${avro.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>io.confluent</groupId>  
 <artifactId>kafka-avro-serializer</artifactId>  
 <version>${kafka-avro-serializer.version}</version>  
 <exclusions>  
 <exclusion>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>io.swagger</groupId>  
 <artifactId>swagger-annotations</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>io.swagger</groupId>  
 <artifactId>swagger-core</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>${maven-compiler-plugin.version}</version>  
 <configuration>  
 <release>11</release>  
 </configuration>  
 </plugin>  
 </plugins>  
 <pluginManagement>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <version>${spring-boot.version}</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
  
</project>

### Change Twitter-to-kafka-service pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>twitter-to-kafka-service</artifactId>  
  
 <dependencies>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-admin</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-producer</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.twitter4j</groupId>  
 <artifactId>twitter4j-stream</artifactId>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

### Change Kafka-admin pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 <relativePath>../../pom.xml</relativePath>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>kafka-admin</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>common-config</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-webflux</artifactId>  
 </dependency>  
  
 </dependencies>  
</project>

### Change Kafka-model pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 <relativePath>../../pom.xml</relativePath>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>kafka-model</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro</artifactId>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.avro</groupId>  
 <artifactId>avro-maven-plugin</artifactId>  
 <version>${avro.version}</version>  
 <configuration>  
 <stringType>String</stringType>  
 </configuration>  
 <executions>  
 <execution>  
 <phase>generate-sources</phase>  
 <goals>  
 <goal>schema</goal>  
 </goals>  
 <configuration>  
 <sourceDirectory>${project.basedir}/src/main/resources/avro/</sourceDirectory>  
 <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

### Change Kafka-producer pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>ai-sandbox</artifactId>  
 <groupId>com.ai.sandbox</groupId>  
 <version>0.0.1-SNAPSHOT</version>  
 <relativePath>../../pom.xml</relativePath>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>kafka-producer</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>app-config-data</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.ai.sandbox</groupId>  
 <artifactId>kafka-model</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.confluent</groupId>  
 <artifactId>kafka-avro-serializer</artifactId>  
 </dependency>  
 </dependencies>  
  
</project>

Attention: as you changed the pom.xml click in the button to update the dependencies and once it is done run the command mvn clean install in the root folder to check if everything will compile

Text

Description automatically generated

### Change application.yml file

twitter-to-kafka-service:  
 twitter-keywords:  
 - Java  
 - Microservices  
 - Kafka  
 - Elasticsearch  
 welcome-message: Hello microservices!  
 enable-mock-tweets: false  
 mock-min-tweet-length: 5  
 mock-max-tweet-length: 15  
 mock-sleep-ms: 10000  
  
retry-config:  
 initial-interval-ms: 1000  
 max-interval-ms: 10000  
 multiplier: 2.0  
 maxAttempts: 3  
 sleep-time-ms: 2000  
  
kafka-config:  
 bootstrap-servers: localhost:19092, localhost:29092, localhost:39092  
 schema-registry-url-key: schema.registry.url  
 schema-registry-url: http://localhost:8081  
 topic-name: twitter-topic  
 topic-names-to-create:  
 - twitter-topic  
 num-of-partitions: 3  
 replication-factor: 3

### Create KafkaProducerConfigData

Graphical user interface, text

Description automatically generated

package com.ai.sandbox.config;  
  
import lombok.Data;  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.context.annotation.Configuration;  
  
@Data  
@Configuration  
@ConfigurationProperties(prefix = "kafka-producer-config")  
public class KafkaProducerConfigData {  
 private String keySerializerClass;  
 private String valueSerializerClass;  
 private String compressionType;  
 private String acks;  
 private Integer batchSize;  
 private Integer batchSizeBoostFactor;  
 private Integer lingerMs;  
 private Integer requestTimeoutMs;  
 private Integer retryCount;  
}

### Create KafkaProducerConfig

A screenshot of a computer

Description automatically generated with medium confidence

package com.ai.sandbox.kafka.producer.config;  
  
import com.ai.sandbox.config.KafkaConfigData;  
import com.ai.sandbox.config.KafkaProducerConfigData;  
import org.apache.avro.specific.SpecificRecordBase;  
import org.apache.kafka.clients.producer.ProducerConfig;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.kafka.core.DefaultKafkaProducerFactory;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.kafka.core.ProducerFactory;  
  
import java.io.Serializable;  
import java.util.HashMap;  
import java.util.Map;  
  
@Configuration  
public class KafkaProducerConfig<K extends Serializable, V extends SpecificRecordBase> {  
  
 private final KafkaConfigData kafkaConfigData;  
  
 private final KafkaProducerConfigData kafkaProducerConfigData;  
  
 public KafkaProducerConfig(KafkaConfigData configData, KafkaProducerConfigData producerConfigData) {  
 this.kafkaConfigData = configData;  
 this.kafkaProducerConfigData = producerConfigData;  
 }  
  
 @Bean  
 public Map<String, Object> producerConfig() {  
 Map<String, Object> props = new HashMap<>();  
 props.put(ProducerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, kafkaConfigData.getBootstrapServers());  
 props.put(kafkaConfigData.getSchemaRegistryUrlKey(), kafkaConfigData.getSchemaRegistryUrl());  
 props.put(ProducerConfig.*KEY\_SERIALIZER\_CLASS\_CONFIG*, kafkaProducerConfigData.getKeySerializerClass());  
 props.put(ProducerConfig.*VALUE\_SERIALIZER\_CLASS\_CONFIG*, kafkaProducerConfigData.getValueSerializerClass());  
 props.put(ProducerConfig.*BATCH\_SIZE\_CONFIG*, kafkaProducerConfigData.getBatchSize() \*  
 kafkaProducerConfigData.getBatchSizeBoostFactor());  
 props.put(ProducerConfig.*LINGER\_MS\_CONFIG*, kafkaProducerConfigData.getLingerMs());  
 props.put(ProducerConfig.*COMPRESSION\_TYPE\_CONFIG*, kafkaProducerConfigData.getCompressionType());  
 props.put(ProducerConfig.*ACKS\_CONFIG*, kafkaProducerConfigData.getAcks());  
 props.put(ProducerConfig.*REQUEST\_TIMEOUT\_MS\_CONFIG*, kafkaProducerConfigData.getRequestTimeoutMs());  
 props.put(ProducerConfig.*RETRIES\_CONFIG*, kafkaProducerConfigData.getRetryCount());  
 return props;  
 }  
  
 @Bean  
 public ProducerFactory<K, V> producerFactory() {  
 return new DefaultKafkaProducerFactory<>(producerConfig());  
 }  
  
 @Bean  
 public KafkaTemplate<K, V> kafkaTemplate() {  
 return new KafkaTemplate<>(producerFactory());  
 }  
}

### Create KafkaProducer service folder

A screenshot of a computer

Description automatically generated with medium confidence

package com.ai.sandbox.kafka.producer.config.service;  
import org.apache.avro.specific.SpecificRecordBase;  
  
import java.io.Serializable;  
  
public interface KafkaProducer<K extends Serializable, V extends SpecificRecordBase> {  
 void send(String topicName, K key, V message);  
}

Create TwitterKafkaProducer in kafka-producer module

package com.ai.sandbox.kafka.producer.config.service.impl;  
  
import com.ai.sandbox.kafka.avro.model.TwitterAvroModel;  
import com.ai.sandbox.kafka.producer.config.service.KafkaProducer;  
import org.apache.kafka.clients.producer.RecordMetadata;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.kafka.support.SendResult;  
import org.springframework.stereotype.Service;  
import org.springframework.util.concurrent.ListenableFuture;  
import org.springframework.util.concurrent.ListenableFutureCallback;  
  
import javax.annotation.PreDestroy;  
  
@Service  
public class TwitterKafkaProducer implements KafkaProducer<Long, TwitterAvroModel> {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(TwitterKafkaProducer.class);  
  
 private KafkaTemplate<Long, TwitterAvroModel> kafkaTemplate;  
  
 public TwitterKafkaProducer(KafkaTemplate<Long, TwitterAvroModel> template) {  
 this.kafkaTemplate = template;  
 }  
  
 @Override  
 public void send(String topicName, Long key, TwitterAvroModel message) {  
 *LOG*.info("Sending message='{}' to topic='{}'", message, topicName);  
 ListenableFuture<SendResult<Long, TwitterAvroModel>> kafkaResultFuture =  
 kafkaTemplate.send(topicName, key, message);  
 addCallback(topicName, message, kafkaResultFuture);  
 }  
  
 @PreDestroy  
 public void close() {  
 if (kafkaTemplate != null) {  
 *LOG*.info("Closing kafka producer!");  
 kafkaTemplate.destroy();  
 }  
 }  
  
 private void addCallback(String topicName, TwitterAvroModel message,  
 ListenableFuture<SendResult<Long, TwitterAvroModel>> kafkaResultFuture) {  
 kafkaResultFuture.addCallback(new ListenableFutureCallback<>() {  
 @Override  
 public void onFailure(Throwable throwable) {  
 *LOG*.error("Error while sending message {} to topic {}", message.toString(), topicName, throwable);  
 }  
  
 @Override  
 public void onSuccess(SendResult<Long, TwitterAvroModel> result) {  
 RecordMetadata metadata = result.getRecordMetadata();  
 *LOG*.debug("Received new metadata. Topic: {}; Partition {}; Offset {}; Timestamp {}, at time {}",  
 metadata.topic(),  
 metadata.partition(),  
 metadata.offset(),  
 metadata.timestamp(),  
 System.*nanoTime*());  
 }  
 });  
 }  
}

# Integrate Kafka Module

## Change KafkaAdminClient

Graphical user interface, text, application

Description automatically generated

package com.ai.sandbox.kafka.admin.config.client;  
  
import com.ai.sandbox.config.KafkaConfigData;  
import com.ai.sandbox.config.RetryConfigData;  
import com.ai.sandbox.kafka.admin.exception.KafkaClientException;  
import org.apache.kafka.clients.admin.AdminClient;  
import org.apache.kafka.clients.admin.CreateTopicsResult;  
import org.apache.kafka.clients.admin.NewTopic;  
import org.apache.kafka.clients.admin.TopicListing;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.http.HttpMethod;  
import org.springframework.http.HttpStatus;  
import org.springframework.retry.RetryContext;  
import org.springframework.retry.support.RetryTemplate;  
import org.springframework.stereotype.Component;  
import org.springframework.web.reactive.function.client.ClientResponse;  
import org.springframework.web.reactive.function.client.WebClient;  
  
import java.util.Collection;  
import java.util.List;  
import java.util.concurrent.ExecutionException;  
import java.util.stream.Collectors;  
  
@Component  
public class KafkaAdminClient {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(KafkaAdminClient.class);  
  
 private final KafkaConfigData kafkaConfigData;  
  
 private final RetryConfigData retryConfigData;  
  
 private final AdminClient adminClient;  
  
 private final RetryTemplate retryTemplate;  
  
 private final WebClient webClient;  
  
  
 public KafkaAdminClient(KafkaConfigData config,  
 RetryConfigData retryConfigData,  
 AdminClient client,  
 RetryTemplate template,  
 WebClient webClient) {  
 this.kafkaConfigData = config;  
 this.retryConfigData = retryConfigData;  
 this.adminClient = client;  
 this.retryTemplate = template;  
 this.webClient = webClient;  
 }  
  
 public void createTopics() {  
 CreateTopicsResult createTopicsResult;  
 try {  
 createTopicsResult = retryTemplate.execute(this::doCreateTopics);  
 *LOG*.info("Create topic result {}", createTopicsResult.values().values());  
 } catch (Throwable t) {  
 throw new KafkaClientException("Reached max number of retry for creating kafka topic(s)!", t);  
 }  
 checkTopicsCreated();  
 }  
  
 public void checkTopicsCreated() {  
 Collection<TopicListing> topics = getTopics();  
 int retryCount = 1;  
 Integer maxRetry = retryConfigData.getMaxAttempts();  
 int multiplier = retryConfigData.getMultiplier().intValue();  
 Long sleepTimeMs = retryConfigData.getSleepTimeMs();  
 for (String topic : kafkaConfigData.getTopicNamesToCreate()) {  
 while (!isTopicCreated(topics, topic)) {  
 checkMaxRetry(retryCount++, maxRetry);  
 sleep(sleepTimeMs);  
 sleepTimeMs \*= multiplier;  
 topics = getTopics();  
 }  
 }  
 }  
  
 public void checkSchemaRegistry() {  
 int retryCount = 1;  
 Integer maxRetry = retryConfigData.getMaxAttempts();  
 int multiplier = retryConfigData.getMultiplier().intValue();  
 Long sleepTimeMs = retryConfigData.getSleepTimeMs();  
 while (!getSchemaRegistryStatus().is2xxSuccessful()) {  
 checkMaxRetry(retryCount++, maxRetry);  
 sleep(sleepTimeMs);  
 sleepTimeMs \*= multiplier;  
 }  
 }  
  
 private HttpStatus getSchemaRegistryStatus() {  
 try {  
 return webClient  
 .method(HttpMethod.*GET*)  
 .uri(kafkaConfigData.getSchemaRegistryUrl())  
 .exchange()  
 .map(ClientResponse::statusCode)  
 .block();  
 } catch (Exception e) {  
 return HttpStatus.*SERVICE\_UNAVAILABLE*;  
 }  
 }  
  
  
 private void sleep(Long sleepTimeMs) {  
 try {  
 Thread.*sleep*(sleepTimeMs);  
 } catch (InterruptedException e) {  
 throw new KafkaClientException("Error while sleeping for waiting new created topics!!");  
 }  
 }  
  
 private void checkMaxRetry(int retry, Integer maxRetry) {  
 if (retry > maxRetry) {  
 throw new KafkaClientException("Reached max number of retry for reading kafka topic(s)!");  
 }  
 }  
  
 private boolean isTopicCreated(Collection<TopicListing> topics, String topicName) {  
 if (topics == null) {  
 return false;  
 }  
 return topics.stream().anyMatch(topic -> topic.name().equals(topicName));  
 }  
  
 private CreateTopicsResult doCreateTopics(RetryContext retryContext) {  
 List<String> topicNames = kafkaConfigData.getTopicNamesToCreate();  
 *LOG*.info("Creating {} topics(s), attempt {}", topicNames.size(), retryContext.getRetryCount());  
 List<NewTopic> kafkaTopics = topicNames.stream().map(topic -> new NewTopic(  
 topic.trim(),  
 kafkaConfigData.getNumOfPartitions(),  
 kafkaConfigData.getReplicationFactor()  
 )).collect(Collectors.*toList*());  
 return adminClient.createTopics(kafkaTopics);  
 }  
  
 private Collection<TopicListing> getTopics() {  
 Collection<TopicListing> topics;  
 try {  
 topics = retryTemplate.execute(this::doGetTopics);  
 } catch (Throwable t) {  
 throw new KafkaClientException("Reached max number of retry for reading kafka topic(s)!", t);  
 }  
 return topics;  
 }  
  
 private Collection<TopicListing> doGetTopics(RetryContext retryContext)  
 throws ExecutionException, InterruptedException {  
 *LOG*.info("Reading kafka topic {}, attempt {}",  
 kafkaConfigData.getTopicNamesToCreate().toArray(), retryContext.getRetryCount());  
 Collection<TopicListing> topics = adminClient.listTopics().listings().get();  
 if (topics != null) {  
 topics.forEach(topic -> *LOG*.debug("Topic with name {}", topic.name()));  
 }  
 return topics;  
 }  
  
}  
}

## Create KafkaClientException

A screenshot of a computer

Description automatically generated with low confidence

package com.ai.sandbox.kafka.admin.exception;  
  
public class KafkaClientException extends RuntimeException {  
  
 public KafkaClientException() {  
 }  
  
 public KafkaClientException(String message) {  
 super(message);  
 }  
  
 public KafkaClientException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

## Create StreamInitializer

Graphical user interface, text

Description automatically generated

package com.ai.sandbox.twitter.to.kafka.service.init;  
  
public interface StreamInitializer {  
 void init();  
}

## Create KafkaStreamInitializer

Graphical user interface, text, application

Description automatically generated

package com.ai.sandbox.twitter.to.kafka.service.init.impl;  
import com.ai.sandbox.config.KafkaConfigData;  
import com.ai.sandbox.kafka.admin.config.client.KafkaAdminClient;  
import com.ai.sandbox.twitter.to.kafka.service.init.StreamInitializer;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Component;  
  
@Component  
public class KafkaStreamInitializer implements StreamInitializer {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(KafkaStreamInitializer.class);  
  
 private final KafkaConfigData kafkaConfigData;  
  
 private final KafkaAdminClient kafkaAdminClient;  
  
 public KafkaStreamInitializer(KafkaConfigData configData, KafkaAdminClient adminClient) {  
 this.kafkaConfigData = configData;  
 this.kafkaAdminClient = adminClient;  
 }  
  
 @Override  
 public void init() {  
 kafkaAdminClient.createTopics();  
 *LOG*.info("Topics with name {} is ready for operations!", kafkaConfigData.getTopicNamesToCreate().toArray());  
 }  
}

## Create TwitterStatusToAvroTransformer

Graphical user interface, text, application

Description automatically generated

package com.ai.sandbox.twitter.to.kafka.service.transformer;  
  
import com.ai.sandbox.kafka.avro.model.TwitterAvroModel;  
import org.springframework.stereotype.Component;  
import twitter4j.Status;  
  
@Component  
public class TwitterStatusToAvroTransformer {  
  
 public TwitterAvroModel getTwitterAvroModelFromStatus(Status status) {  
 return TwitterAvroModel  
 .*newBuilder*()  
 .setId(status.getId())  
 .setUserId(status.getUser().getId())  
 .setText(status.getText())  
 .setCreatedAt(status.getCreatedAt().getTime())  
 .build();  
 }  
}

## Change TwitterKafkaStatusListener

Graphical user interface, text, application

Description automatically generated

package com.ai.sandbox.twitter.to.kafka.service.listener;  
import com.ai.sandbox.config.KafkaConfigData;  
import com.ai.sandbox.kafka.avro.model.TwitterAvroModel;  
import com.ai.sandbox.kafka.producer.config.service.KafkaProducer;  
import com.ai.sandbox.twitter.to.kafka.service.transformer.TwitterStatusToAvroTransformer;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Component;  
import twitter4j.Status;  
import twitter4j.StatusAdapter;  
  
@Component  
public class TwitterKafkaStatusListener extends StatusAdapter {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(TwitterKafkaStatusListener.class);  
  
 private final KafkaConfigData kafkaConfigData;  
  
 private final KafkaProducer<Long, TwitterAvroModel> kafkaProducer;  
  
 private final TwitterStatusToAvroTransformer twitterStatusToAvroTransformer;  
  
 public TwitterKafkaStatusListener(KafkaConfigData configData,  
 KafkaProducer<Long, TwitterAvroModel> producer,  
 TwitterStatusToAvroTransformer transformer) {  
 this.kafkaConfigData = configData;  
 this.kafkaProducer = producer;  
 this.twitterStatusToAvroTransformer = transformer;  
 }  
  
 @Override  
 public void onStatus(Status status) {  
 *LOG*.info("Received status text {} sending to kafka topic {}", status.getText(), kafkaConfigData.getTopicName());  
 TwitterAvroModel twitterAvroModel = twitterStatusToAvroTransformer.getTwitterAvroModelFromStatus(status);  
 kafkaProducer.send(kafkaConfigData.getTopicName(), twitterAvroModel.getUserId(), twitterAvroModel);  
 }  
}

## Change TwitterToKafkaServiceApplication

Graphical user interface, text, application

Description automatically generated

package com.ai.sandbox.twitter.to.kafka.service;  
  
import com.ai.sandbox.twitter.to.kafka.service.init.StreamInitializer;  
import com.ai.sandbox.twitter.to.kafka.service.runner.StreamRunner;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.ComponentScan;  
  
@SpringBootApplication  
@ComponentScan(basePackages = "com.ai.sandbox")  
public class TwitterToKafkaServiceApplication implements CommandLineRunner {  
  
 private static final Logger *LOG* = LoggerFactory.*getLogger*(TwitterToKafkaServiceApplication.class);  
  
 private final StreamRunner streamRunner;  
  
 private final StreamInitializer streamInitializer;  
  
 public TwitterToKafkaServiceApplication(StreamRunner runner, StreamInitializer initializer) {  
 this.streamRunner = runner;  
 this.streamInitializer = initializer;  
 }  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(TwitterToKafkaServiceApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 *LOG*.info("App starts...");  
 streamInitializer.init();  
 streamRunner.start();  
 }  
}

Run docker and examine kafka topic running kafkacat

Kafkacat -L -b localhost:19092

Kakfa Consumer

Run the command to see the twitter message being consume by kafka

Kafkacat -C -b localhost:19092 -t twitter-topic

Create docker image for the microservice

Change twitter-to-kafka-service pom.xml file

Create a service.yml file in the docker-compose folder

Run docker-compose

docker-compose -f common.yml -f kafka\_cluster.yml -f services.yml up

or docker-compose up

run kafkacat -L -b localhost:19092

run to display the logs

docker logs <containerid>

or

docker logs -f <containerid>

Run the consumer

kafkacat -C -b localhost:19092 -t twitter-topic

Change services.yml to make the microservice read the data from mock

docker-compose down

change services

docker-compose up