***Kafka Implementation***

***Step 1 : Install open Jdk, please ignore if you have Java***

sudo apt update

sudo yum install java-1.8.0-openjdk

vi .bashrc

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.272.b10-3.el8\_3.x86\_64

. .bashrc

sudo yum install wget

**Step 2 : - Get binary of kafka**

wget <http://www-us.apache.org/dist/kafka/2.4.0/kafka_2.13-2.4.0.tgz>

tar xzf kafka\_2.13-2.4.0.tgz

Then extract the archive file

tar xzf kafka\_2.13-2.4.0.tgz

mv kafka\_2.13-2.4.0 /usr/local/kafka

## Step 3 – run and zookeeper and kafka server

sh zookeeper-server-start.sh ../config/zookeeper.properties &

sh kafka-server-start.sh ../config/server.properties &

All done. The Kafka installation has been successfully completed. The part of this tutorial will help you to work with the Kafka server.

## Step 4 – Create a Topic in Kafka

Kafka provides multiple pre-built shell script to work on it. First, create a topic named “testTopic” with a single partition with single replica:

cd /usr/local/kafka

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic testTopic

## Step 5 – Describe Topic

sh ../bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic testTopic

## Step 6 – Send Messages to Kafka

The “producer” is the process responsible for put data into our Kafka. The Kafka comes with a command-line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster. The default Kafka sends each line as a separate message.

Let’s run the producer and then type a few messages into the console to send to the server.

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testTopic

>Welcome to kafka

>This is my first topic

>

You can exit this command or keep this terminal running for further testing. Now open a new terminal to the Kafka consumer process on the next step.

## Step 7 – Using Kafka Consumer

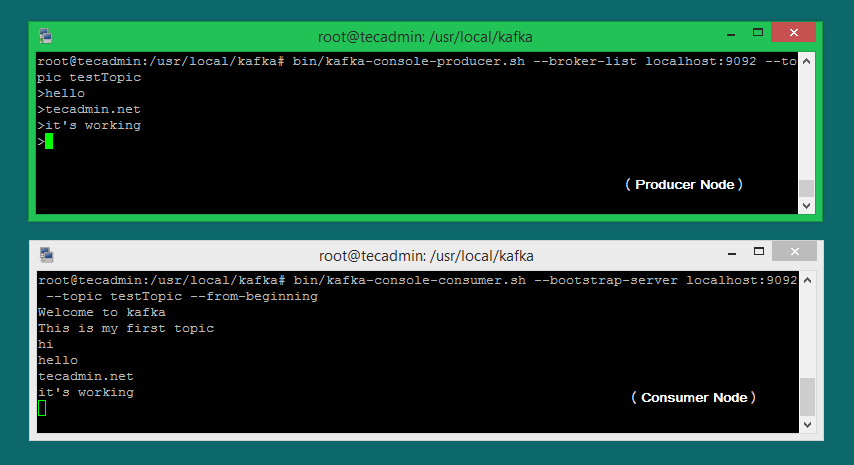
Kafka also has a command-line consumer to read data from the Kafka cluster and display messages to standard output.

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic testTopic --from-beginning

Welcome to kafka

This is my first topic

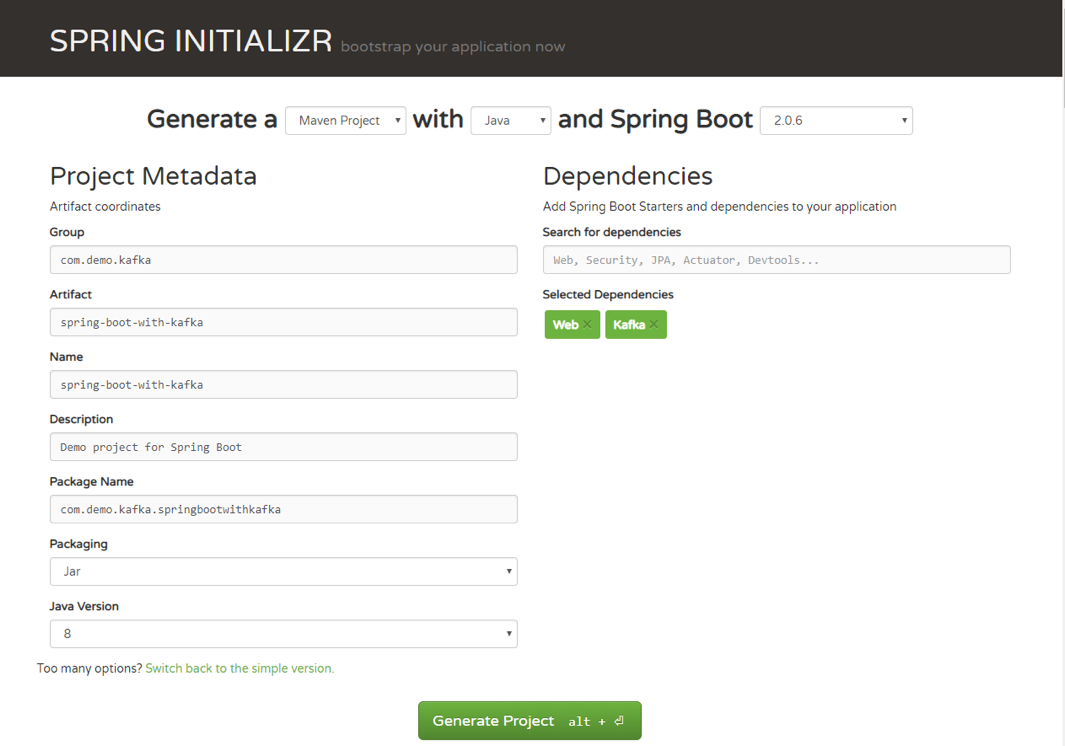
Now, If you have still running Kafka producer (Step #6) in another terminal. Just type some text on that producer terminal. it will immediately visible on consumer terminal. See the below screenshot of Kafka producer and consumer in working:



**Spring boot implementation**

## **Step 1: Generate our project**

First, let’s go to [Spring Initializr](https://start.spring.io/) to generate our project. Our project will have Spring MVC/web support and Apache Kafka support.



Once you have unzipped the project, you’ll have a very simple structure. I’ll show you how the project will look like at the end of this article so you can easily follow the same structure. I’m going to use Intellij IDEA, but you can use any Java IDE.

## **Step 2: Publish/read messages from the Kafka topic**

Now, you can see what it looks like. Let’s move on to publishing/reading messages from the Kafka topic.

Start by creating a simple Java class, which we will use for our example: package com.demo.models;

public class User {

private String name;

private int age;

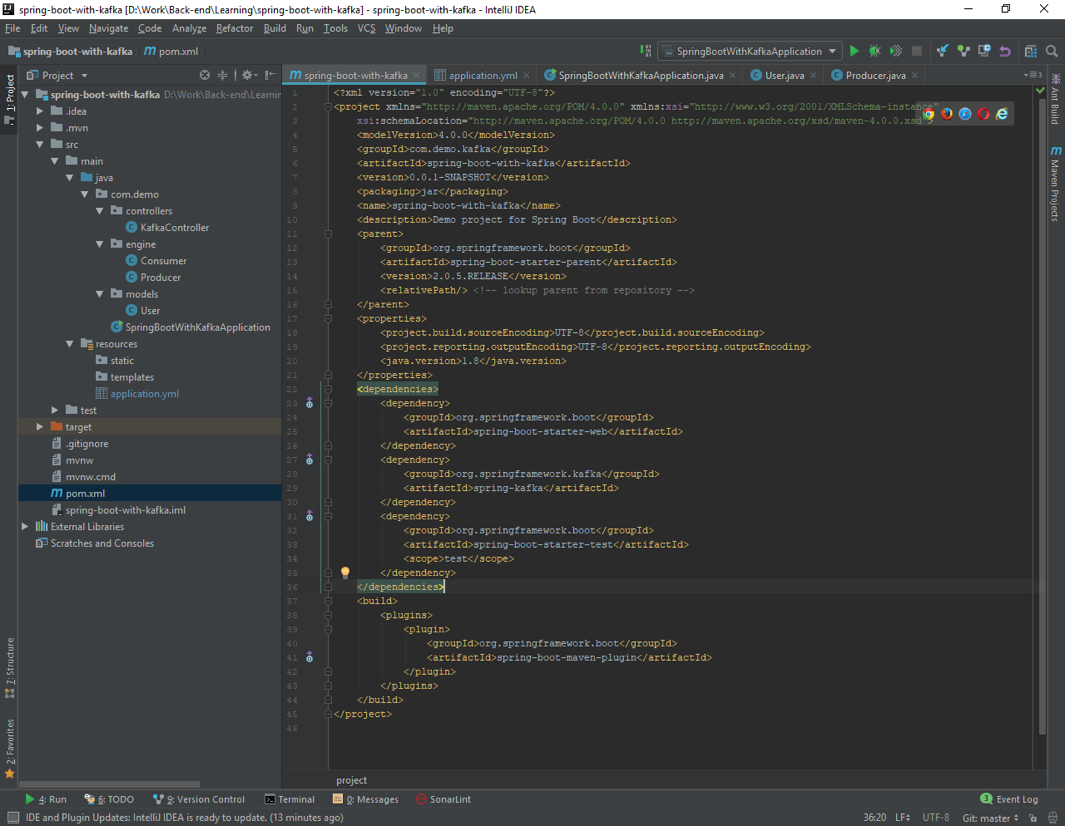
public User(String name, int age) {

this.name = name;

this.age = age;

    }

}



## **Step 3: Configure Kafka through application.yml configuration file**

Next, we need to create the configuration file. We need to somehow configure our Kafka producer and consumer to be able to publish and read messages to and from the topic. Instead of creating a Java class, marking it with @Configuration annotation, we can use either application.properties file or application.yml. Spring Boot allows us to avoid all the boilerplate code we used to write in the past, and provide us with much more intelligent way of configuring our application, like this:

server: port: 9000

spring:

kafka:

consumer:

bootstrap-servers: localhost:9092

group-id: group\_id

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

producer:

bootstrap-servers: localhost:9092

key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

If you want to get more about Spring Boot auto-configuration, you can read this short and useful [article](http://www.springboottutorial.com/spring-boot-application-configuration). For a full list of available configuration properties, you can refer to the official [documentation](https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html).

## **Step 4: Create a producer**

Creating a producer will write our messages to the topic.

@Service

public class Producer {

private static final Logger logger = LoggerFactory.getLogger(Producer.class);

private static final String TOPIC = "users";

@Autowired

private KafkaTemplate<String, String> kafkaTemplate;

public void sendMessage(String message) {

logger.info(String.format("#### -> Producing message -> %s", message));

this.kafkaTemplate.send(TOPIC, message);

}

}

We just auto-wired [KafkaTemplate](https://docs.spring.io/spring-kafka/api/org/springframework/kafka/core/KafkaTemplate.html) and will use this instance to publish messages to the topic—that’s it for producer!

## **Step 5: Create a consumer**

Consumer is  the service that will be responsible for reading messages processing them according to the needs of your own business logic. To set it up, enter the following:

@Service

public class Consumer {

private final Logger logger = LoggerFactory.getLogger(Producer.class);

@KafkaListener(topics = "users", groupId = "group\_id")

public void consume(String message) throws IOException {

logger.info(String.format("#### -> Consumed message -> %s", message));

}

}

Here, we told our method void consume (String message) to subscribe to the user’s topic and just emit every message to the application log. In your real application, you can handle messages the way your business requires you to.

## **Step 6: Create a REST controller**

If we already have a consumer, then we already have all we need to be able to consume Kafka messages.

To fully show how everything that we created works, we need to create a controller with single endpoint. The message will be published to this endpoint, and then handled by our producer.

Then, our consumer will catch and handle it the way we set it up  by logging to the console.

@RestController

@RequestMapping(value = "/kafka")

public class KafkaController {

private final Producer producer;

@Autowired

KafkaController(Producer producer) {

this.producer = producer;

}

@PostMapping(value = "/publish")

public void sendMessageToKafkaTopic(@RequestParam("message") String message) {

this.producer.sendMessage(message);

}

}

Let’s send our message to Kafka using cURL:

curl -X POST -F 'message=test' http://localhost:9000/kafka/publish

