




[DZone \(/\)](#) > [Big Data Zone \(/big-data-analytics-tutorials-tools-news\)](#) > [The Magic of Kafka With Spring Boot](#)

The Magic of Kafka With Spring Boot

(/users/2808752/rahulmlokurte.html) by **Rahul Lokurte** (/users/2808752/rahulmlokurte.html) · Jan. 30, 19 · Big Data[Zone \(/big-data-analytics-tutorials-tools-news\)](#) · Tutorial Like (35)  Comment (3)  Save  Tweet

How to Simplify Development of Modern Applications: Le
JSON and temporal tables can be used to simplify the de
modern applications without sacrificing all of the things w
depend upon: transactions, data integrity and reliability. [\[](#)
[free toolkit!](#)

Apache Kafka is a distributed streaming platform with capabilities such as publishing and subscribing to a stream of records, storing the records in a fault tolerant way, and processing that stream of records.

It is used to build real-time streaming data pipelines, that can perform functionalities such as reliably passing a stream of records from one application to another and processing and transferring the records to the target applications.

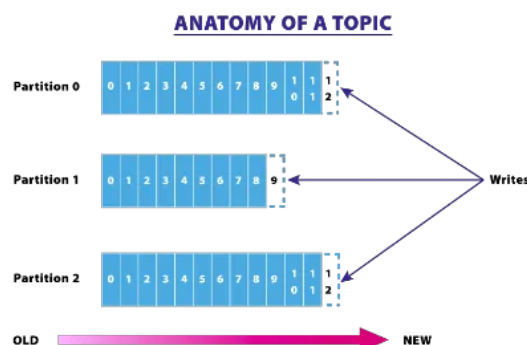
Topics

Kafka is run as a cluster in one or more servers and the cluster stores/retrieves the records in a feed/category called **Topics**. Each record in the topic is stored with a key, value, and timestamp.

The topics can have zero, one, or multiple consumers, who will subscribe to the data written to that topic. In Kafka terms, topics are always part of a **multi-subscriber** feed.

Partitions

The Kafka cluster uses a partitioned log for each topic.



The partition maintains the order in which data was inserted and once the record is published to the topic, it remains there depending on retention (expiry) of records (configurable). The records are always appended at the end of the partitions. It maintains a flag called 'offsets,' which uniquely identifies each record within the partition.

The offset is controlled by the consuming applications. Using offset, consumers might backtrack to older offsets and reprocess the records if needed.

Producers

The stream of records, i.e. data, is published to the topics by the producers. They can also assign the partition when it is publishing data to the topic. The producer can send data in a round robin way or it can implement a priority system based on sending records to certain partitions based on the priority of the record.

Consumers

Consumers consume the records from the topic. They are based on the concept of a consumer-group, where some of the consumers are assigned in the group. The record which is published to the topic is only delivered to one instance of the consumer from one consumer-group. Kafka internally uses a mechanism of consuming records inside the consumer-group. Each instance of the consumer will get hold of the particular partition log, such that within a consumer-group, the records can be processed parallelly by each consumer.

Spring Boot Kafka

Spring provides good support for Kafka and provides the abstraction layers to work with over the native Kafka Java clients.

We can add the below dependencies to get started with Spring Boot and Kafka.

```
1 <dependency>;
2     <groupId>org.springframework.kafka</groupId>;
3     <artifactId>spring-kafka</artifactId>;
4     <version>2.2.3.RELEASE</version>;
5 </dependency>;
```

To download and install Kafka, please refer the official guide <https://kafka.apache.org/quickstart> (<https://kafka.apache.org/quickstart>).

Once you download Kafka, you can issue a command to start [ZooKeeper](https://dzone.com/articles/an-introduction-to-zookeeper-1) (<https://dzone.com/articles/an-introduction-to-zookeeper-1>) which is used by Kafka to store metadata.

```
zookeeper-server-start.bat .\config\zookeeper.properties
```

Next, we need to start the Kafka cluster locally by issuing the below command.

```
kafka-server-start.bat .\config\server.properties
```

Now, by default, the Kafka server starts on `localhost:9092`.

Write a simple REST controller and expose one endpoint, `/publish`, as shown below. It is used to publish the message to the topic.

```
1 package com.rahul.kafkaspringboot.controllers;
2
3 import com.rahul.kafkaspringboot.services.Producer;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.PostMapping;
```



```

5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.RestController;

```



(/users/login.html)



(/search)

REFCARDZ (/refcardz) **RESEARCH** (/research) **WEBINARS** (/webinars) **ZONES** ▾

```

10 @RestController
11 @RequestMapping(value = "/kafka")
12 public class KafkaController {
13
14     private final Producer producer;
15
16     @Autowired
17     public KafkaController(Producer producer) {
18         this.producer = producer;
19     }
20
21     @PostMapping(value = "/publish")
22     public void sendMessageToKafkaTopic(@RequestParam("message") String message){
23         this.producer.sendMessage(message);
24     }
25 }

```

We can then write the producer which uses Spring's `KafkaTemplate` to send the message to a topic named `users`, as shown below.

```

1 package com.rahul.kafkaspringboot.services;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.kafka.core.KafkaTemplate;
7 import org.springframework.stereotype.Service;
8
9 @Service
10 public class Producer {
11
12     private static final Logger logger = LoggerFactory.getLogger(Producer.class);
13     private static final String TOPIC = "users";
14
15     @Autowired
16     private KafkaTemplate<String,String> kafkaTemplate;
17
18     public void sendMessage(String message){
19         logger.info(String.format("$$$ -> Producing message --> %s",message));
20         this.kafkaTemplate.send(TOPIC,message);
21     }
22 }

```

We can also write the consumer as shown below, which consumes the message from the topic `users` and output the logs to the console.

```

1 package com.rahul.kafkaspringboot.services;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.kafka.annotation.KafkaListener;
6 import org.springframework.stereotype.Service;
7
8 @Service
9 public class Consumer {
10
11     private final Logger logger = LoggerFactory.getLogger(Consumer.class);
12
13     @KafkaListener(topics = "users", groupId = "group_id")
14     public void consume(String message){
15         logger.info(String.format("$$$ -> Consumed Message -> %s",message));
16     }
17 }

```

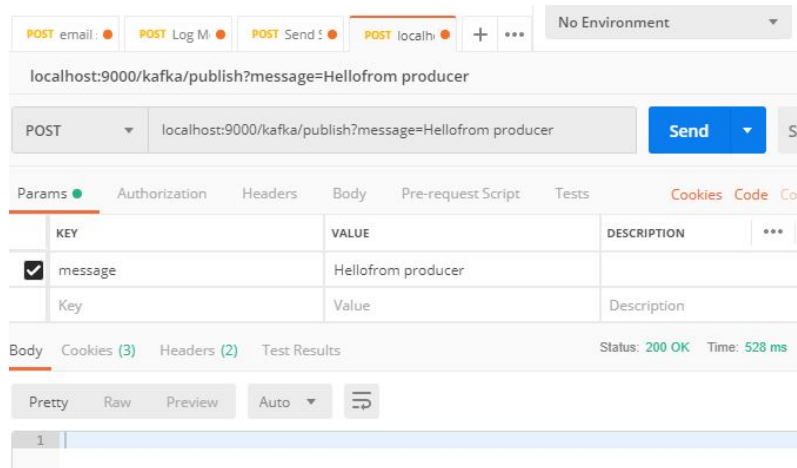
Now, we need a way to publish a message to the Kafka servers and create a topic and publish to it. We can do it using `application.yml` as shown below.

```

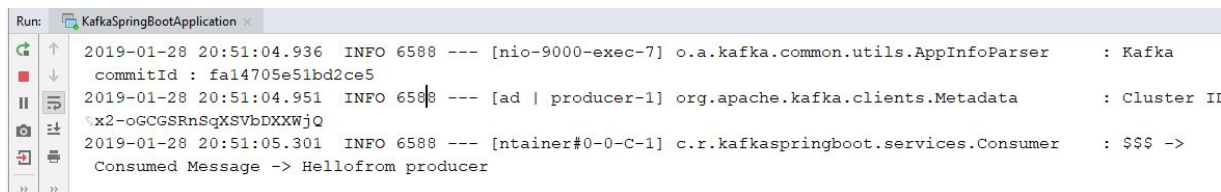
1 server:
2   port: 9000
3
4 spring:
5   kafka:
6     consumer:
7       bootstrap-servers: localhost:9092
8       group-id: group-id
9       auto-offset-reset: earliest
10      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
11      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
12     producer:
13       bootstrap-servers: localhost:9092
14       key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
15       value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

```

Now, if we run the application and hit the endpoint as shown below, we have published a message to the topic.

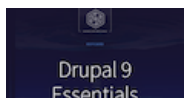


Now, if we check the logs from the console, it should print the message which was sent to the publish endpoint as seen below.



Summary

In this post, we have seen basic terminology used in the Kafka system. We also saw how easy it is to configure Kafka with Spring Boot. Most of the magic is done behind the scenes by Spring Boot. One easy and fast way is configuring Kafka-related details in the `application.yml` file, which is good if we change the Kafka clusters and we have to point the servers to new Kafka cluster address.



Drupal 9 Essentials

This Refcard introduces the core features and illustrates how a developer can use pre-defined, security functionality to create complex data collection and delivery solutions. [Download now](#) ►



REFCARDZ (/refcardz) **RESEARCH** (/research) **WEBINARS** (/webinars) **ZONES** ▾

Topics: SPRING KAFKA, BIG DATA, KAFKA TUTORIAL, SPRING BOOT TUTORIAL

Opinions expressed by DZone contributors are their own.

Big Data Partner Resources

ABOUT US

About DZone (/pages/about)

Send feedback (mailto:support@dzone.com)

Careers (https://devada.com/careers/)

ADVERTISE

Developer Marketing Blog (https://devada.com/blog/developer-marketing)

Advertise with DZone (/pages/advertise)

+1 (919) 238-7100 (tel:+19192387100)

CONTRIBUTE ON DZONE

MVB Program (/pages/mvb)

Zone Leader Program (/pages/zoneleader)

Become a Contributor (/pages/contribute)

Visit the Writers' Zone (/writers-zone)

LEGAL

Terms of Service (/pages/tos)

Privacy Policy (/pages/privacy)

CONTACT US

600 Park Offices Drive

Suite 150

Research Triangle Park, NC 27709

support@dzone.com (mailto:support@dzone.com)

+1 (919) 678-0300 (tel:+19196780300)

Let's be friends:



(https://www.linkedin.com/company/devada-
 (/pages/help-center)/corporate/DZoneInc)

DZone.com is powered by



(https://devada.com/answerhub/)