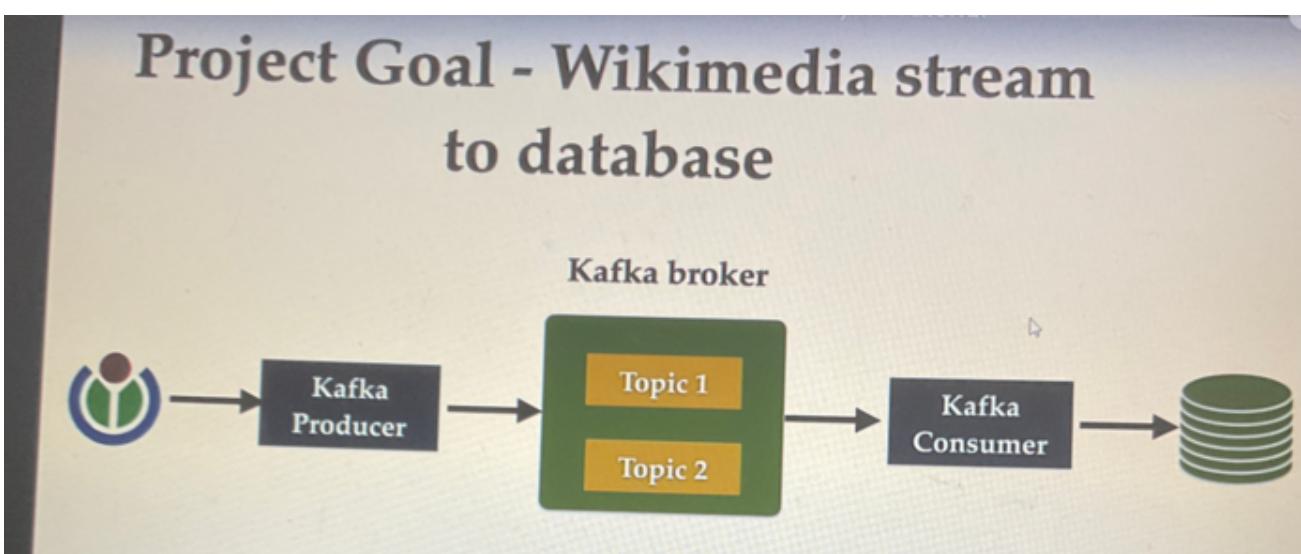


Kafka

Apache Kafka

Apache Kafka is a messaging system used to communicate message between two microservices.



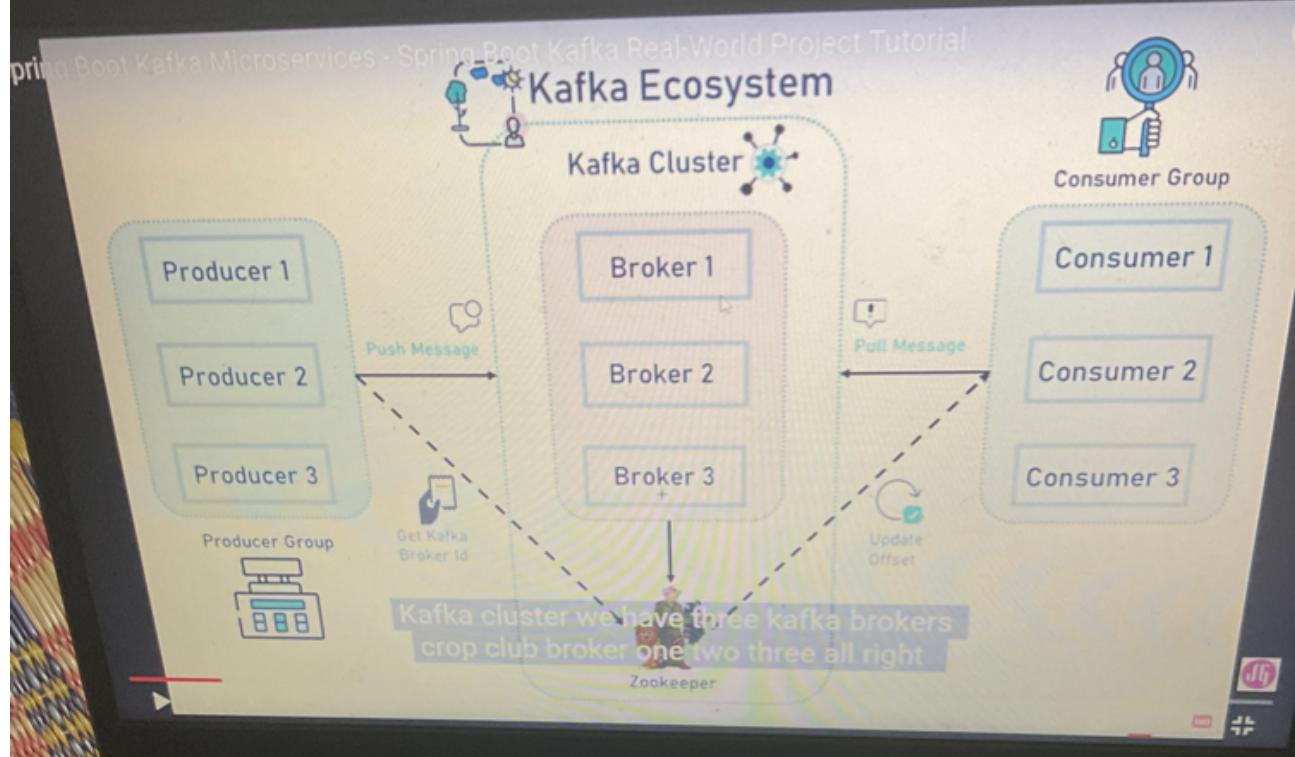
Flow is like It'll consume data

from the source using **Kafka Producer**.

Then after reading the data from **Kafka Producer**, after it'll write the data in Kafka broker.

Then Kafka Consumer - consume the source data from Kafka broker and then then it'll write the database.

[Apache Kafka used for **batch** with **large amount of data**]



Zoo Keepers → It is a Service
 It'll manage the State of all the
 Kafka brokers within a Kafka
 Cluster and It'll also manage the
 Configuration of Topic

Apache Server Run on Port 9092

Dependencies : Spring for Apache Kafka



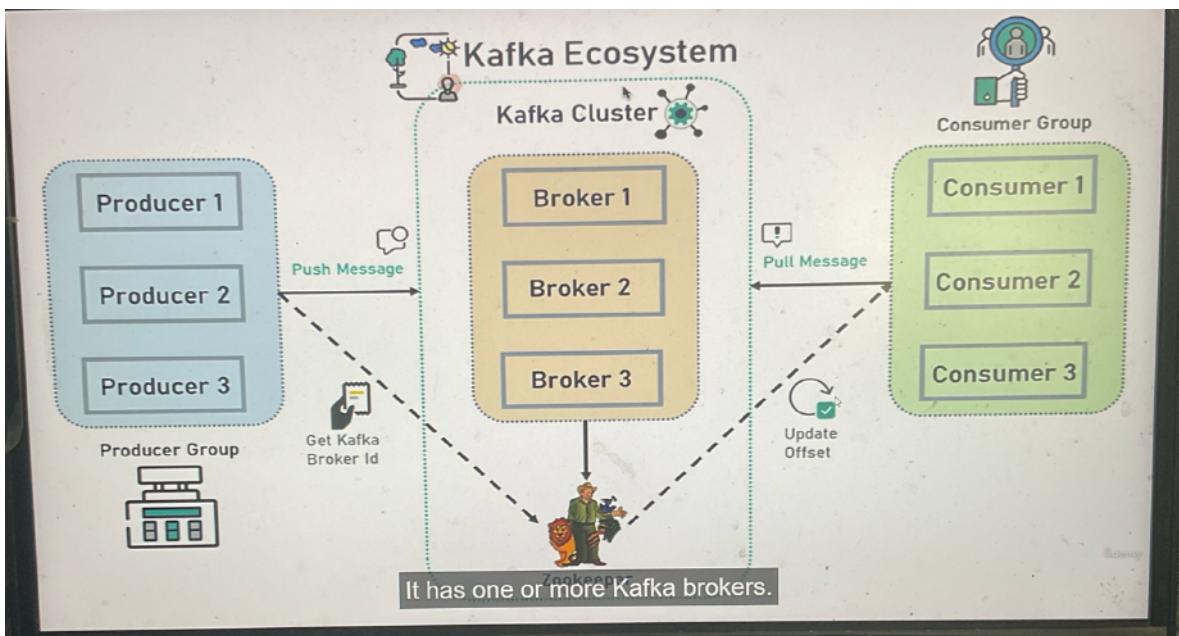
whenever we want to create a

multi model. maven Project we have to

add <Packaging> Pom </Packaging>

Apache Kafka

It built by LinkedIn, now managed by Apache Foundation.



Apache Kafka is distributed System.
So it acts as Kafka Cluster.

Producers: It is nothing but application that produces the message send that messages to the kafka broker.

- Any kind of message \Rightarrow
- event
 - stream of records
 - Stream of data
 - Avro PlainText
 - String Json.

Consumer :

Consumer will basically consume those messages from the Kafka broker.

Kafka Topic :

So basically we create a topic in a Kafka cluster. So that consumer can able to subscribe to that particular topic.

Kafka Cluster. It is nothing but a cluster
It has one (or) more kafka brokers.

Zoo keeper :

Zoo keeper can manages The State
of all the kafka brokers.

- Zookeeper maintains the state of all the Kafka brokers in a Kafka cluster.
- As well as, Zookeeper managing the Configuration of all the topics Producer and Consumer.

Apache Kafka is Full tolerance. means

If one or a broker goes down, then it

can have capability to manage the other nodes

coz the data is distributed among different Kafka brokers.

- The data can be stored in apache kafka at any duration. So the consumer can able to consume the history data from the Apache broker

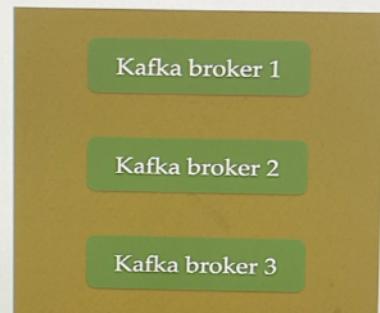
Apache Kafka Core Concepts:

Kafka Cluster

Since Kafka is a distributed system, it acts as a cluster.

A Kafka cluster consists of a set of brokers. A cluster has a minimum of 3 brokers.

Kafka Cluster

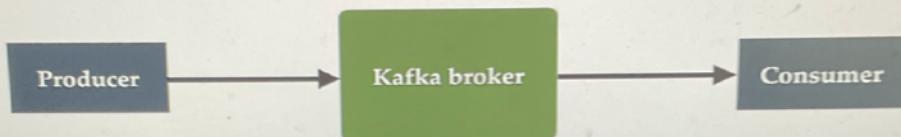


Well, as we know that Kafka is a distributed system, right?

Kafka Broker

The broker is the Kafka server. It's just a meaningful name given to the Kafka server. And this name makes sense as well because all that Kafka does is act as a message broker between producer and consumer.

The producer and consumer don't interact directly. They use Kafka server as an agent or a broker to exchange messages.



Producer

Producer is an application that sends messages. It does not send messages directly to the recipient. It send messages only to the Kafka server.

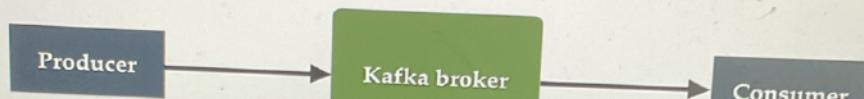


Just remember, Producer is nothing but an application that produces the messages and sends them to the Kafka

Consumer

Consumer is an application that reads messages from the Kafka server.

If producers are sending data, they must be sending it to someone, right? The consumers are the recipients. But remember that the producers don't send data to a recipient address. They just send it to Kafka server. And anyone who is interested in that data can come forward and take it from Kafka server. So, any application that requests data from a Kafka server is a consumer, and they can ask for data sent by any producer provided they have permissions to read it.

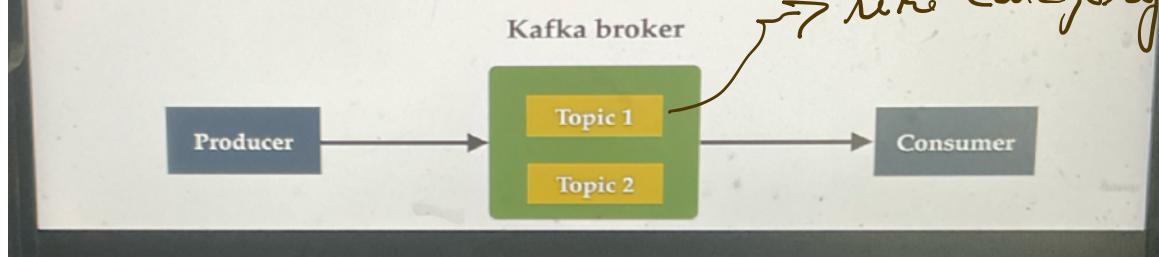


So any application that requests data from the server is a Consumer and they can ask for data sent by

Kafka Topic

We learned that producer sends data to the Kafka broker. Then a consumer can ask for data from the Kafka broker. But the question is, Which data? We need to have some identification mechanism to request data from a broker. There comes the notion of the topic.

- Topic is like a table in database or folder in a file system.
- Topic is identified by a name.
- You can have any number of topics.

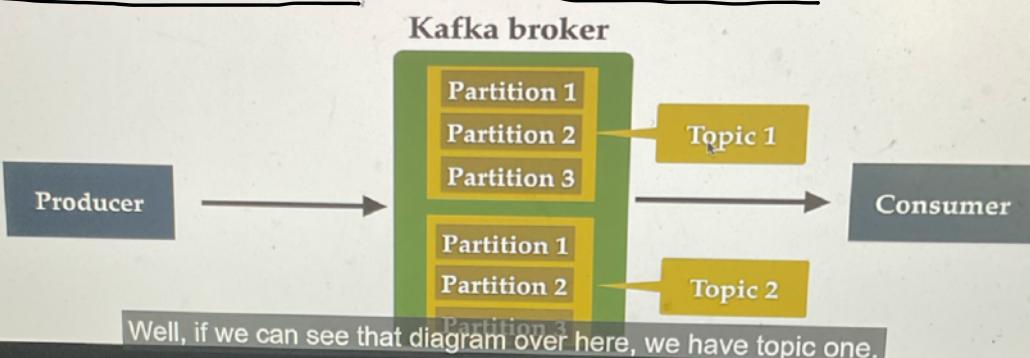


Topic = Category : (Categorize message to the Kafka broker) . query unpanni message access panha multiqathu like db query.

Kafka Partitions

Kafka topics are divided into a number of partitions, which contain records in an unchangeable sequence.

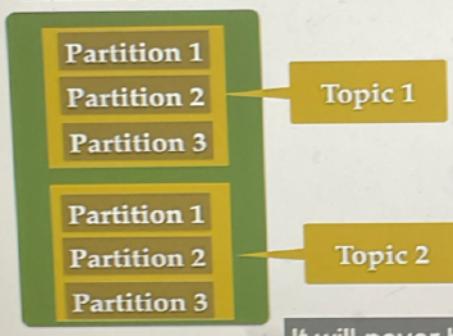
Kafka Brokers will store messages for a topic. But the capacity of data can be enormous and it may not be possible to store in a single computer. Therefore it will be partitioned into multiple parts and distributed among multiple computers, since Kafka is a distributed system.



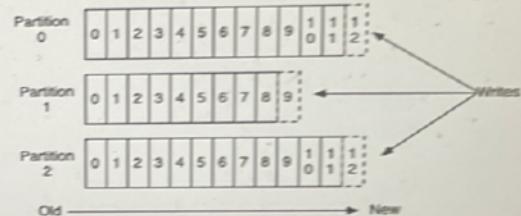
Offsets

Offset is a sequence of ids given to messages as they arrive at a partition. Once the offset is assigned it will never be changed. The first message gets an offset zero. The next message receives an offset one and so on.

Kafka broker

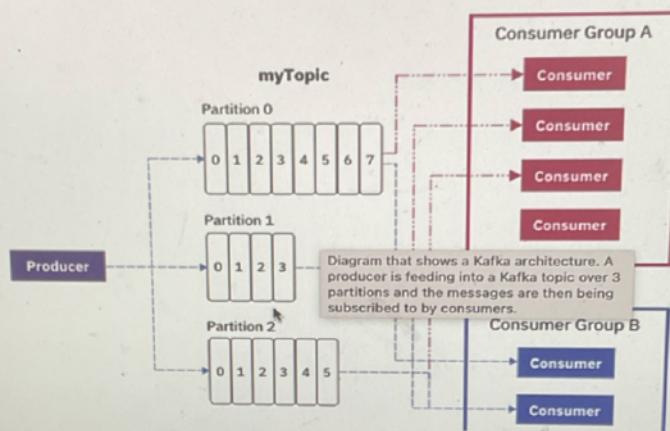


Anatomy of a Topic



Consumer Groups

A consumer group contains one or more consumers working together to process the messages.



Kafka Installation:

Zoo Keeper:

```
D:\installs>cd kafka  
D:\installs\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

```
8 direct buffers, {org.apache.zookeeper.server.NIOServerCnxnFactory}  
[2024-02-01 19:17:52,710] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)  
[2024-02-01 19:17:52,762] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)  
[2024-02-01 19:17:52,763] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
```

means Zoo Keeper Started.

Server

```
PS D:\> cd .\installs\  
PS D:\installs> cd .\kafka\  
PS D:\installs\kafka> .\bin\windows\kafka-server-start.bat .\config\server.properties
```

```
[2024-02-01 19:24:51,938] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)  
[2024-02-01 19:24:51,978] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafka.network.SocketServer)  
[2024-02-01 19:24:51,988] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)  
[2024-02-01 19:24:52,008] INFO Kafka version: 3.6.1 (org.apache.kafka.common.utils.AppInfoParser)  
[2024-02-01 19:24:52,009] INFO Kafka commitId: 5e3c2b738d253ff5 (org.apache.kafka.common.utils.AppInfoParser)  
[2024-02-01 19:24:52,017] INFO Kafka startTimeMs: 1706795691997 (org.apache.kafka.common.utils.AppInfoParser)  
[2024-02-01 19:24:52,097] INFO [kafka-broker-0] started (kafka.server.KafkaServer)
```

means Server Started.

Prefor github textfile => for Commands

→ Create topic

→ read Event

→ Create Event

Dependency :

Spring for Apache Kafka

- without Springboot we have to do lot configuration for Kafka.

Configure Kafka Producer and Consumers.

The screenshot shows the IntelliJ IDEA interface with the file `application.properties` open. The code defines Kafka consumer and producer configurations:

```
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: myGroup
spring.kafka.consumer.auto-offset-reset: earliest
spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

A handwritten note on the right side of the code explains the configuration:

[Configure all the Kafka servers] → we need to provide consumer group ID which [consumes] to [the] [topic]

The bottom part of the screenshot shows the terminal output of the application running:

```
main n.j.s.SpringbootKafkaTutorialApplication : Starting SpringbootKafkaTutorialApplication using ...
main] n.j.s.SpringbootKafkaTutorialApplication : No active profile set, falling back to 1 default profile
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.62]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context root
main] n.j.s.SpringbootKafkaTutorialApplication : Started SpringbootKafkaTutorialApplication in 1.132
```

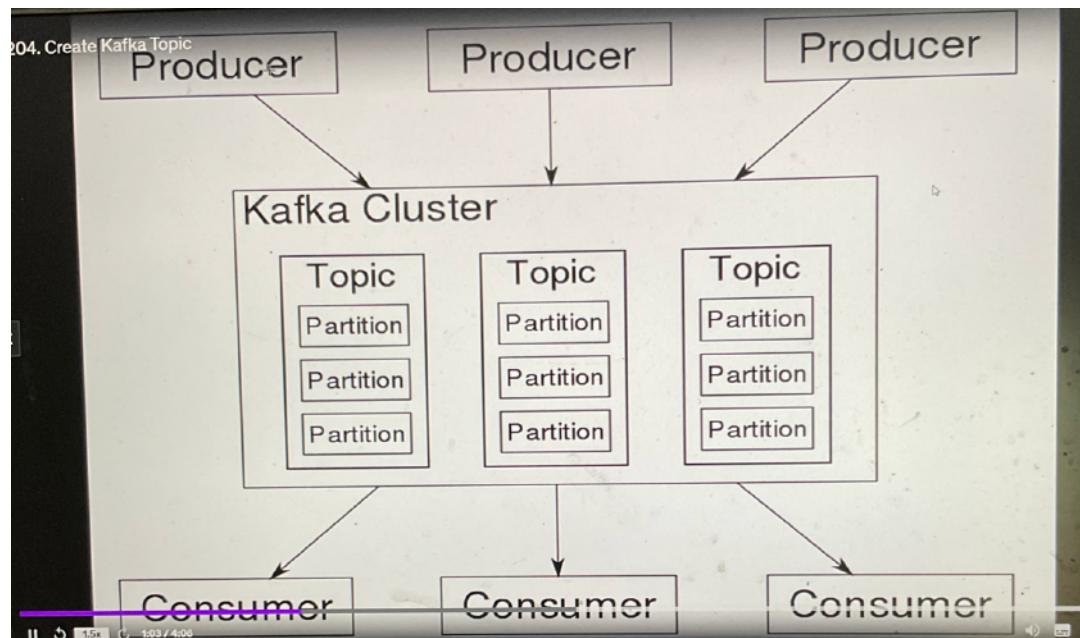
All right, great.

→ It will automatically set the offset to the earliest offset.

6th : Product Port number

F^n Serializer } Serializal Value and Key in a
Serializer message .

Create Kafka topic



Create kafka Config Package . In That Package write one class and Inside one method and annotate with @Bean.

```
javaguides > springboot > config > KafkaTopicConfig > javaguidesTopic > SpringbootKafkaTutorialApplication > application.properties > KafkaTopicConfig.java
```

Project

springboot-kafka-tutorial ~ / D

- .idea
- .mvn
- src
 - main
 - java
 - net.javaguides.springboot
 - config
 - KafkaTopicConfig
 - resources
 - static
 - templates
 - application.properties
- test
- target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

External Libraries

Scratches and Consoles

```
package net.javaguides.springboot.config;  
import org.apache.kafka.clients.admin.NewTopic;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.kafka.config.TopicBuilder;  
  
@Configuration  
public class KafkaTopicConfig {  
  
    @Bean  
    public NewTopic javaguidesTopic(){  
        return TopicBuilder.name("javaguides")  
            .build();  
    }  
}
```

Structure

Bookmarks

Version Control Run TODO Problems Terminal Build Dependencies

Build completed successfully in 798 ms (moments ago)

11:10 LF UTF-8

Create Kafka Producer:

Create Kafka Producer to write a message to the topic.

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service
public class KafkaProducer {

    private static final Logger LOGGER = LoggerFactory.getLogger(KafkaProducer.class);

    private KafkaTemplate<String, String> kafkaTemplate;

    public KafkaProducer(KafkaTemplate<String, String> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    public void sendMessage(String message) {
        LOGGER.info(String.format("Message sent %s", message));
        kafkaTemplate.send("javaguides", message);
    }
}

```

So in this lecture, basically we have created a Capcom. which used the Kafka template to send a message

Kafka Producer :

```

import org.apache.naming.java.JndiURLConnectionFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service
public class KafkaProducer {

    private static final Logger LOGGER = LoggerFactory.getLogger(KafkaProducer.class);

    private KafkaTemplate<String, String> template;

    public KafkaProducer(KafkaTemplate<String, String> template) {
        this.template = template;
    }

    public void sendMessage(String message) {
        LOGGER.info(String.format("Message sent %s", message));
        template.send("floppy", message);
    }
}

```

→ string format

Kafka Consumer :

```

package net.floppy.springbootkafka.kafka;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class KafkaConsumer {

    private static final Logger LOGGER = LoggerFactory.getLogger(KafkaConsumer.class);

    @KafkaListener(topics = "floppy", groupId = "myGroup")
    public void consumer(String message) { LOGGER.info(String.format("Message received : %s", message)); }
}

```

JSON Serializer & Deserializer

Configure Kafka Producer and Consumer for JSON Serializer and Deserializer

How to send and receive a Java Object as a JSON byte[] to and from Apache Kafka.

Apache Kafka stores and transports byte[]. There are a number of built-in serializers and deserializers but it doesn't include any for JSON. Spring Kafka created a JsonSerializer and JsonDeserializer which we can use to convert Java Objects to and from JSON.

We'll send a Java Object as JSON byte[] to a Kafka Topic using a **JsonSerializer**. Afterward, we'll configure how to receive a JSON byte[] and automatically convert it to a Java Object using a **JsonDeserializer**.

you know, it provides JsonSerializer and JsonDeserializer

The screenshot shows the IntelliJ IDEA interface with the file `application.properties` open. The code defines configurations for both Kafka consumer and producer. For the consumer, it sets bootstrap servers to `localhost:9092`, group ID to `myGroup`, and auto-offset-reset to `earliest`. It also specifies the key-deserializer as `org.apache.kafka.common.serialization.StringDeserializer` and the value-deserializer as `org.springframework.kafka.support.serializer.JsonDeserializer`. For the producer, it sets bootstrap servers to `localhost:9092` and uses the same string and json deserializers for both key and value. A note at the bottom states: "It means that Kafka Consumer can JsonDeserializer all the classes from this package."

```
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: myGroup
spring.kafka.consumer.auto-offset-reset: earliest
spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
#spring.kafka.consumer.value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=>

spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer
# spring.kafka.producer.value-serializer: org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```

It means that Kafka Consumer can JsonDeserializer all the classes from this package.

Difference between Synchronous and Asynchronous Communication

RS

Rahul - Lecture 210 · 1 month ago

0

Hey Ramesh, I want to know when to use Synchronous and Asynchronous Communication, like we learnt in starting about Rest Template and WebClient to send DTOs from one service to another. And here also we will be sending something similar I guess. Can you help me with the difference like, when do we use what type of communication, which type is preferred over other, if we have to only send DTOs?

2 replies

Follow replies



Ramesh — Instructor

1 month ago

0

Synchronous Communication:

- Blocking operations; the sender waits for a response.
- Used when immediate feedback is needed, such as CRUD operations in web services.
- Preferred for simple request-response interactions.

Asynchronous Communication:

- Non-blocking; the sender doesn't wait for a response.
- Ideal for long-running operations or when the response isn't immediately needed.
- Better for scalability and handling high-volume traffic.

Choose synchronous for direct, immediate interactions like form submissions or data fetches. Use asynchronous for background tasks, notifications, or when dealing with operations that don't require an instant response.

JSON Producer

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "kafka" and contains a "src" directory with "main" and "test" sub-directories. "main" contains "java" and "resources" sub-directories. "java" contains "net.flopypyspringbootkafka" and "kafka" packages. "kafka" contains "JsonKafkaProducer", "KafkaConsumer", "KafkaProducer", and "MessageController" classes. "resources" contains "static" and "templates" folders and an "application.properties" file.
- Code Editor:** The code editor shows the "JsonKafkaProducer.java" file. It imports Kafka-related classes and defines a service class "JsonKafkaProducer". It includes a method "sendMessage" that logs the message sent and creates a Kafka message using a template and a builder.
- Terminal:** The bottom status bar indicates "Build completed successfully in 6 sec, 678 ms (moments ago)".

It is not a good manner to use hardcoded
String in Program.

So,

```
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service
public class KafkaProducer {

    @Value("${spring.kafka.topic.name}")
    private String topicName;

    private static final Logger LOGGER = LoggerFactory.getLogger(KafkaProducer.class);

    private KafkaTemplate<String, String> kafkaTemplate;

    public KafkaProducer(KafkaTemplate<String, String> kafkaTemplate) { this.kafkaTemplate = kafkaTemplate; }

    public void sendMessage(String message){
        LOGGER.info(String.format("Message sent %s", message));
        kafkaTemplate.send(topicName, message);
    }
}
```

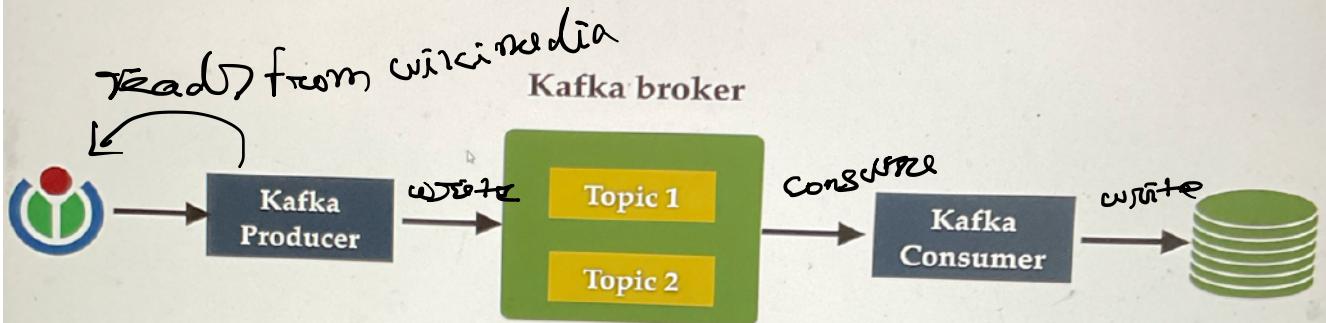
Apart from that, we can also change capture consumer.

```
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: myGroup
spring.kafka.consumer.auto-offset-reset: earliest
spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
#spring.kafka.consumer.value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer
# spring.kafka.producer.value-serializer: org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.topic.name=javaguides
spring.kafka.topic.json.name=javaguides_json
```

We don't have to use both!

Real World Project

Project Goal - Wikimedia stream to database

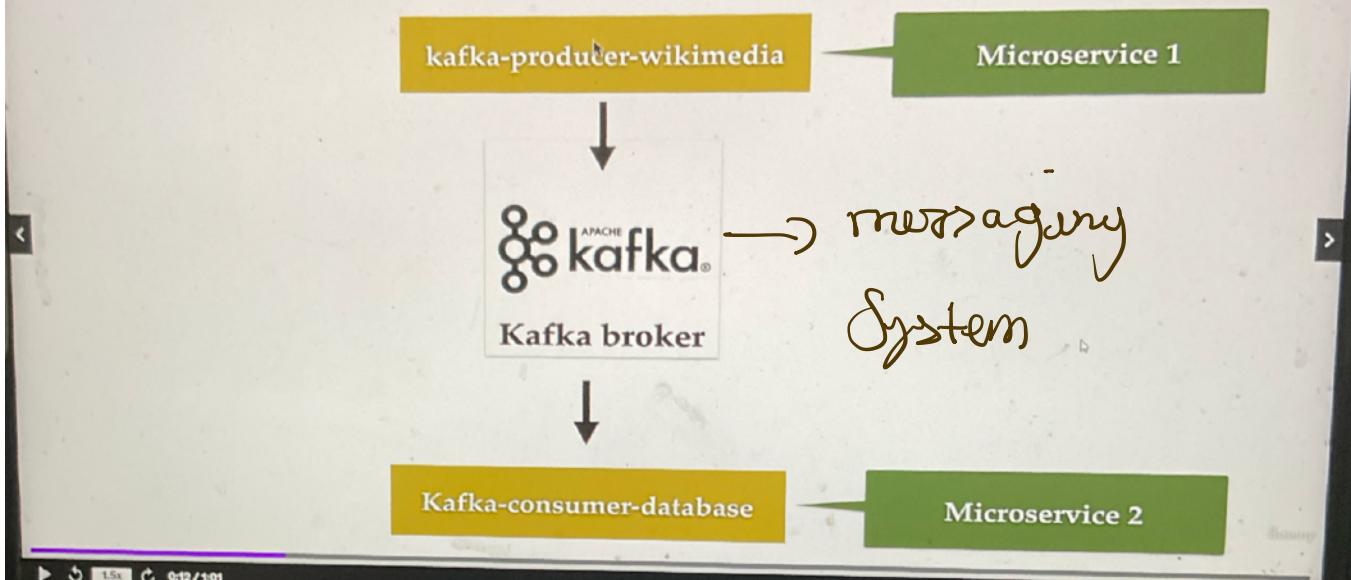


Link: <https://stream.wikimedia.org/v2/stream/recentchange>

Well, basically we're going to read a huge amount of real-time stream data from the Wikimedia tool.

217. Spring Boot Kafka Project Setup - Create Two Microservices

Spring Boot Kafka Project Setup



Wikimedia

Producers Setup

To make multi module maven Project.

The necessary to make , first Project as

Parent Project. To make that . . .

```
<?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">
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.7</version>
        <relativePath/> <!-- lookup parent from repository --&gt;
    &lt;/parent&gt;
    &lt;groupId&gt;net.javaguides&lt;/groupId&gt;
    &lt;artifactId&gt;springboot-kafka-real-world-project&lt;/artifactId&gt;
    &lt;version&gt;0.0.1-SNAPSHOT&lt;/version&gt;
    &lt;name&gt;springboot-kafka-real-world-project&lt;/name&gt;
    &lt;description&gt;Demo project for Spring Boot and Kafka&lt;/description&gt;
    &lt;packaging&gt;pom&lt;/packaging&gt; → Now It is Parent Project.
    &lt;properties&gt;
        &lt;java.version&gt;11&lt;/java.version&gt;
    &lt;/properties&gt;
    &lt;dependencies&gt;
        &lt;dependency&gt;
            &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
            &lt;artifactId&gt;spring-boot-starter&lt;/artifactId&gt;
        &lt;/dependency&gt;
    &lt;/dependencies&gt;
</pre>

a bomb for a parent project.


```

Create Sub module :

Created on DWT Own, we need to create

```
springboot-kafka-real-world-project/kafka-producer-wikimedia/src/main/java/net/javaguides.springboot
  package net.javaguides.springboot;
  import org.springframework.boot.SpringApplication;
  import org.springframework.boot.autoconfigure.SpringBootApplication;
  @SpringBootApplication
  public class SpringBootProducerApplication {
      public static void main(String[] args) {
          SpringApplication.run(SpringBootProducerApplication.class);
      }
  }
```

Okay, so if you can read this book project, it should work.

```
<?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>springboot-kafka-real-world-project</artifactId>
        <groupId>net.javaguides</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <packaging>jar</packaging>
    <artifactId>kafka-producer-wikimedia</artifactId>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

So let's do a packaging as a jar for this model and save it and click on this load my engineers and

→ check jar

```
package net.javaguides.springboot.entity;
import lombok.Getter;
import lombok.Setter;
import javax.persistence.*;
@Entity
@Table(name = "wikimedia_recentchange")
@Getter
@Setter
public class WikimediaData {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Lob
    private String wikiEventData;
}
```

So go to main package, right click new and then choose package.

In order to accept large amount of data. we use @Lob

[action] Event Driven Architecture

What is event-driven architecture?

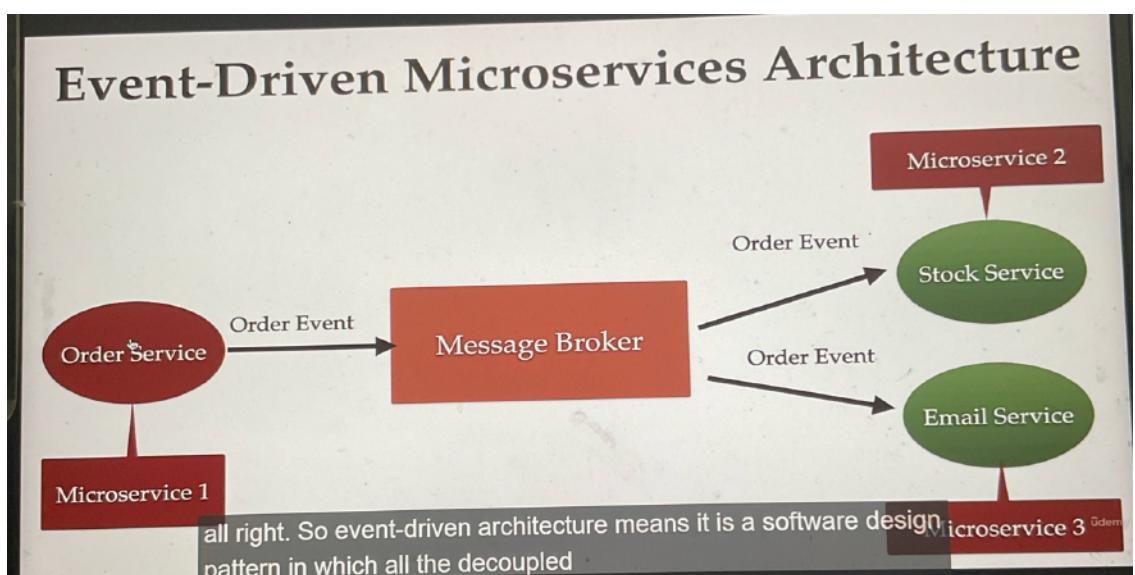
Event-driven architecture (EDA) is a software design pattern in which decoupled applications can asynchronously publish and subscribe to events via an event broker/message broker.

In an Event-Driven Architecture, applications communicate with each other by sending and/or receiving events or messages

Event-driven architecture is often referred to as "asynchronous" communication.

Event-driven apps can be created in any programming language because event-driven is a programming approach, not a language.

An event-driven architecture is loosely coupled.



whenever customer place an order, the order service will basically create an order event and it will publish that order event to the message broker and stock service

and email service. So there are the basically consumers who subscribed to this message broker. So as soon as order service published order event to the message broker, then two microservice will receive that order stock service: responsible for receiver the order event from message broker. and it will update the order details in the database.

Email Service: responsible for receiver order and send mail to customer.

Advantages

- * Improved Flexibility and Maintainability.
- * High Scalability.
- * Improved Availability.

```
private OrderProducer orderProducer;

public OrderController(OrderProducer orderProducer) {
    this.orderProducer = orderProducer;
}

@PostMapping("/orders")
public String placeOrder(@RequestBody Order order){
    order.setOrderId(UUID.randomUUID().toString());
}

}
```

to generate random unique Id.