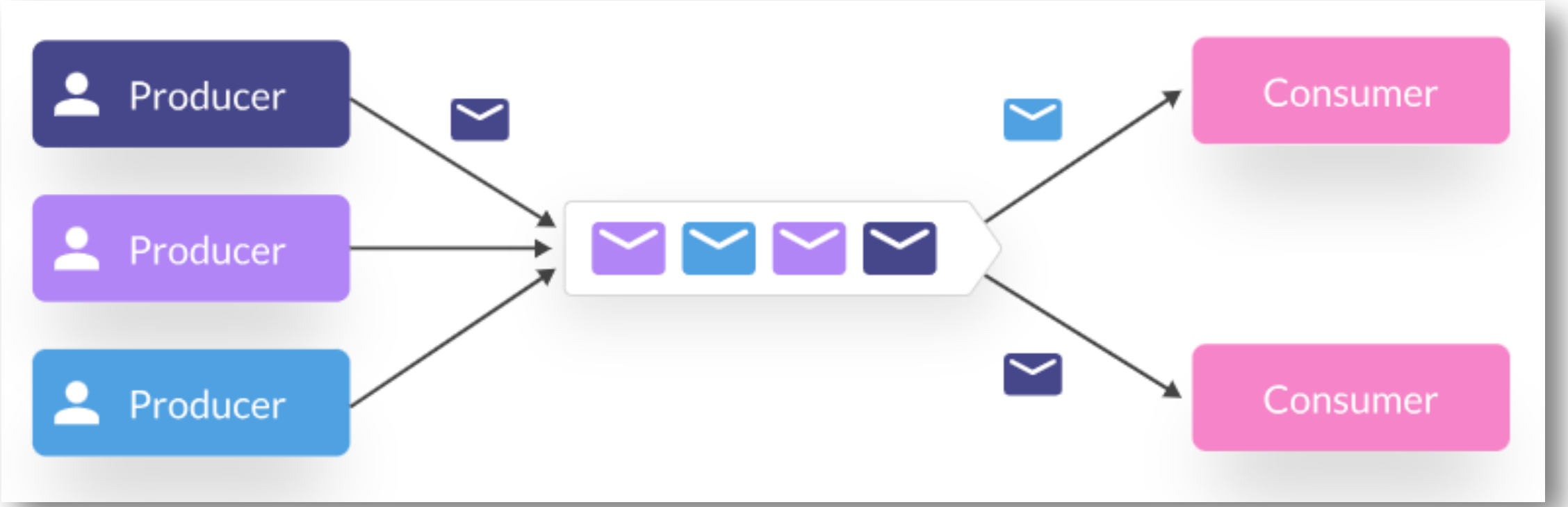
# Apache kafka

An open-source distributed event streaming platform

# Kafka

- Apache Kafka is an open-source distributed event streaming platform developed by the Apache Software Foundation.
- A distributed event streaming platform is a system designed to handle and process a continuous flow of data (events) across multiple servers or nodes.
- Kafka was originally created by LinkedIn and later open-sourced in 2011.

# Event Streaming Platform



kafka.apache.org

kafka

GET STARTED | DOCS | POWERED BY | COMMUNITY | APACHE | DOWNLOAD KAFKA

# APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

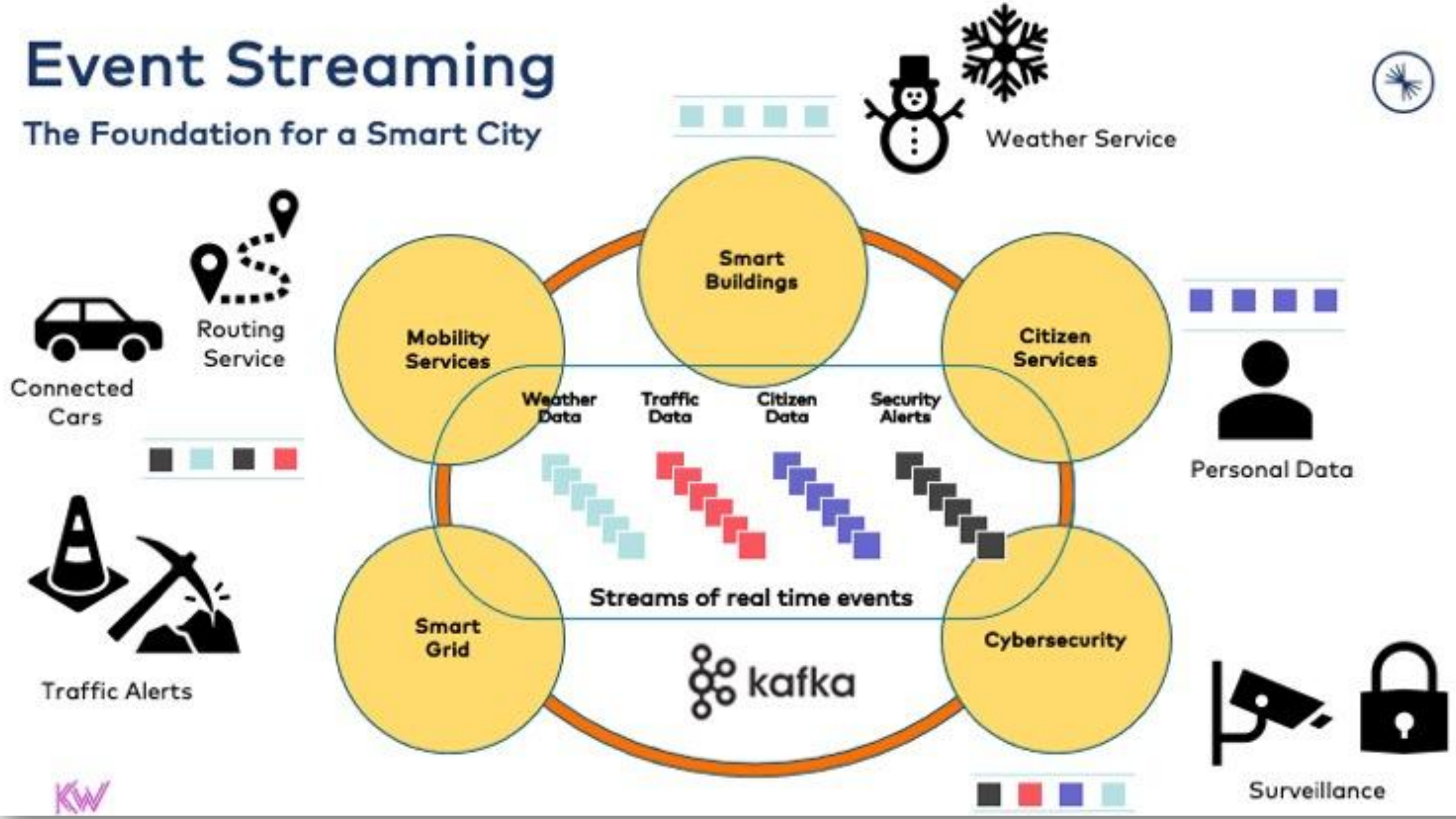
| 10 OUT OF 10 | 7 OUT OF 10 | 10 OUT OF 10 | 8 OUT OF 10 |
|---|---|---|---|
| MANUFACTURING | BANKS | INSURANCE | TELECOM |

# Key Points about Kafka

- **Event Streaming:** Kafka is designed to handle real-time data feeds. It allows you to publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- **Scalability:** Kafka can scale horizontally by distributing data across multiple servers (called brokers). This makes it capable of handling large volumes of data.
- **Fault Tolerance:** Data in Kafka is replicated across multiple brokers, ensuring that it remains available even if some brokers fail.
- **High Throughput and Low Latency:** Kafka is optimized for high throughput and low latency, making it suitable for applications that require real-time data processing.

# Event Streaming

## The Foundation for a Smart City

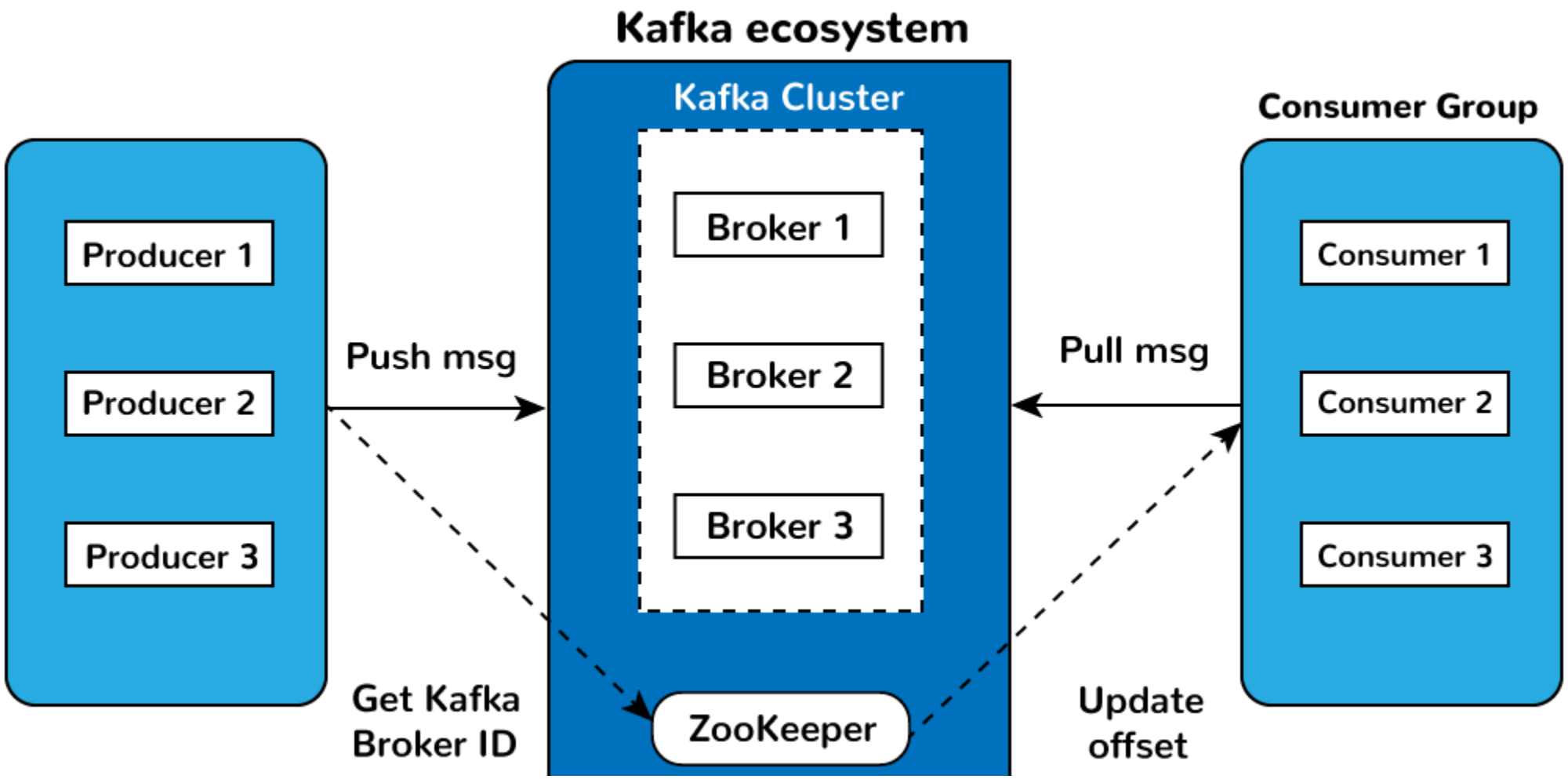


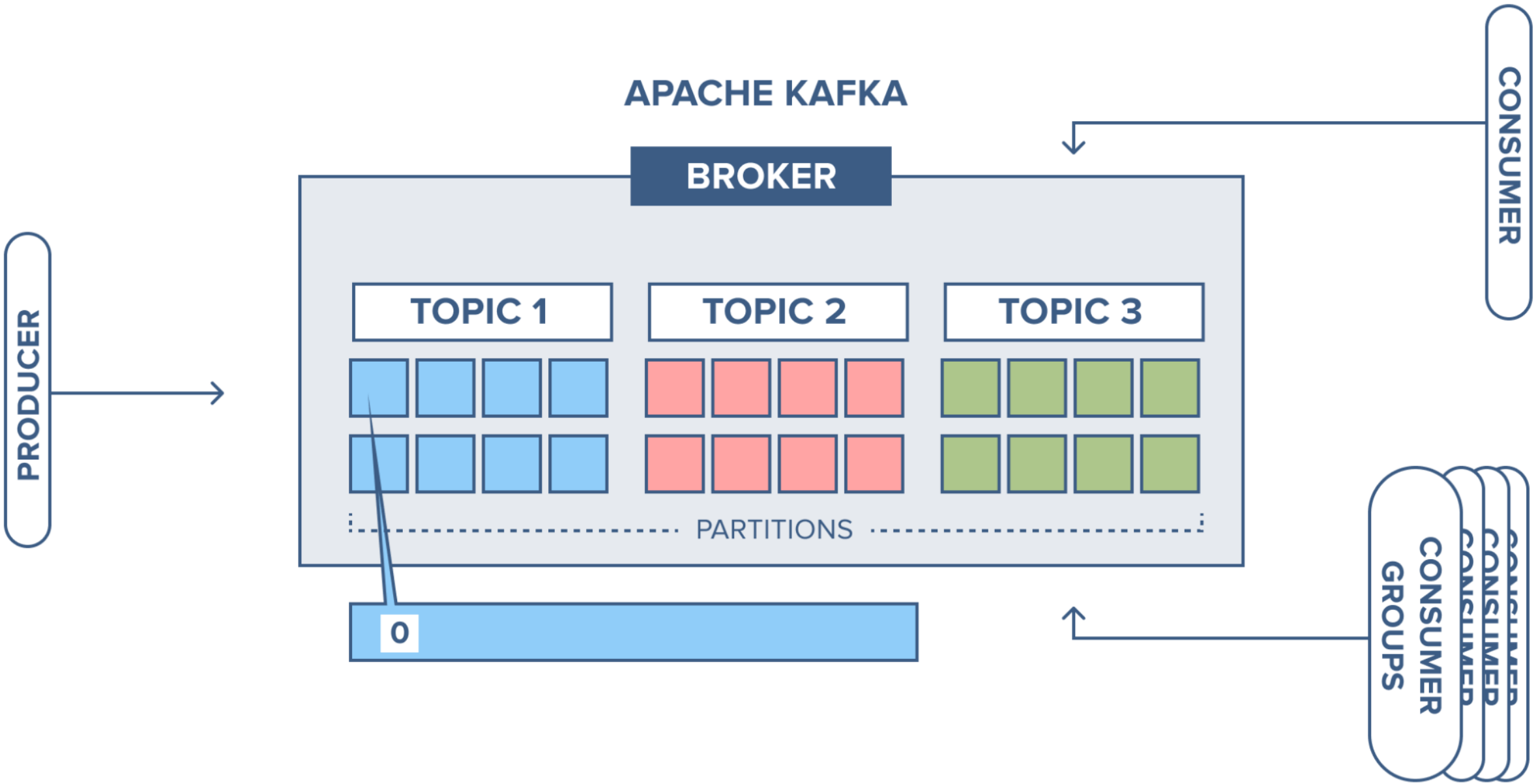
# Kafka Use Cases:

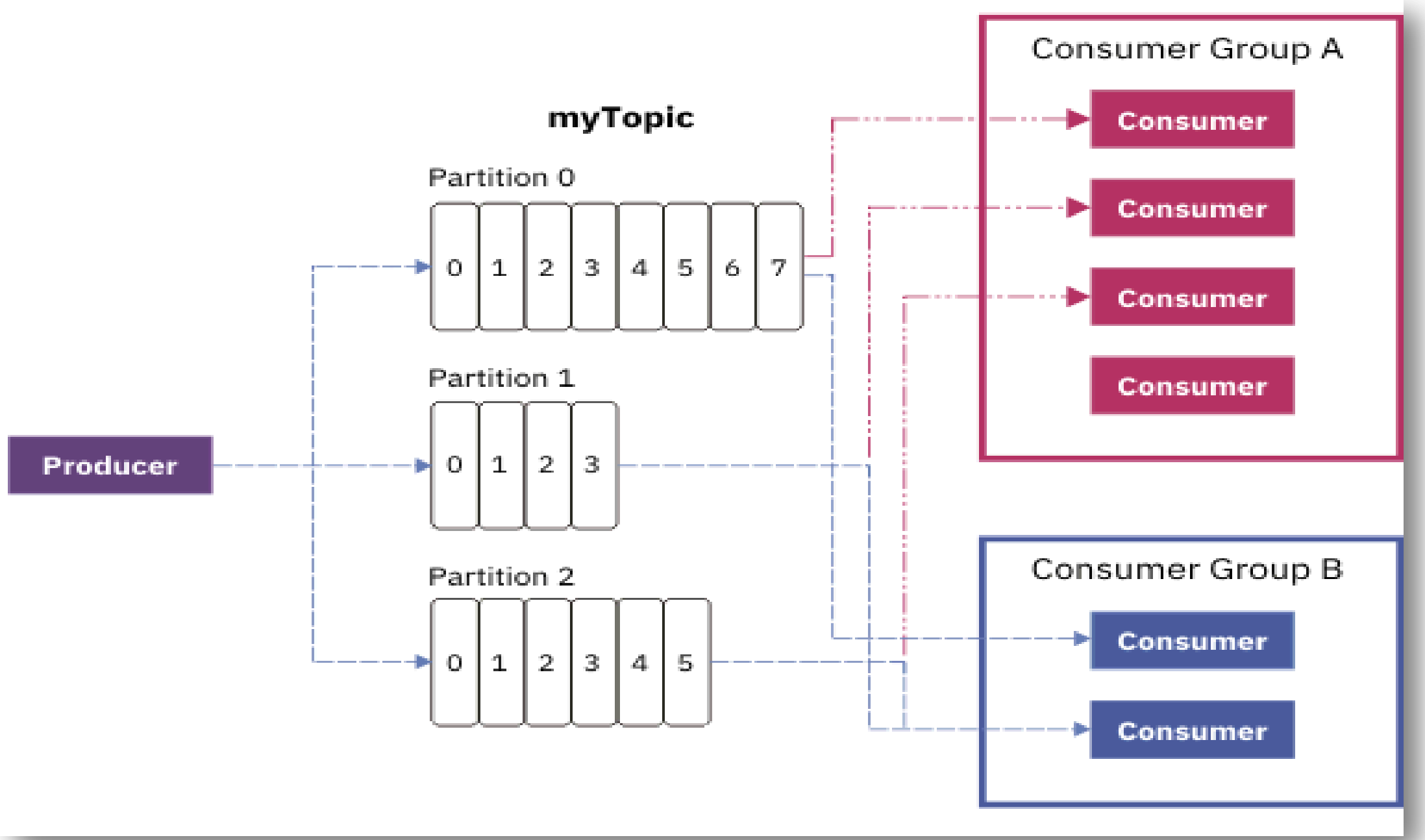
- **Real-Time Analytics**: Companies use event streaming to analyze data in real-time for insights into customer behavior, website traffic, and sales trends. For example, e-commerce platforms can track user interactions to provide personalized recommendations instantly.
- **Fraud Detection:** Financial institutions use event streaming to monitor transactions in real-time and detect fraudulent activities. By analyzing patterns and anomalies as they occur, they can prevent fraud more effectively.
- **IoT Data Processing:** In the Internet of Things (IoT), event streaming platforms process data from sensors and devices in real-time. This is crucial for applications like smart home systems, industrial monitoring, and telematics.
- **Real-Time Stock Trading:** Stock exchanges and trading platforms use event streaming to process and analyze market data in real-time. This allows traders to make informed decisions based on the latest market conditions.

# Key Components

- Producers
- Consumers
- Consumer Groups
- Brokers
- Topic
- Partition
- Offset
- Cluster
- Zookeeper


Kafka ecosystem
Kafka Cluster
Consumer Group
Producer 1
Producer 2
Producer 3
Broker 1
Broker 2
Broker 3
Consumer 1
Consumer 2
Consumer 3
Push msg
Pull msg
Get Kafka
Broker ID
ZooKeeper
Update
offset


APACHE KAFKA
BROKER
TOPIC 1
TOPIC 2
TOPIC 3
PARTITIONS
0
PRODUCER
CONSUMER
CONSUMER GROUPS


myTopic
Partition 0
0 1 2 3 4 5 6 7
Partition 1
0 1 2 3
Partition 2
0 1 2 3 4 5
Producer
Consumer Group A
Consumer
Consumer
Consumer
Consumer
Consumer Group B
Consumer
Consumer

# Key Components

- **Producers:** Producers are clients that publish (write) records to Kafka topics. They are responsible for choosing which partition to send records to.
- **Consumers:** Consumers are clients that subscribe to Kafka topics and process the records. Each consumer reads data from one or more partitions.
- **Consumer Groups:** A consumer group is a group of consumers that work together to consume records from a topic. Each record is delivered to one consumer in the group.

# Key Components

- **Brokers**: Kafka brokers are servers that store and manage the data. Each broker is identified by a unique ID and can handle hundreds of thousands of reads and writes per second.
- **Clusters**: A Kafka cluster is a collection of one or more Kafka brokers working together.
- **ZooKeeper**: ZooKeeper is used to manage and coordinate Kafka brokers. It handles tasks such as leader election for partitions and configuration management.

# Key Components

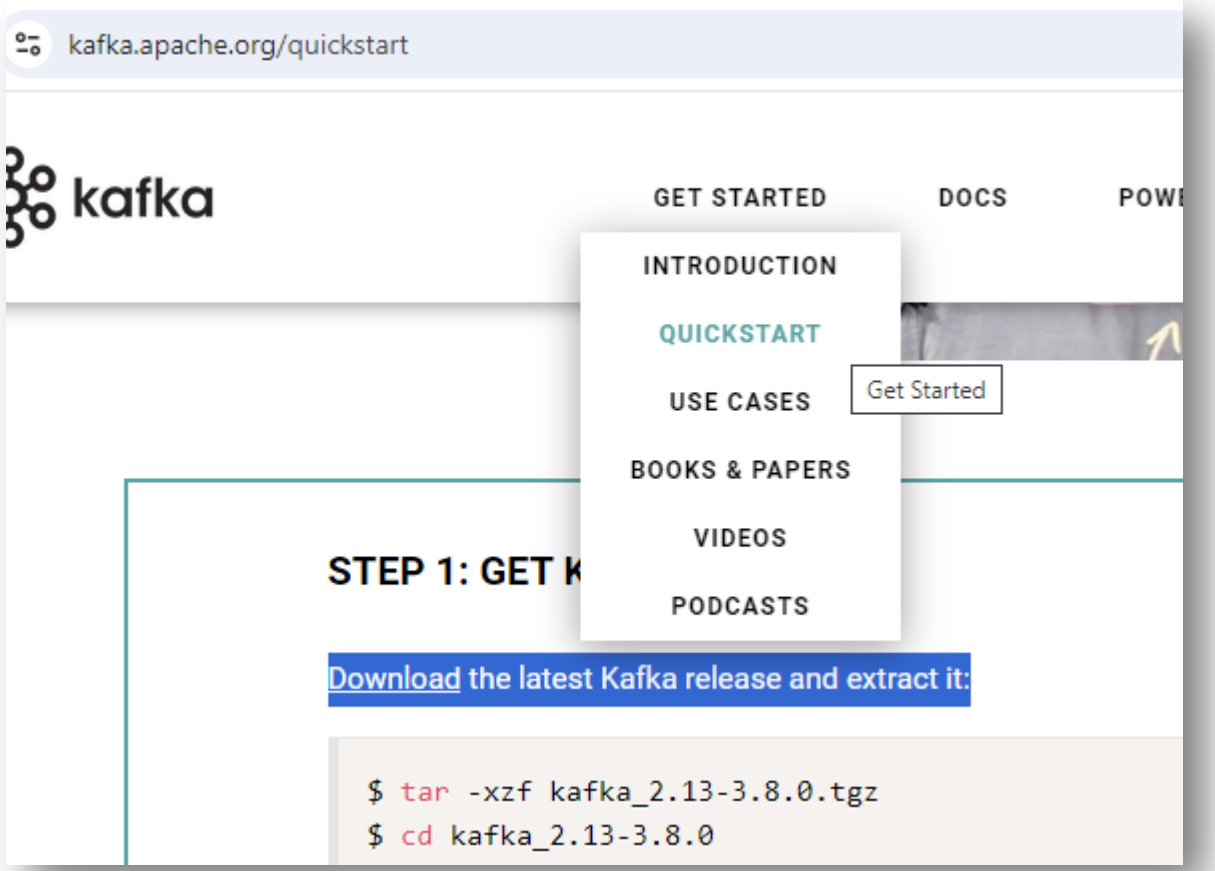
- **Topics**: A topic is a category or feed name to which records are stored and published. Topics are split into partitions to allow for parallel processing.
- **Partitions**: Each topic is divided into partitions, which are ordered, immutable sequences of records. Partitions enable Kafka to scale horizontally.
- **Offsets**: An offset is a unique identifier assigned to each message within a partition. It represents the position of the message in the partition. Each partition has its own sequence of offsets, starting from 0 and incrementing by 1 for each new message.

# Event Streaming step-by-step

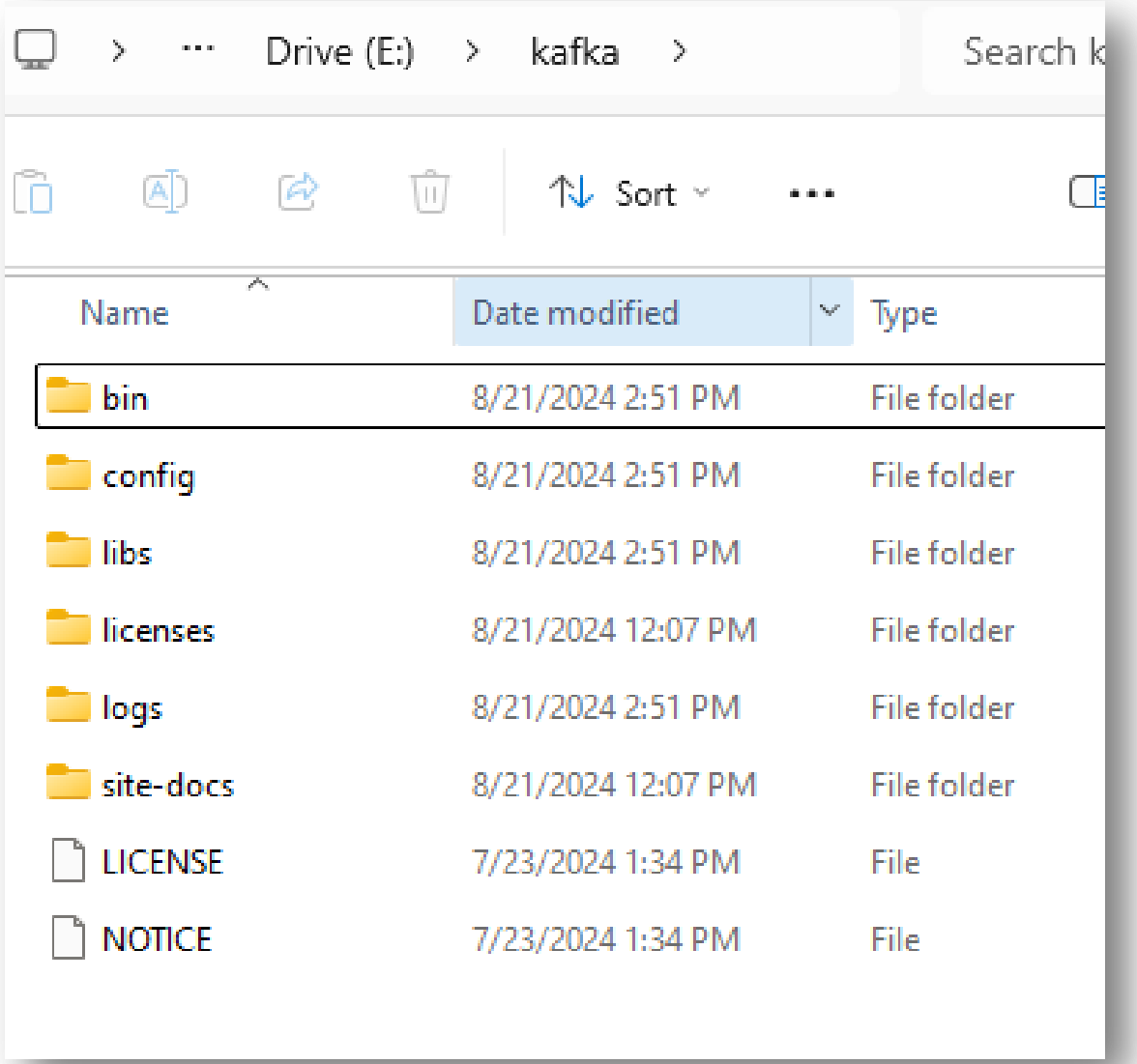
- Download & Extract Kafka
- Start Zookeeper to manage the environment
- Start Kafka Server (Broker)
- Create a Topic to store events
- Create a Producer (Console based)
- Write Events to Topic
- Create a Consumer (Console based)
- Read the Events from Topic

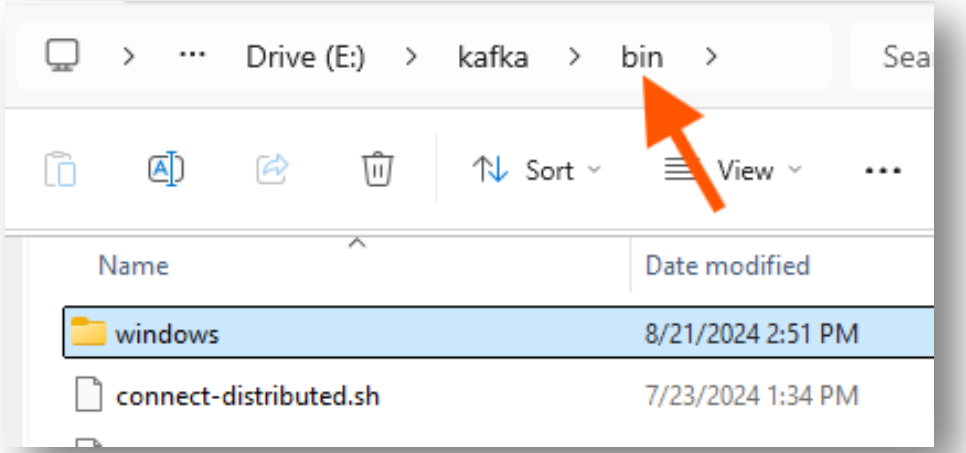
# Kafka Download

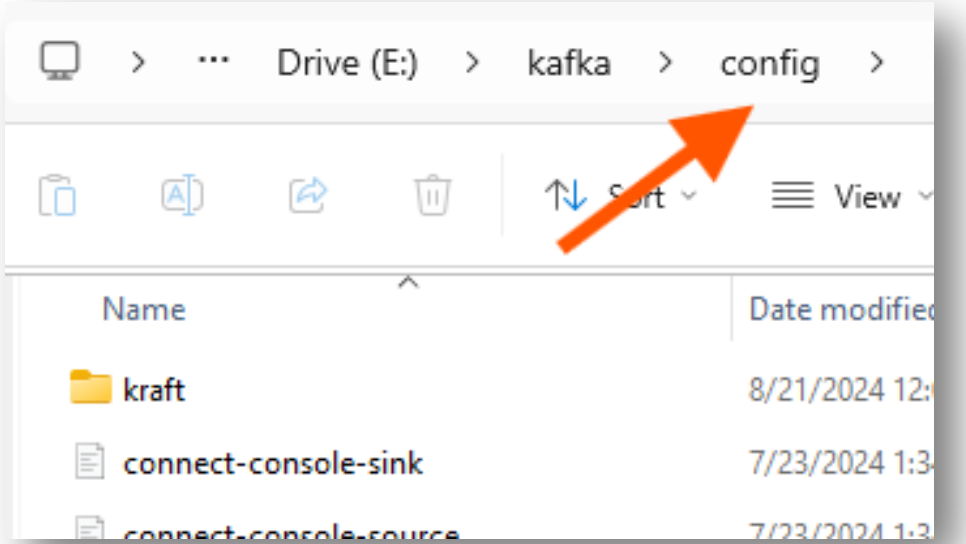
- Visit https://kafka.apache.org/quickstart
- Download the kafka server
- Extract it.
- You can find an another zip file
- Extract that and delete the version for easy reference
- Copy and paste the extracted folder in a drive.



# Extracted Kafka folder



# Start Zoopker

- Open a cmd from kafka folder
- Type the following command

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

```
E:\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
[2024-08-21 14:51:56,027] INFO Reading configuration from: .\config\zookeeper.proper
orum.QuorumPeerConfig)
[2024-08-21 14:51:56,043] WARN \tmp\zookeeper is relative. Prepend .\ to indicate th
r.server.quorum.QuorumPeerConfig)
[2024-08-21 14:51:56,051] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeep
```

# Start Kafka server (Broker)

- Open another cmd from kafka folder
- Type the following command

.\bin\windows\kafka-server-start.bat .\config\server.properties

```
E:\kafka>.\bin\windows\kafka-server-start.bat .\config\server.properties
[2024-08-21 15:34:11,891] INFO Registered kafka:type=kafka.Log4jController M
n$)
log4j:ERROR Failed to rename [E:\kafka/logs/server.log] to [E:\kafka/logs/se
[2024-08-21 15:34:13,063] INFO Setting -D jdk.tls.rejectClientInitiatedReneg
S renegotiation (org.apache.zookeeper.common.X509Util)
[2024-08-21 15:34:13,073] INFO RemoteLogManagerConfig values:
        log.local.retention.bytes = -2
        log.local.retention.ms = -2
        remote.fetch.max.wait.ms = 500
        remote.log.index.file.cache.total.size.bytes = 1073741824
```

# Create a Topic

- Open another cmd from kafka folder
- Type the following command

.\bin\windows\kafka-topics.bat --create --topic mytopic-1 --bootstrap-server localhost:9092

```
E:\kafka>.\bin\windows\kafka-topics.bat --create --topic mytopic-1 --bootstr
ap-server localhost:9092
Created topic mytopic-1.

E:\kafka>
```

# Create a Producer and write events

- Type the following command

.\bin\windows\kafka-console-producer.bat --topic mytopic-1 --bootstrap-server localhost:9092

- Write events and exit from the prompt

```
E:\kafka>.\bin\windows\kafka-console-producer.bat --topic mytopic-1 --bootst
rap-server localhost:9092
>MyEvent 1
>Hello World
>Welcome Marlabs
>Thank you
>Terminate batch job (Y/N)? y

E:\kafka>
```

# Create a consumer and get events from Topic

- Type the following command

.\bin\windows\kafka-console-consumer.bat --topic mytopic-1 --from-beginning --bootstrap-server localhost:9092

```
E:\kafka>.\bin\windows\kafka-console-consumer.bat --topic mytopic-1
--from-beginning --bootstrap-server localhost:9092
MyEvent 1
Hello World
Welcome Marlabs
Thank you
```

# Kafka + Springboot Application

# Springboot + Kafka

- Create a springboot app with web & kafka dependencies.

Service URL: https://start.spring.io

Name: spring-kafka

☑ Use default location

Location: E:\Java\MicroServices\Workspace1\spring-kafka

Type: Maven

Packaging: Jar

Java Version: 17

Language: Java

Group: com.rit

Artifact: spring-kafka

Version: 0.0.1-SNAPSHOT

Description: SpringKafka

Package: com.rit

Spring Boot Version: 3.3.2

Available:

kafka

- Messaging
  - ☑ Spring for Apache Kafka
  - ☐ Spring for Apache Kafka Streams
- Spring Cloud Messaging
  - ☐ Cloud Bus
  - ☐ Cloud Stream

Selected:

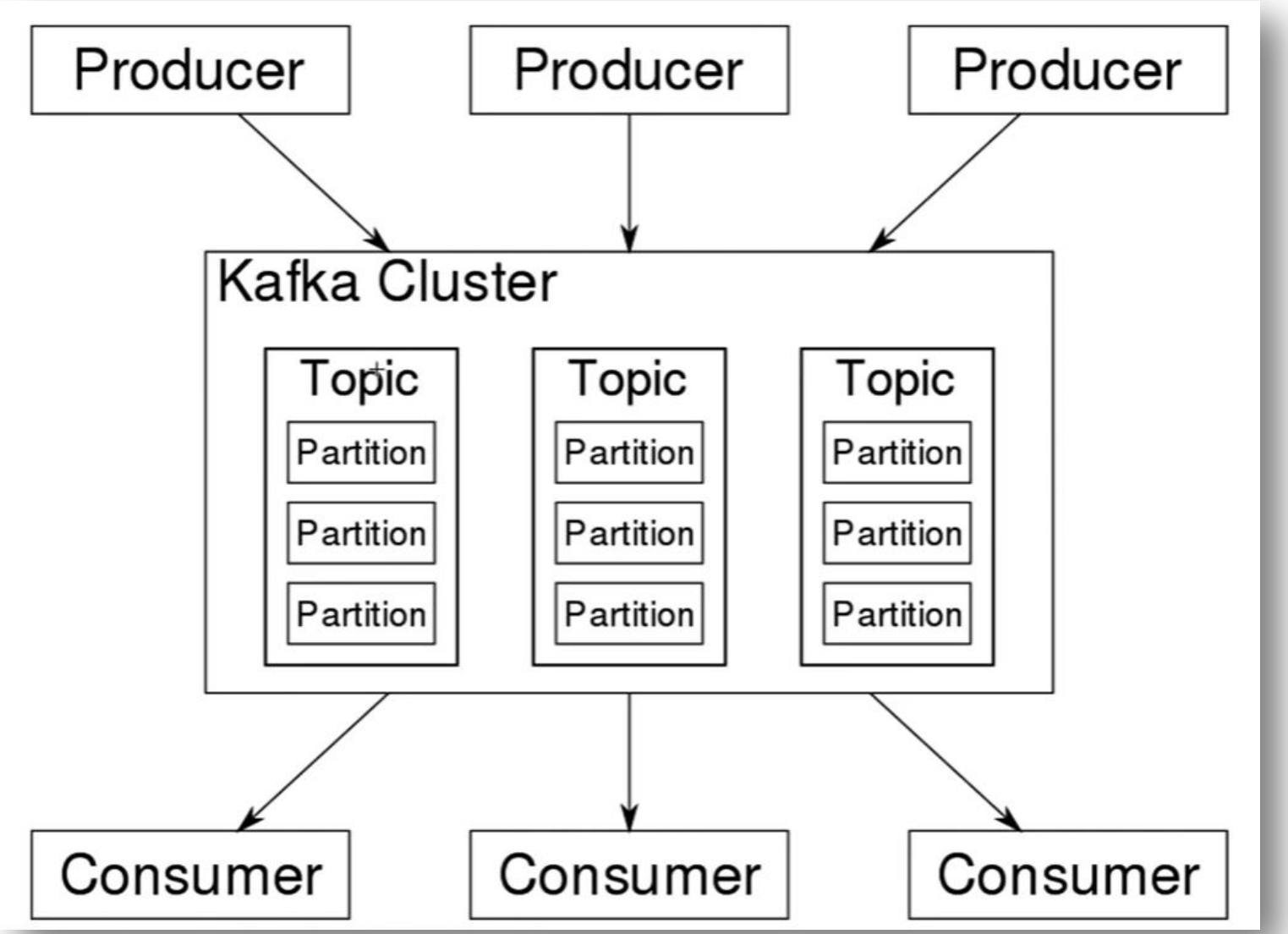
- X Spring for Apache Kafka
- X Spring Web

```
# Kafka broker address separated by comma, where the consumer will connect to fetch messages.
spring.kafka.consumer.bootstrap-servers=localhost:9092
# The consumer group ID used to identify the group of consumers to which this consumer belongs.
spring.kafka.consumer.group-id=myGroup
# 'earliest' means the consumer will start reading from the earliest available message.
spring.kafka.consumer.auto-offset-reset=earliest
# Used to deserialize the key of the message. Here, it is set to deserialize keys as strings.
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# Used to deserialize the value of the message. Here, it is set to deserialize values as strings.
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# The address of the Kafka broker. This is where the producer will connect to send messages.
spring.kafka.producer.bootstrap-servers=localhost:9092
# The class used to serialize the key of the message. Here, it is set to serialize keys as strings.
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
# The class used to serialize the value of the message. Here, it is set to serialize values as strings.
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
```

application.properties

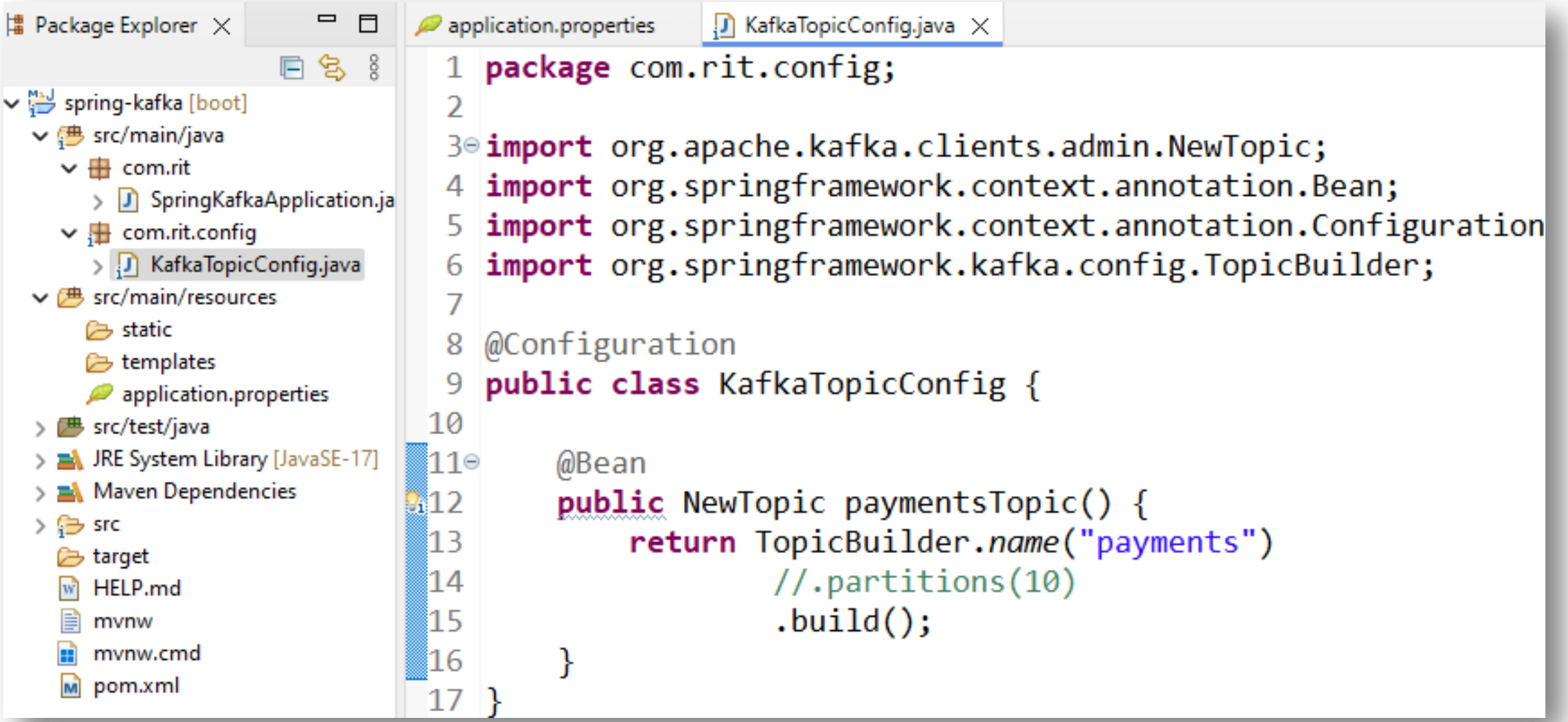
```
spring.application.name=spring-kafka

# The address of the Kafka broker. This is where the consumer will connect to fetch messages.
spring.kafka.consumer.bootstrap-servers=localhost:9092
# The consumer group ID, used to identify the group of consumers to which this consumer belongs.
spring.kafka.consumer.group-id=myGroup
# 'earliest' means the consumer will start reading from the earliest available message.
spring.kafka.consumer.auto-offset-reset=earliest
# Used to deserialize the key of the message. Here, it is set to deserialize keys as strings.
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# Used to deserialize the value of the message. Here, it is set to deserialize values as strings.
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# The address of the Kafka broker. This is where the producer will connect to send messages.


spring.kafka.producer.bootstrap-servers=localhost:9092
# Used to serialize the key of the message. Here, it is set to serialize keys as strings.
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
# Used to serialize the value of the message. Here, it is set to serialize values as strings.
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

```

# Create a Kafka Topic



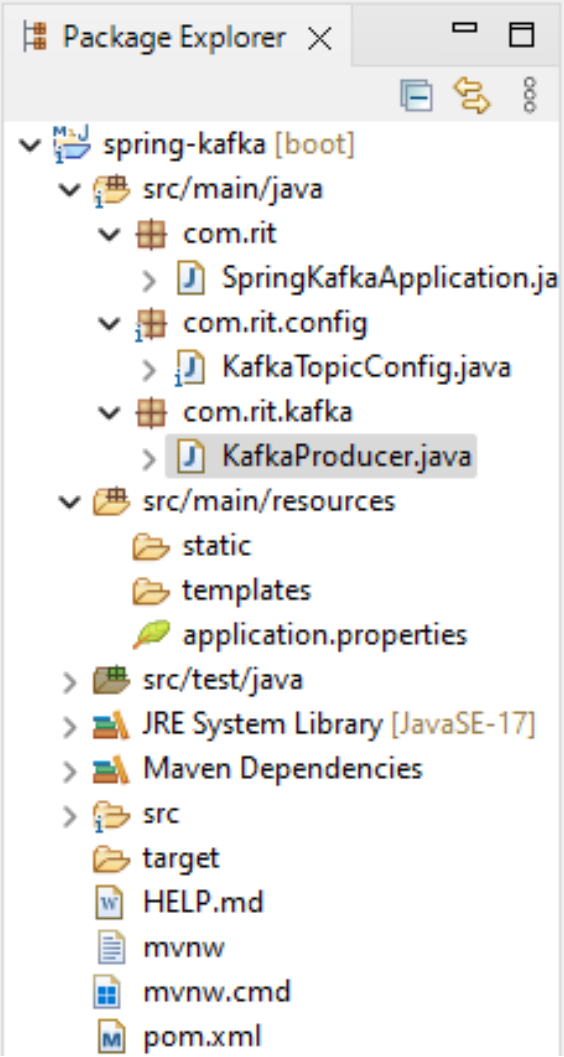
# Creating Topic

Create a class kafkaTopicConfig for Topic configuration in a new package.



# Creating Producer

Create a class kafkaProducer for send Message in a new package.




KafkaTopicConfig.java | KafkaProducer.java

```java
package com.rit.kafka;

import org.slf4j.Logger;

@Service
public class KafkaProducer {
    private 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("payments", message);
    }
}
```

KafkaTopicConfig.java | KafkaProducer.java | *MessageController.java

```java
package com.rit.controller;

import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/api/v1/kafka")
public class MessageController {

    private KafkaProducer kafkaProducer;

    public MessageController(KafkaProducer kafkaProducer) {
        this.kafkaProducer = kafkaProducer;
    }

    //http:localhost:8080/api/v1/kafka/publish?message=creditcard payment
    @GetMapping("/publish")
    public ResponseEntity<String> publish(@RequestParam("message") String message){
        kafkaProducer.sendMessage(message);
        return ResponseEntity.ok("Message Sent to the topic");
    }
}
```

# Run the project

- Ensure Zookeeper and kafka server is running
- Hit the endpoint url in browser
- To verify the sent message in topic
- Open a cmd to read from consumer console

.\bin\windows\kafka-console-consumer.bat --topic payments --from-beginning --bootstrap-server localhost:9092

```
E:\kafka>.\bin\windows\kafka-console-consumer.bat --topic payments
--from-beginning --bootstrap-server localhost:9092
creditcard payment
```

localhost:8080/api/v1/kafka/publish?message=creditcard%20payment

```
E:\kafka>.\bin\windows\kafka-console-consumer.bat --topic payments
--from-beginning --bootstrap-server localhost:9092
creditcard payment
```

localhost:8080/api/v1/kafka/publish?message=debitcard%20payment

```
E:\kafka>.\bin\windows\kafka-console-consumer.bat --topic payments
--from-beginning --bootstrap-server localhost:9092
creditcard payment
debitcard payment
```

# Consumer

KafkaTopicConfig.java | KafkaProducer.java | MessageController.java | KafkaConsumer.java

```java
package com.rit.kafka;

import org.slf4j.Logger;

@Service
public class KafkaConsumer {

    private Logger logger = LoggerFactory.getLogger(KafkaConsumer.class);

    @KafkaListener(topics="payments", groupId="myGroup")
    public void consume(String message) {
        logger.info(String.format("Message Received %s", message));
    }
}
```

# Just run the project and find all messages in the log

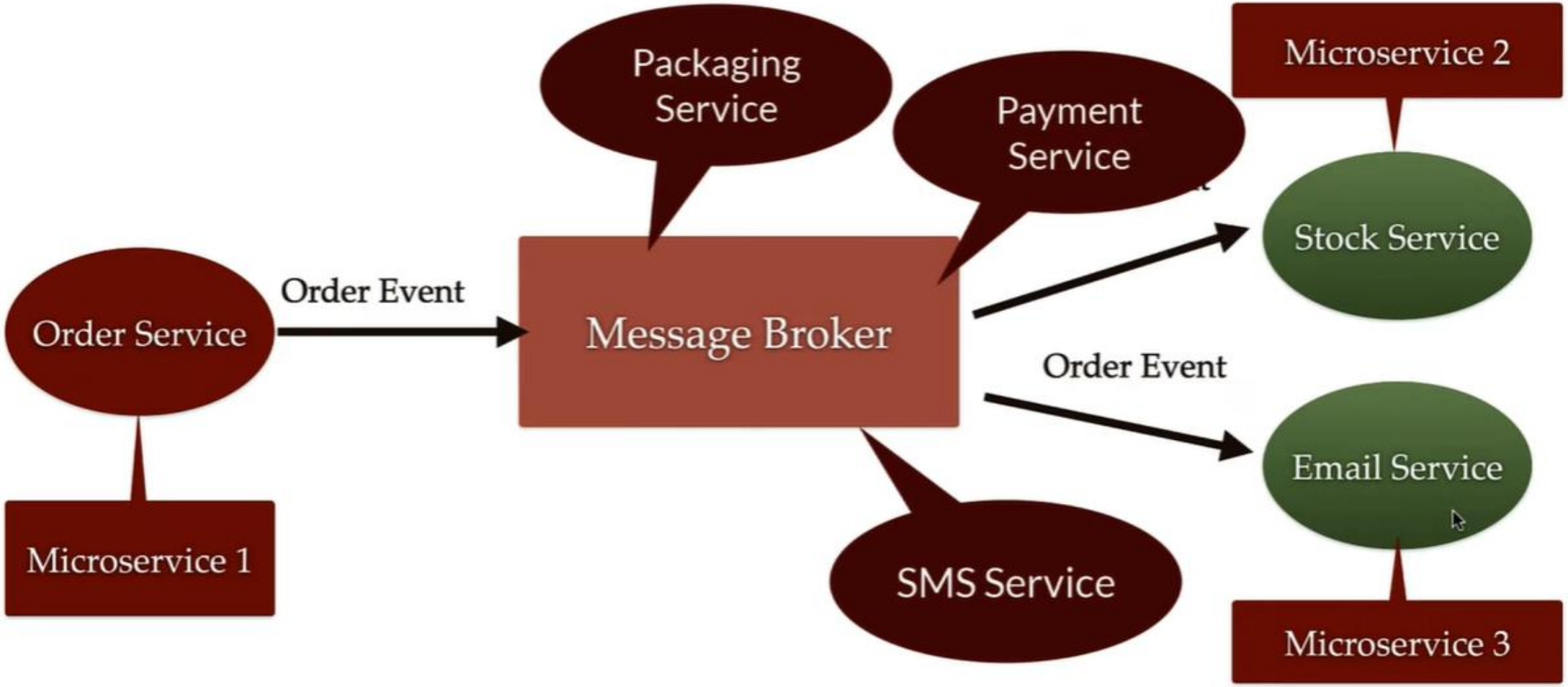
localhost:8080/api/v1/kafka/publish?message=Thank%20you

Message Sent to the topic

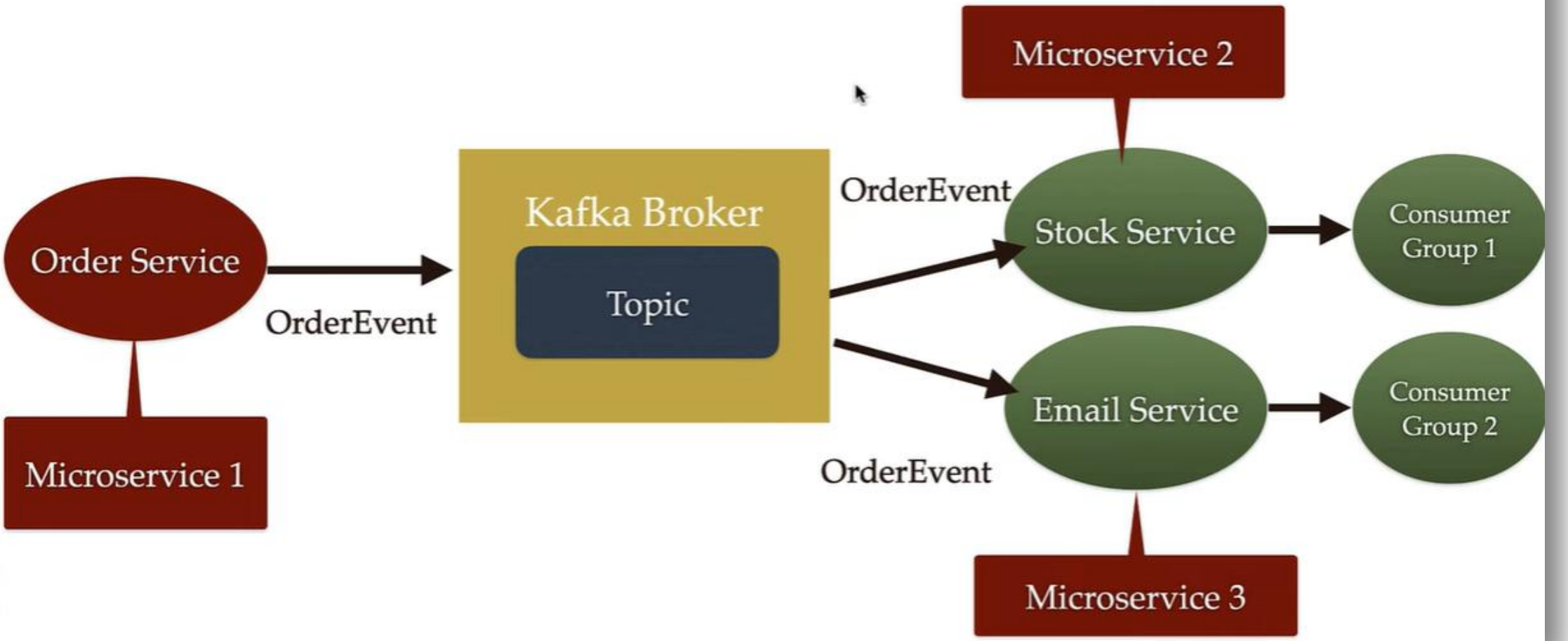
```
        : initializing Kafka metrics collec
cer     : [Producer clientId=spring-kafka-p
ser     : Kafka version: 3.7.1
ser     : Kafka commitId: e2494e6ffb89f828
ser     : Kafka startTimeMs: 1724264645343
        : [Producer clientId=spring-kafka-p
nager   : [Producer clientId=spring-kafka-p
        : Message Received Thank you
```

# Kafka + Microservices

# Event-Driven Microservices Architecture



# Spring Boot Kafka Event-Driven Microservices Architecture with Multiple Consumers



# Step by step implementation

- Create 4 micorservices
  1. OrderService, StockService, EmailService & BaseDomains
  2. Update Port nos in OrderService, StockService & EmailService
  3. Create Order, OrderEvent in BaseDomains
  4. Create and Congifure OrderService as a Producer

# OrderService

Service URL: https://start.spring.io

Name: Order-Service

☑ Use default location

Location: E:\Java\MicroServices\Workspace1\Order-Service

Type: Maven

Packaging: Jar

Java Version: 17

Language: Java

Group: com.rit

Artifact: Order-Service

Version: 0.0.1-SNAPSHOT

Description: Order Service

Package: com.rit

Spring Boot Version: 3.3.2

Frequently Used:

☑ Spring Web

☑ Spring for Apache Kafka

Available:

Type to search dependencies

- AI
- Developer Tools
- Google Cloud

Selected:

X Spring for Apache Kafka

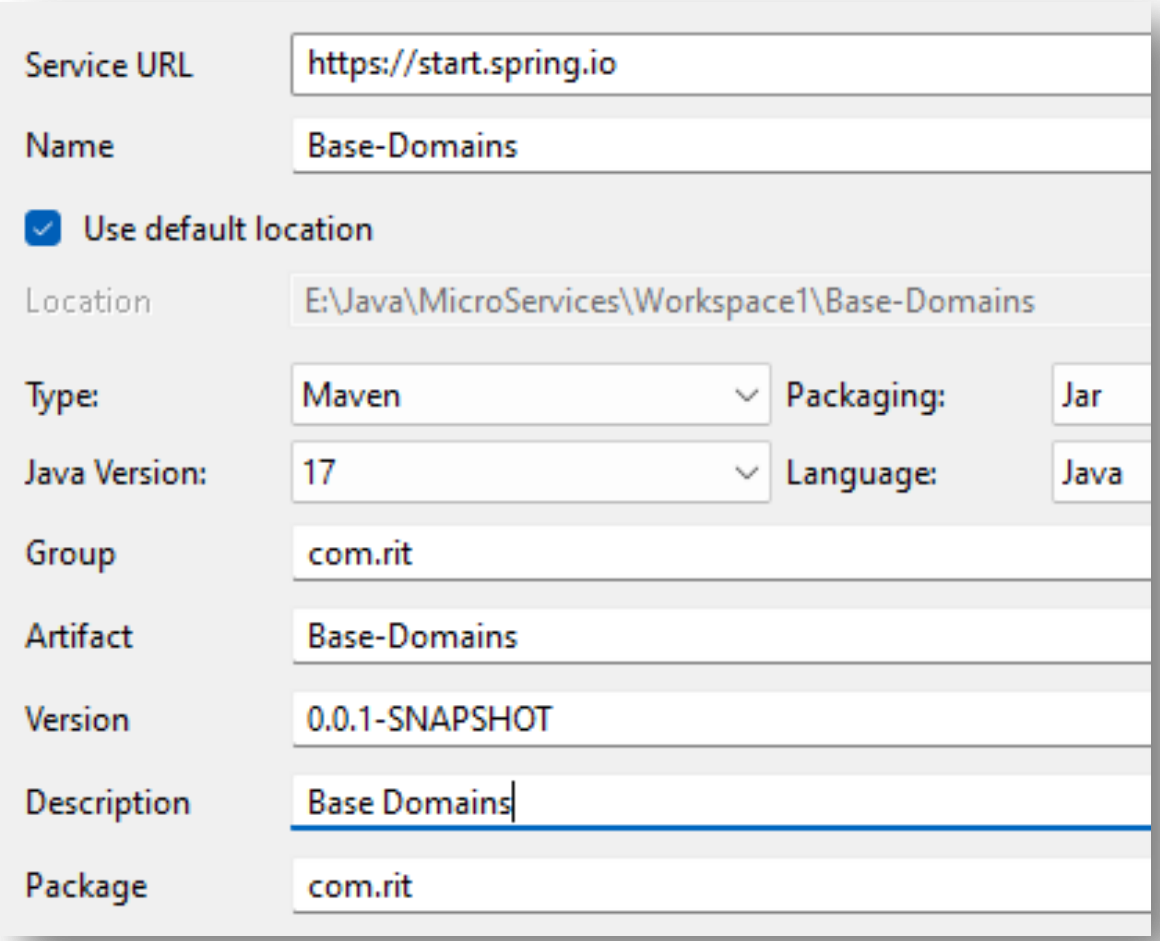
X Spring Web

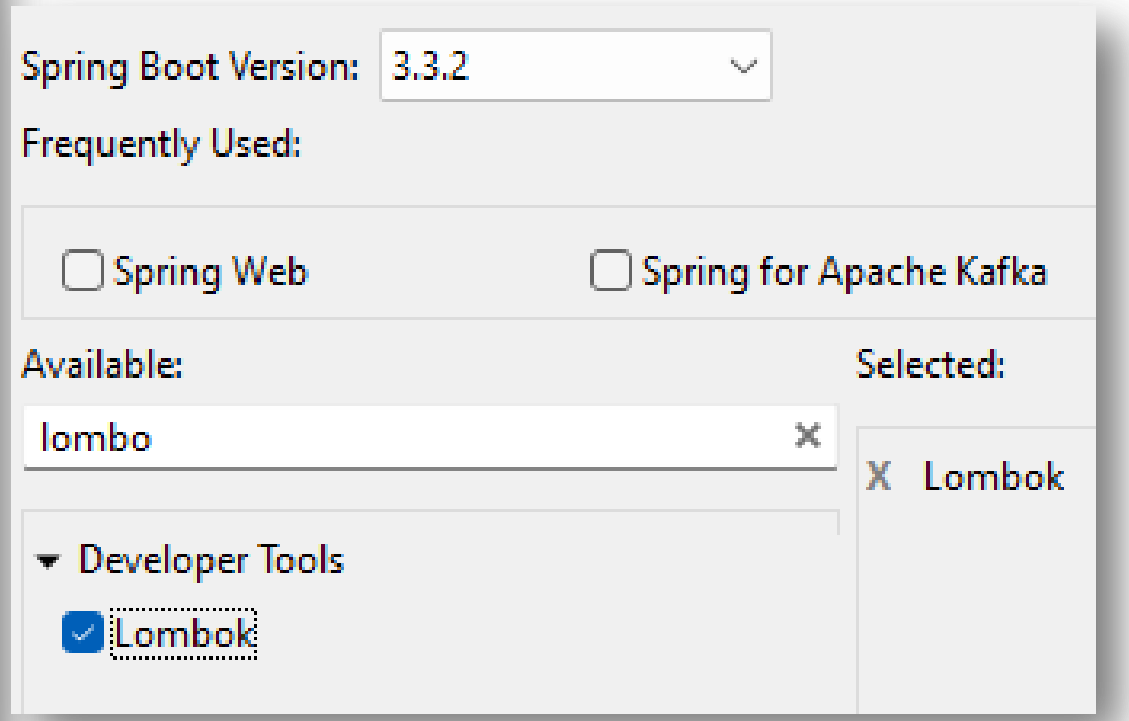
# Stock & Email services same as Order Service
# with web & kafka dependencies

| | |
|---|---|
| Service URL | https://start.spring.io |
| Name | Stock-Service |
| Use default location | ☑ |
| Location | E:\Java\MicroServices\Workspace1\Stock-Service |
| Type: | Maven |
| Packaging: | Jar |
| Java Version: | 17 |
| Language: | Java |
| Group | com.rit |
| Artifact | Stock-Service |
| Version | 0.0.1-SNAPSHOT |
| Description | Stock Service |
| Package | com.rit |

| | |
|---|---|
| Service URL | https://start.spring.io |
| Name | Email-Service |
| Use default location | ☑ |
| Location | E:\Java\MicroServices\Workspace1\Email-Service |
| Type: | Maven |
| Packaging: | Jar |
| Java Version: | 17 |
| Language: | Java |
| Group | com.rit |
| Artifact | Email-Service |
| Version | 0.0.1-SNAPSHOT |
| Description | Email Service |
| Package | com.rit |

# BaseDomains just with Lombok dependency



# Update port nos

- Order-Service : 8080
- Stock-Service : 8081
- Email-Service : 8082

- Base-Domains is doesn’t have web dependency so no need of port no.

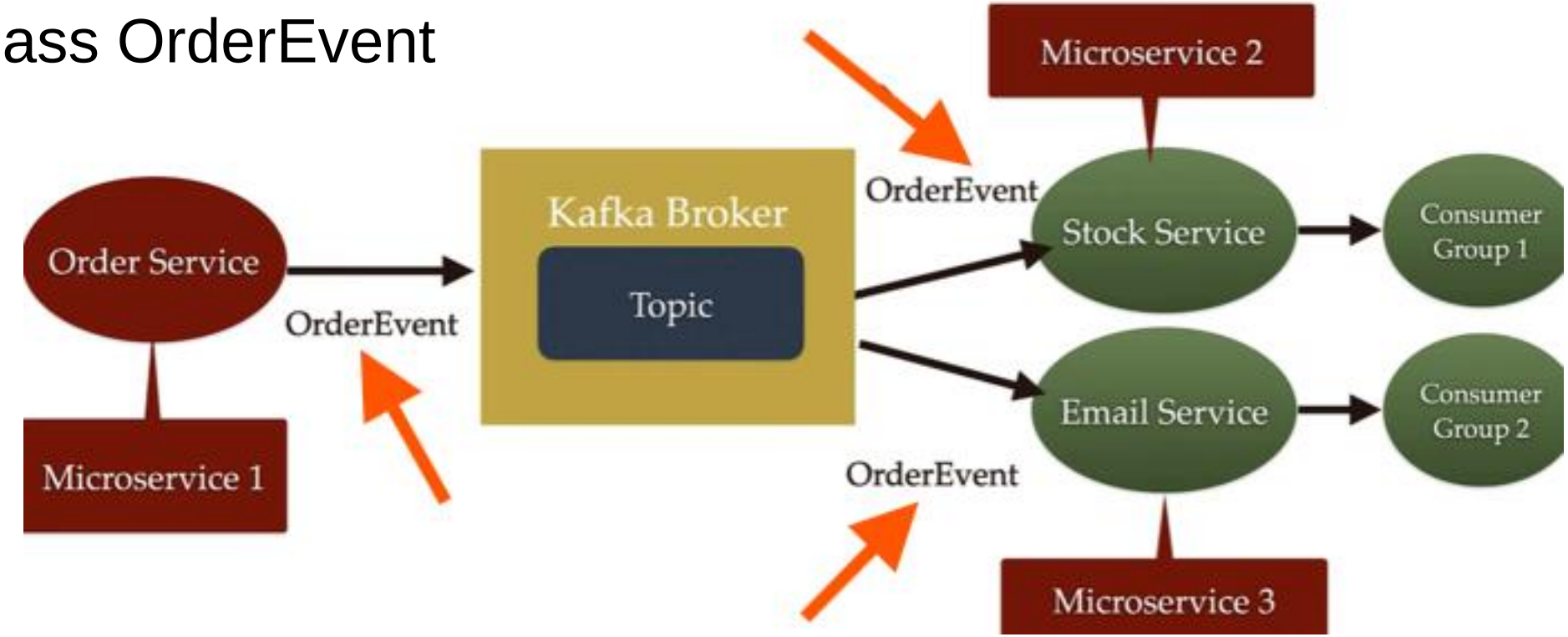
```
spring.application.name=Order-Service

server.port=8080
```

```
spring.application.name=Stock-Service

server.port=8081
```

```
spring.application.name=Email-Service

server.port=8082
```

# In Base-Domains Service

- Create a package dto
- Create a class Order
- Create a class OrderEvent



Package Explorer

- Base-Domains [boot]
  - src/main/java
    - com.rit
      - BaseDomainsApplica
    - com.rit.dto
      - Order.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-
  - Maven Dependencies
  - target/generated-sources/a
  - target/generated-test-sourc
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

*Order.java

```java
package com.rit.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Order {
    private String orderId;
    private String name;
    private int qty;
    private double price;
}
```

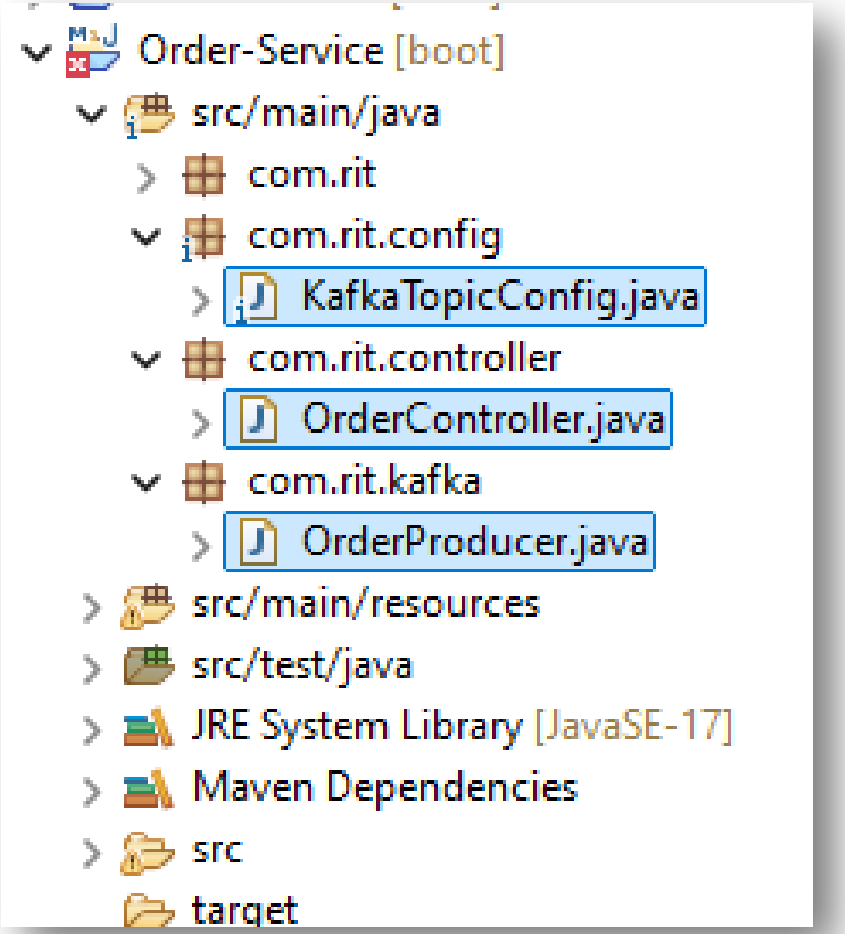
Package Explorer

- Base-Domains [boot]
  - src/main/java
    - com.rit
      - BaseDomainsApplica
    - com.rit.dto
      - Order.java
      - OrderEvent.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-
  - Maven Dependencies
  - target/generated-sources/a
  - target/generated-test-sourc
  - src
  - target
  - HELP.md
  - mvnw

Order.java | OrderEvent.java

```java
package com.rit.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderEvent {
    private String message;
    private String status;
    private Order order;
}
```

# Configure Order-Service as a Producer




- Start the Zookeeper
- Start the Kafka Server
- In OrderService
  - Add Base-Domains Dependency
  - Add Producer Config Properties
  - Create a KafkaTopicConfig class
  - Create OrderProducer Class
  - Create OrderController for Rest EndPoint

1. Copy the dependency details from Base-Domains and add it as a dependency in OrderService
2. Add the following properties to configure it as a Producer

```
    </parent>
    <groupId>com.rit</groupId>
    <artifactId>Base-Domains</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Base-Domains</name>
    <description>Base Domains</description>
```

```
    </dependency>
    <dependency>
        <groupId>com.rit</groupId>
        <artifactId>Base-Domains</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
```

plication.properties

```
spring.application.name=Order-Service

server.port=8080

spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
# Kafka doesnt have JsonSerialization, Hence we use spring kafka for it
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.topic.name=order_topics
```

KafkaTopicConfig.java

```java
package com.rit.config;

import org.apache.kafka.clients.admin.NewTopic;

@Configuration
public class KafkaTopicConfig {

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

    @Bean
    public NewTopic topic() {
        return TopicBuilder.name(topicName)
                .build();
    }
}
```

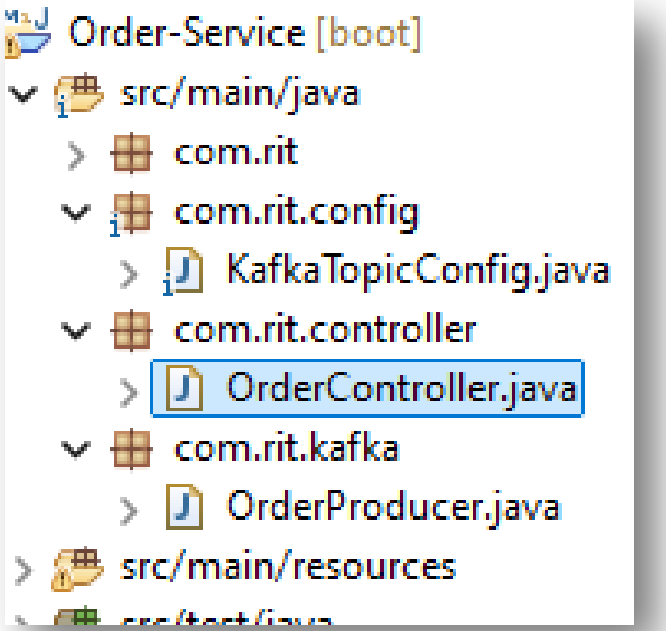
- Order-Service [boot]
  - src/main/java
    - com.rit
    - com.rit.config
      - KafkaTopicConfig.java
    - com.rit.controller
      - OrderController.java
    - com.rit.kafka
      - OrderProducer.java
  - src/main/resources
  - src/test/java

KafkaTopicConfig.java | OrderProducer.java

```java
12 import com.rit.dto.OrderEvent;
13
14 @Service
15 public class OrderProducer {
16     private Logger logger = LoggerFactory.getLogger(OrderProducer.class);
17
18     private NewTopic topic;
19     private KafkaTemplate<String, OrderEvent> kafkaTemplate;
20
21     public OrderProducer(NewTopic topic, KafkaTemplate<String, OrderEvent> kafkaTemplate) {
22         this.topic = topic;
23         this.kafkaTemplate = kafkaTemplate;
24     }
25
26     public void sendMessage(OrderEvent event) {
27         logger.info(String.format("Order event => %s", event.toString()));
28
29         //import org.springframework.messaging.Message;
30         Message<OrderEvent> message = MessageBuilder
31             .withPayload(event)
32             .setHeader(KafkaHeaders.TOPIC, topic.name())
33             .build();
34
35         kafkaTemplate.send(message);
36     }
37 }
```

- src/main/java
  - com.rit
  - com.rit.config
    - KafkaTopicConfig.java
  - com.rit.controller
    - OrderController.java
  - com.rit.kafka
    - OrderProducer.java
- src/main/resources

```java
@RestController
@RequestMapping("/api/v1")
public class OrderController {

    private OrderProducer orderProducer;

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

    @PostMapping("/orders")
    public String placeOder(@RequestBody Order order) {

        order.setOrderId(UUID.randomUUID().toString());

        OrderEvent orderEvent = new OrderEvent();
        orderEvent.setStatus("PENDING");
        orderEvent.setMessage("Order is Pending");
        orderEvent.setOrder(order);

        orderProducer.sendMessage(orderEvent);
        return "Order Placed Successfully";
    }
}
```



POST localhost:8080/api/v1/orders Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
{
    "orderId":101,
    "name":"Mobile Order",
    "qty" : 2,
    "price" : 23000.00
}
```

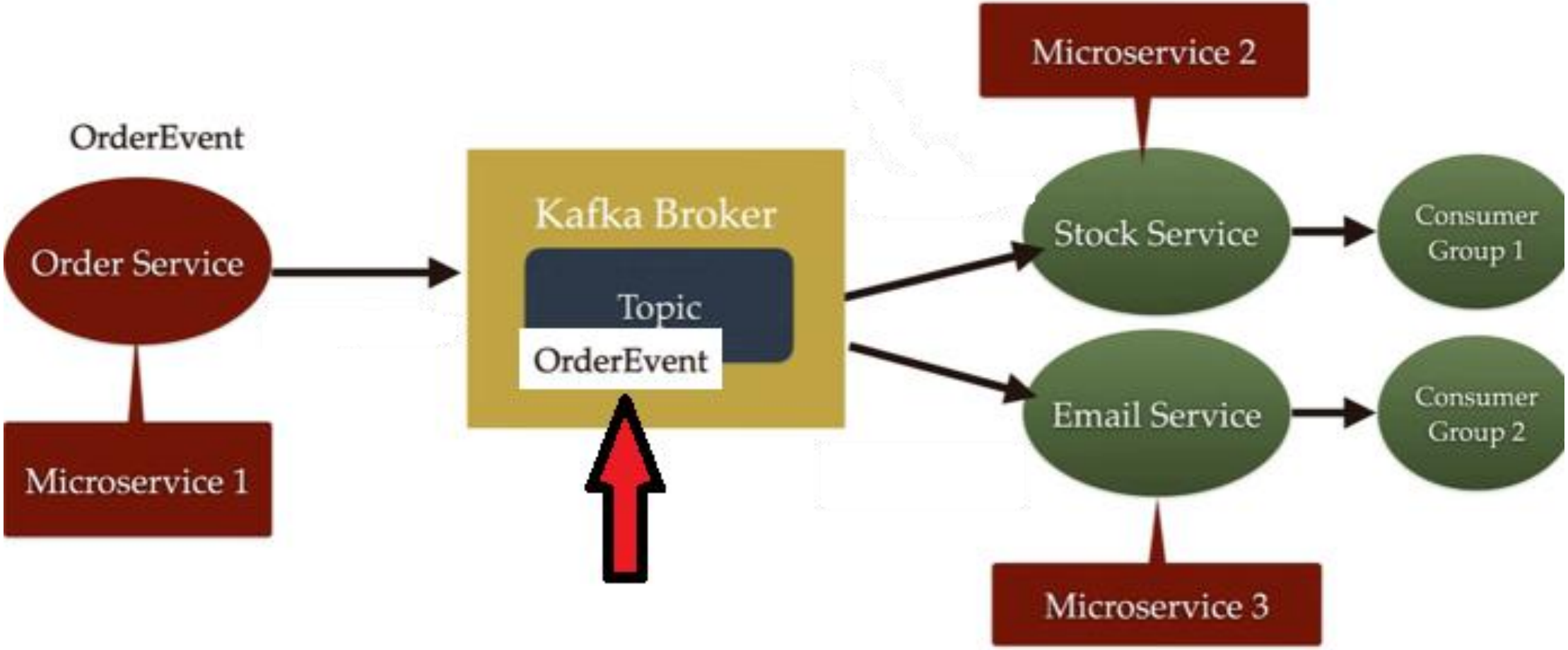
Body 200 OK 293 ms 189 B Save Response

Pretty Raw Preview Visualize Text

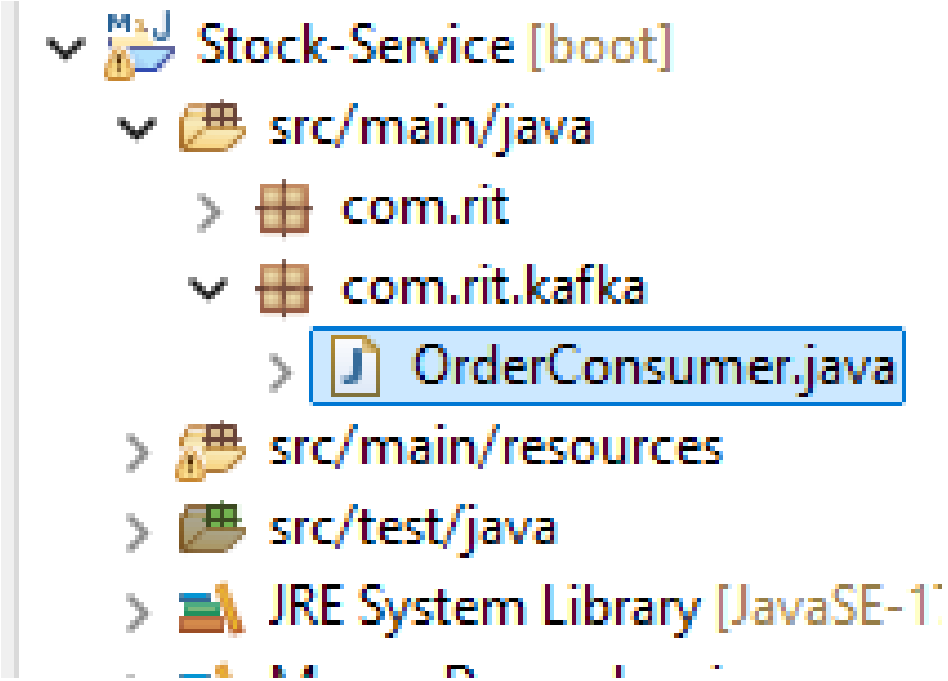
```
Order Placed Successfully
```

```
: Completed initialization in 1 ms
ver : Resolved [org.springframework.web.HttpRequestMethodNotSupportedExceptio
: Order event => OrderEvent [message=Order is Pending, status=PENDING, or
: ProducerConfig values:
```

# Now Producer Sent an OrderEvent to Topic



# Configure Stock-Service as a Consumer




- In StockService
  - Add Base-Domains Dependency
  - Add Consumer Config Properties
  - Create OrderConsumer Class
  - Run the app
  - Verify the logs

# Add Dependency

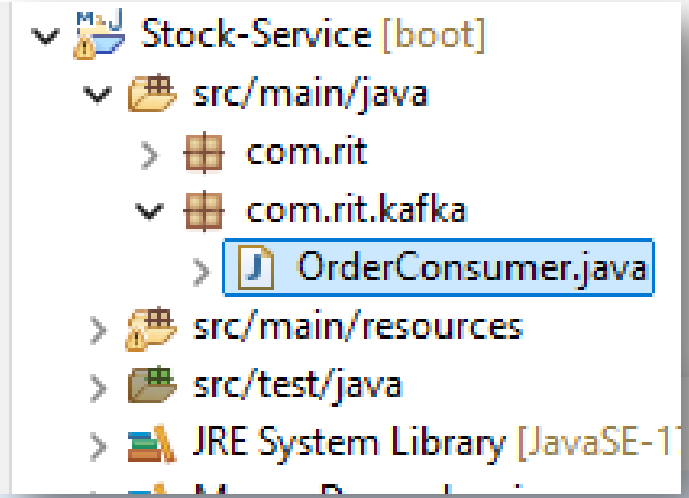
```
        <artifactId>spring-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>com.rit</groupId>
        <artifactId>Base-Domains</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
```

# Properties Configuration

Order-Service/pom.xml | application.properties

```
spring.application.name=Stock-Service

server.port=8081

# The address of the Kafka broker. This is where the consumer will connect to fetch messages.
spring.kafka.consumer.bootstrap-servers=localhost:9092
# The consumer group ID, used to identify the group of consumers to which this consumer belongs.
spring.kafka.consumer.group-id=stock
# 'earliest' means the consumer will start reading from the earliest available message.
spring.kafka.consumer.auto-offset-reset=earliest
# Used to deserialize the key of the message. Here, it is set to deserialize keys as strings.
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# Used to deserialize the value of the message. Here, it is set to deserialize values as strings.
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

# OrderConsumer




```java
@Service
public class OrderConsumer {

    private Logger logger = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(
        topics="${spring.kafka.topic.name}",
        groupId="${spring.kafka.consumer.group-id}"
    )
    public void consume(OrderEvent event) {
        logger.info("Order event received in Stock Service => %s ", event.toString() );

        //Save the event in database
    }
}
```

# Run the application

```
: Setting offset for partition order_topics-0 to the committed offset Fetch
: stock: partitions assigned: [order_topics-0]
: Order event received in Stock Service => OrderEvent [message=Order is Per
```

# Configure Email-Service as a Consumer

- Same as StockService
  - Add Base-Domains Dependency
  - Add Consumer Config Properties
  - Create OrderConsumer Class
  - Run the app
  - Verify the logs

# Add dependency

```
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>com.rit</groupId>
        <artifactId>Base-Domains</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
```

# Email Service Properties

```
spring.application.name=Email-Service

server.port=8082

# The address of the Kafka broker. This is where the consumer will connect to fetch messages.
spring.kafka.consumer.bootstrap-servers=localhost:9092
# The consumer group ID, used to identify the group of consumers to which this consumer belongs.
spring.kafka.consumer.group-id=email
# 'earliest' means the consumer will start reading from the earliest available message.
spring.kafka.consumer.auto-offset-reset=earliest
# Used to deserialize the key of the message. Here, it is set to deserialize keys as strings.
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
# Used to deserialize the value of the message. Here, it is set to deserialize values as strings.
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.topic.name=order_topics
```

# Copy & paste the kafka package

```
@Service
public class OrderConsumer {

    private Logger logger = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(
        topics="${spring.kafka.topic.name}",
        groupId="${spring.kafka.consumer.group-id}"
    )
    public void consume(OrderEvent event) {
        logger.info(String.format("Order event received in Email Service => %s ", event.toString()) );

        //send mail code snippet
    }
}
```

# Run the app & check the logs in all services

POST localhost:8080/api/v1/orders

Params Auth Headers (8) Body Pre-req.

raw JSON

```
1 {
2     "orderId":101,
3     "name":"laptop Order",
4     "qty" : 2,
5     "price" : 23000.00
6 }
```

Body 200 O

Pretty Raw Preview Visualize Te

```
1 Order Placed Successfully
```

```
[nio-8080-exec-9] com.rit.kafka.OrderProducer                    : Order
event => OrderEvent [message=Order is Pending, status=PENDING,
order=Order [orderId=95487270-4ef8-4bbc-9f17-d1ef5e3b8a25,
name=laptop Order, qty=2, price=23000.0]]
```

```
[ntainer#0-0-C-1] com.rit.kafka.OrderConsumer                    : Order
event received in Stock Service => OrderEvent [message=Order is
Pending, status=PENDING, order=Order [orderId=95487270-4ef8-
4bbc-9f17-d1ef5e3b8a25, name=laptop Order, qty=2, price=23000.0]]
```

```
[ntainer#0-0-C-1] com.rit.kafka.OrderConsumer                    : Order
event received in Email Service => OrderEvent [message=Order is
Pending, status=PENDING, order=Order [orderId=95487270-4ef8-
4bbc-9f17-d1ef5e3b8a25, name=laptop Order, qty=2, price=23000.0]]
```

Thank you