Apache Kafka

🌀 What is Kafka?

Apache Kafka is a **distributed messaging system** used to **send, store, and read data between applications**.

📦 Simple Example:

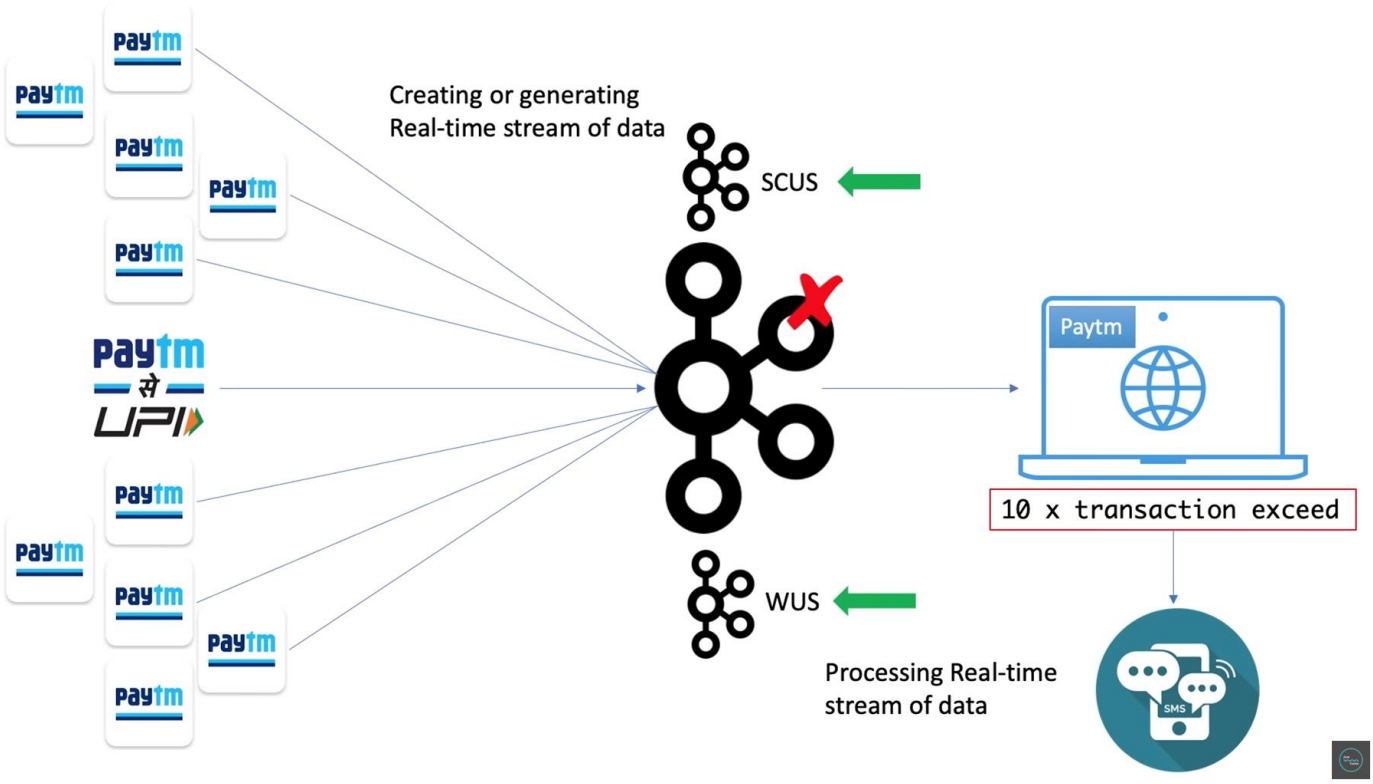
Imagine you have:

- 🛒 **Online Store** (Sends order info)
- 📦 **Shipping Service** (Needs to receive order info)
- 📧 **Email Service** (Sends confirmation emails)

Instead of each system talking directly to each other, they all send and receive messages **through Kafka**.

💡 Kafka works like:

- A **middleman** (message broker)
- That holds **messages (data)** in something called a **topic**
- Producers **send** messages to Kafka
- Consumers **read** messages from Kafka



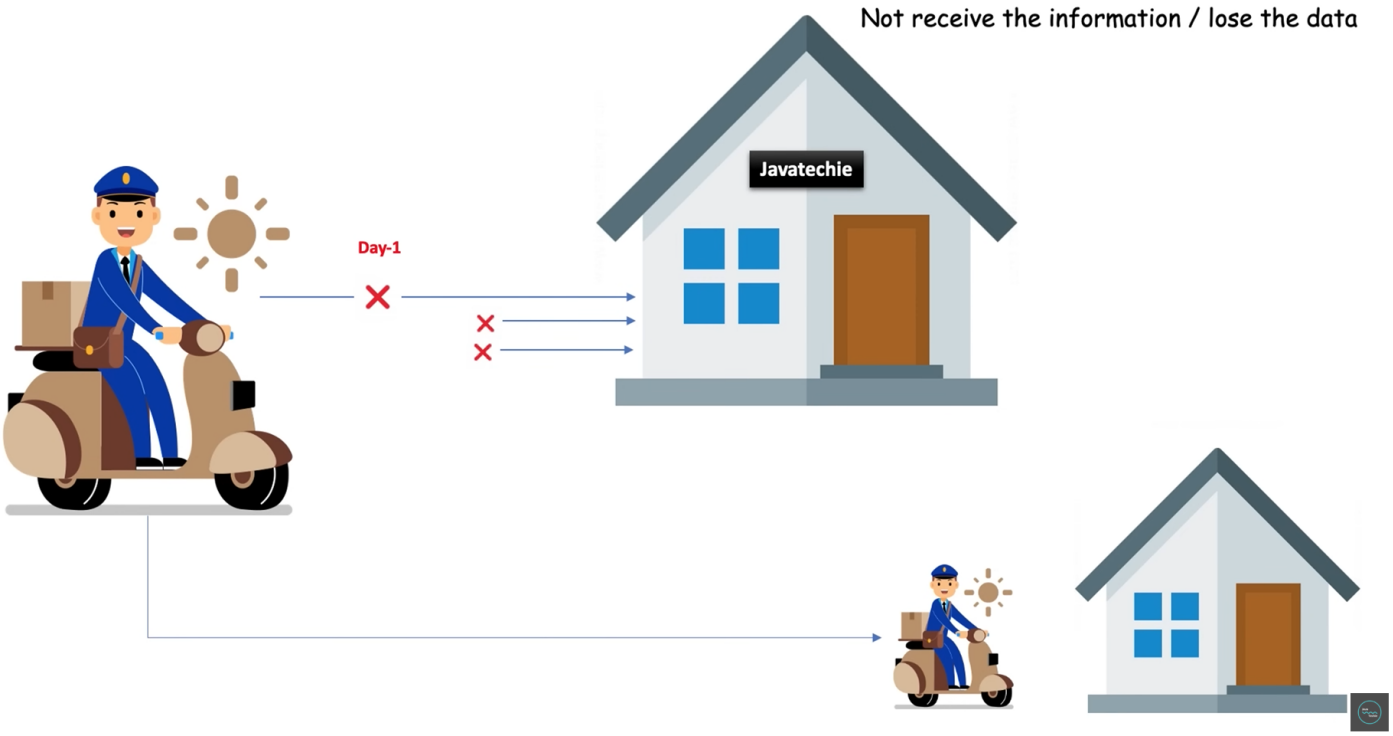
Where does Kafka come from ?

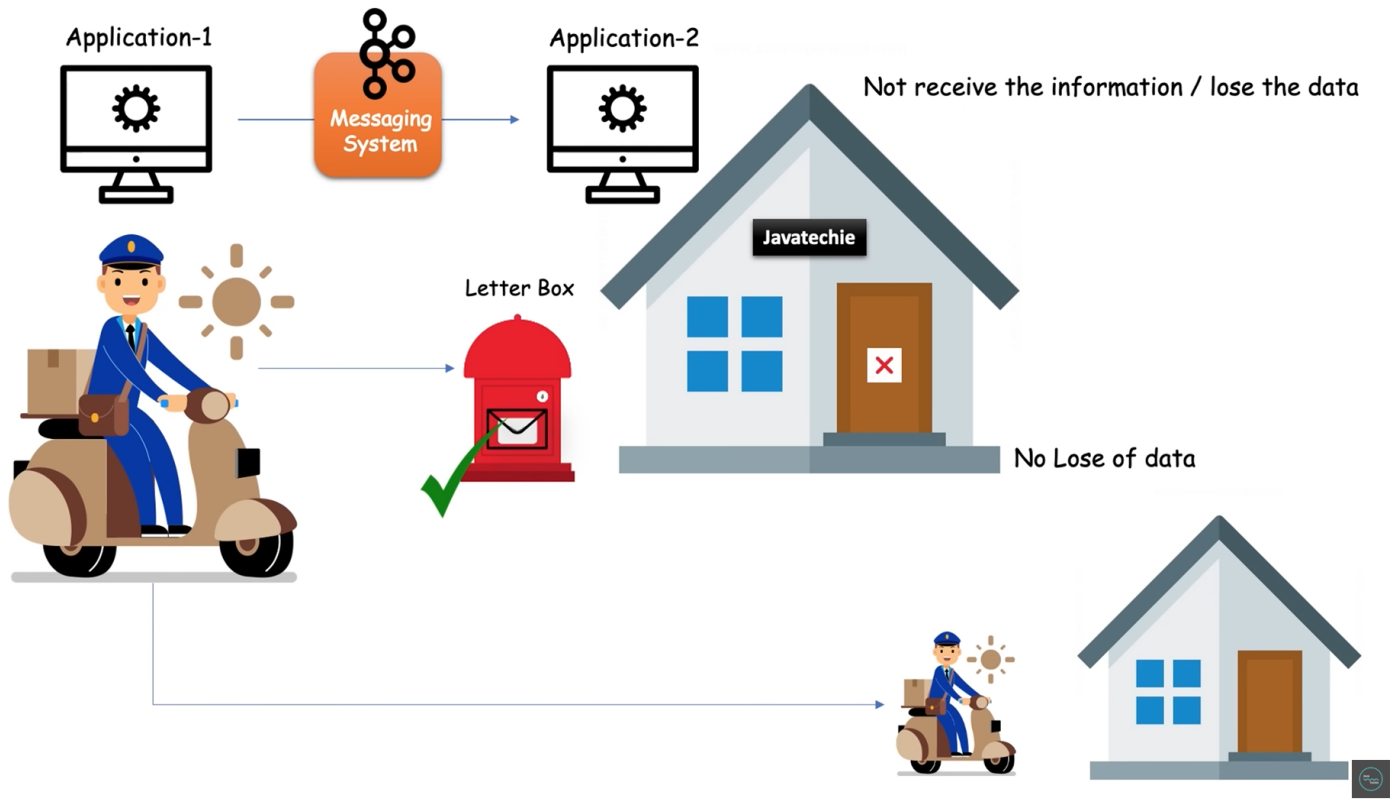
Kafka was originally developed at in , and was subsequently open sourced in early **2011**

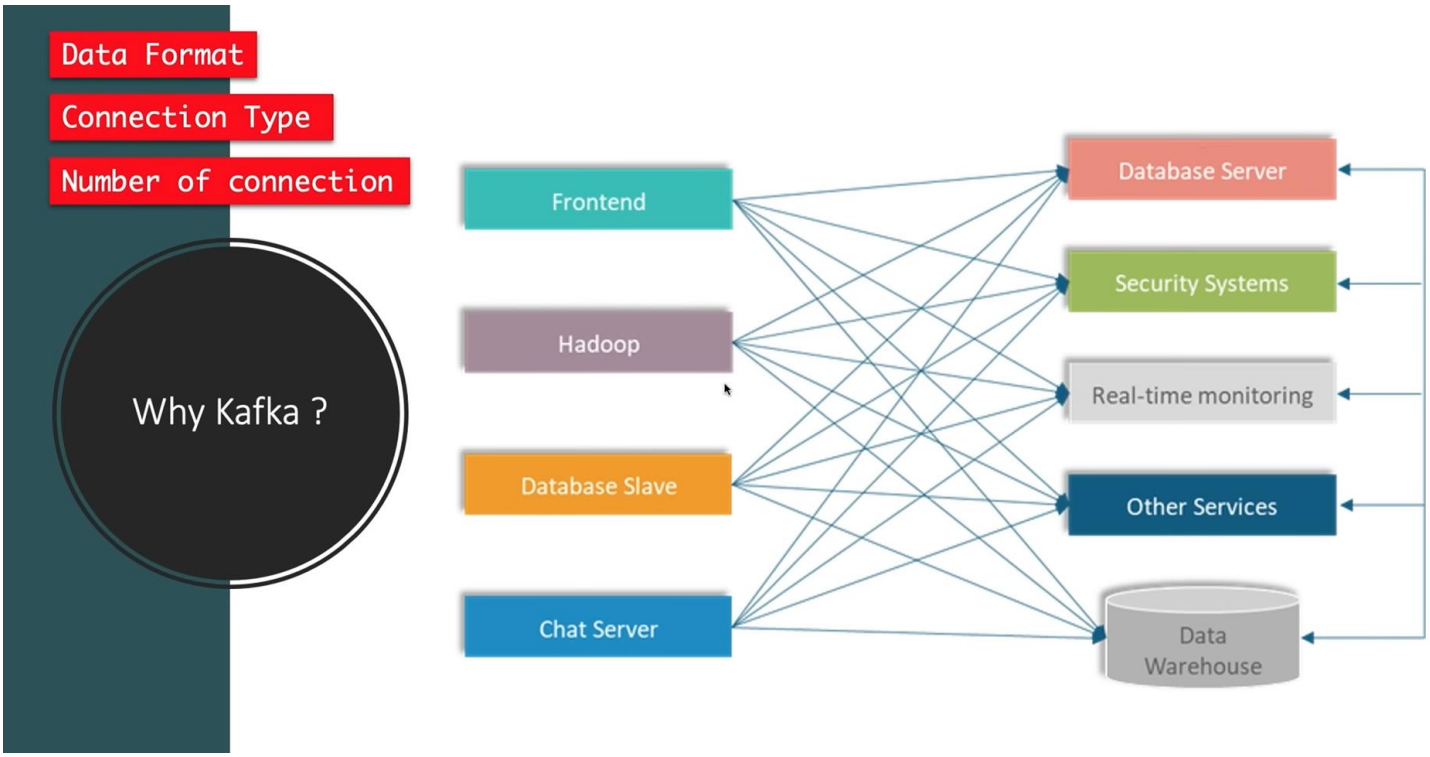
THE APACHE® SOFTWARE FOUNDATION

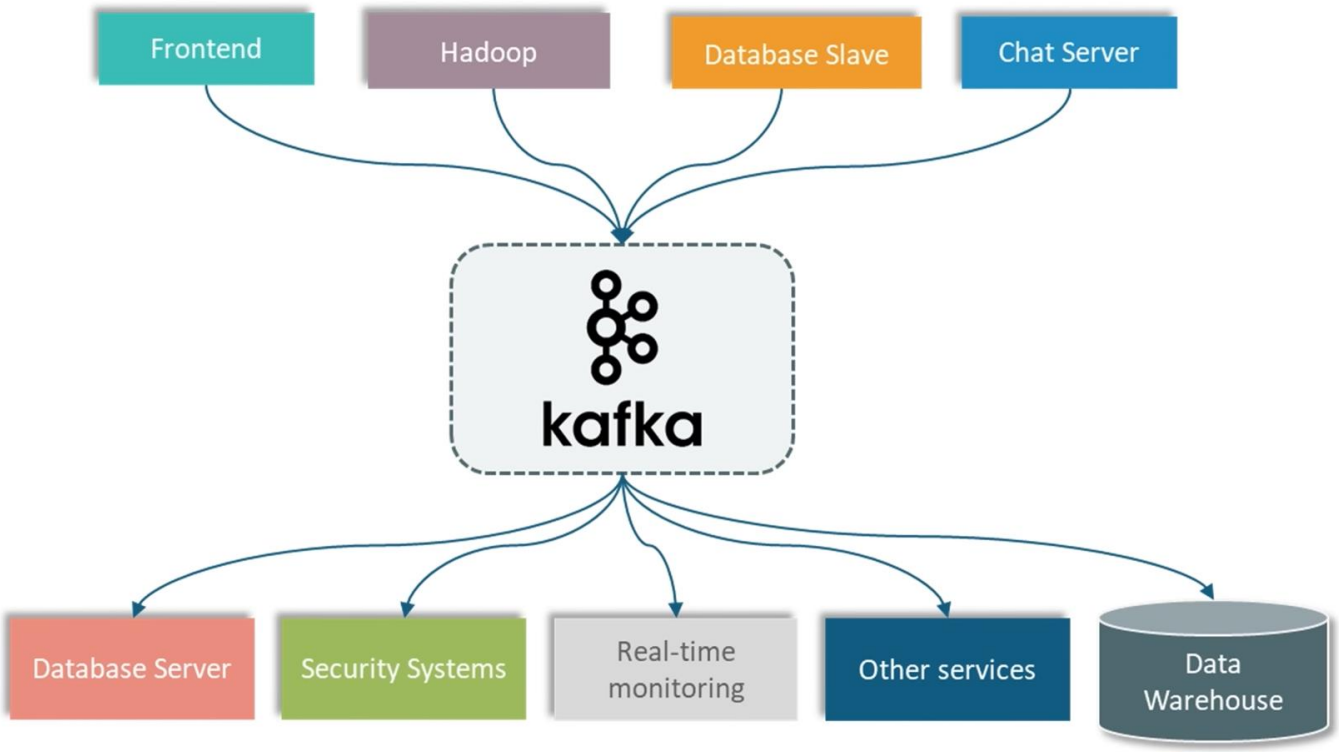
Why do we need Kafka?

We need Kafka when we want to handle real-time data and allow different systems to communicate efficiently, especially in microservices or large systems.

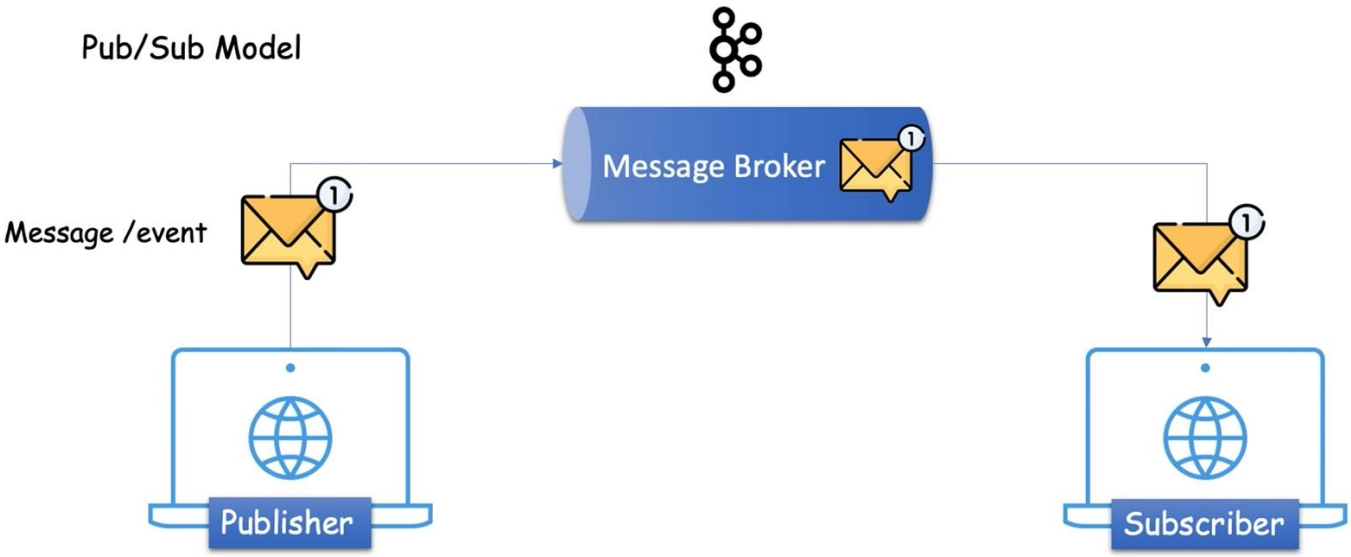



Application-1
Messaging System
Application-2
Not receive the information / lose the data
Javatechie
Letter Box
No Lose of data


Data Format
Connection Type
Number of connection
Why Kafka ?
Frontend
Hadoop
Database Slave
Chat Server
Database Server
Security Systems
Real-time monitoring
Other Services
Data Warehouse

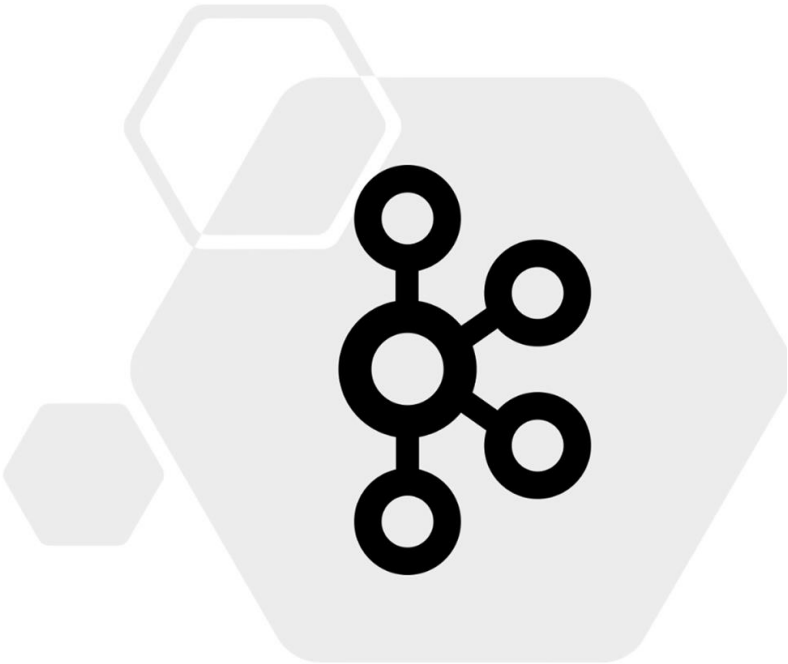


How does it work (High-level overview)

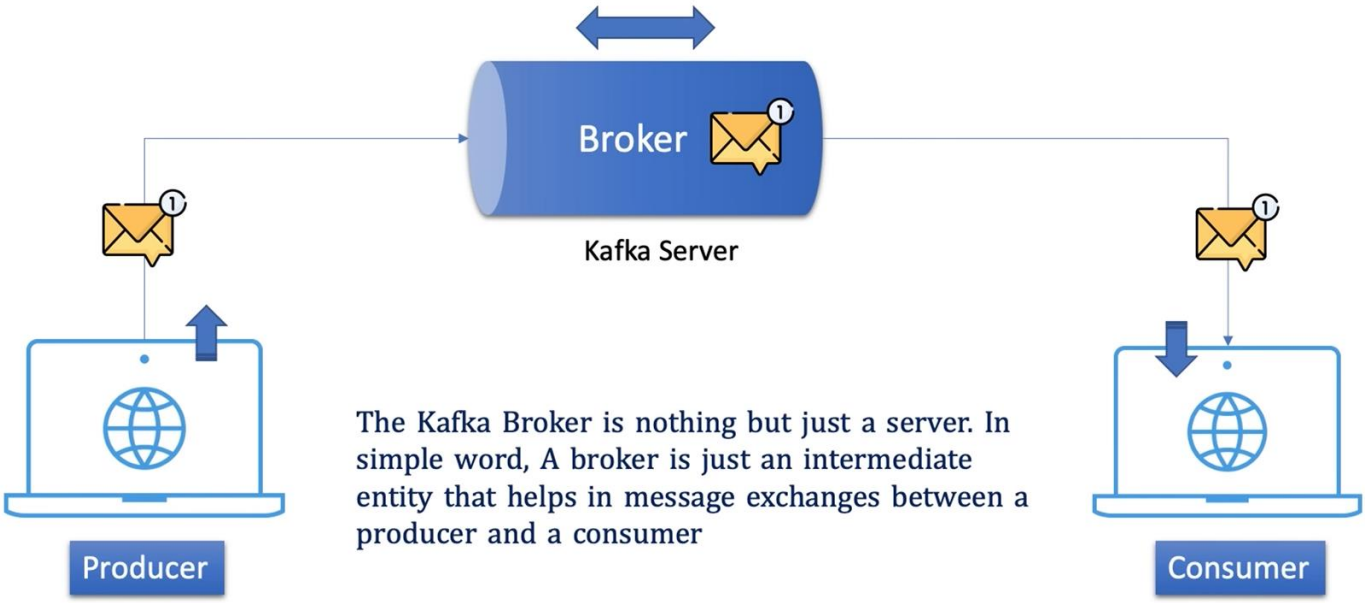


Kafka Components

- Producer
- Consumer
- Broker
- Cluster
- Topic
- Partitions
- Offset
- Consumer Groups
- Zookeeper

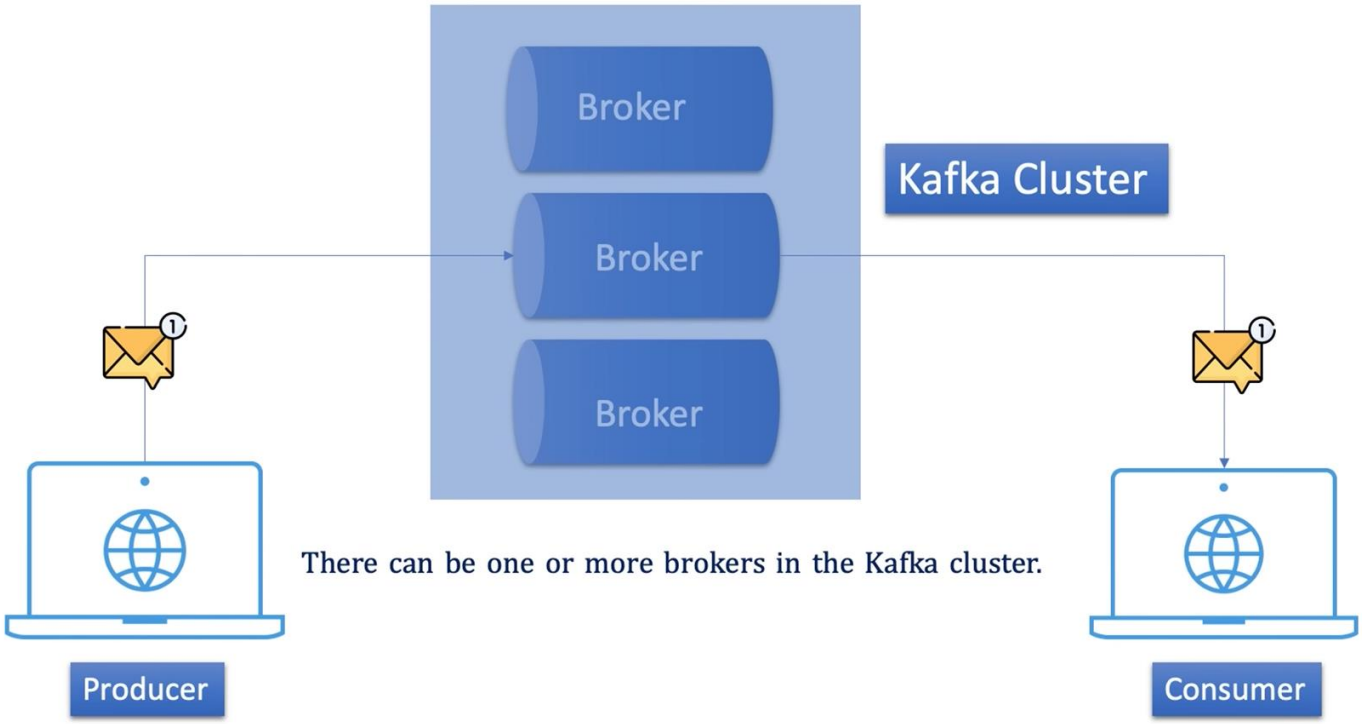


Producer & Consumer and Broker :

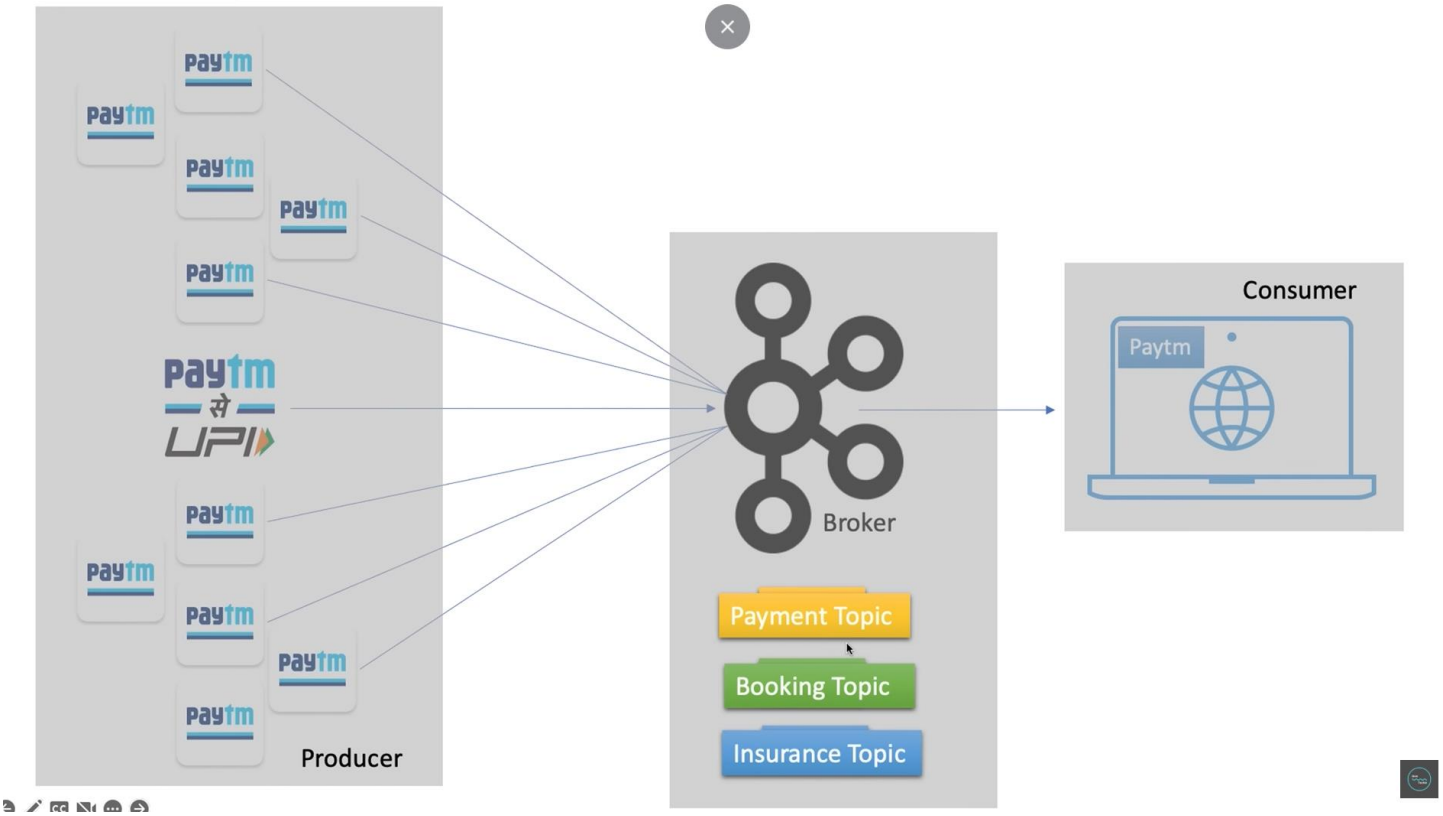


The Kafka Broker is nothing but just a server. In simple word, A broker is just an intermediate entity that helps in message exchanges between a producer and a consumer

Cluster :



Topic :

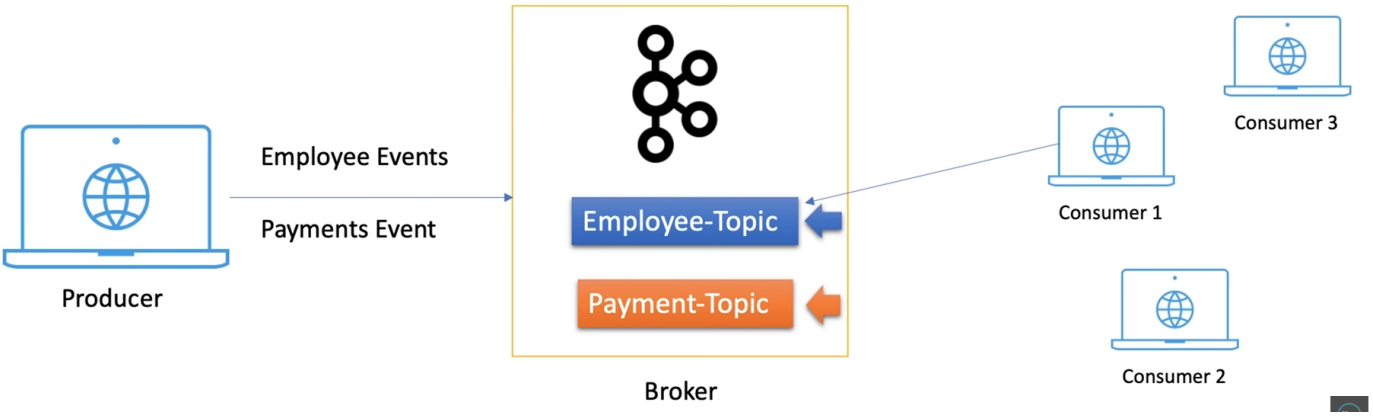


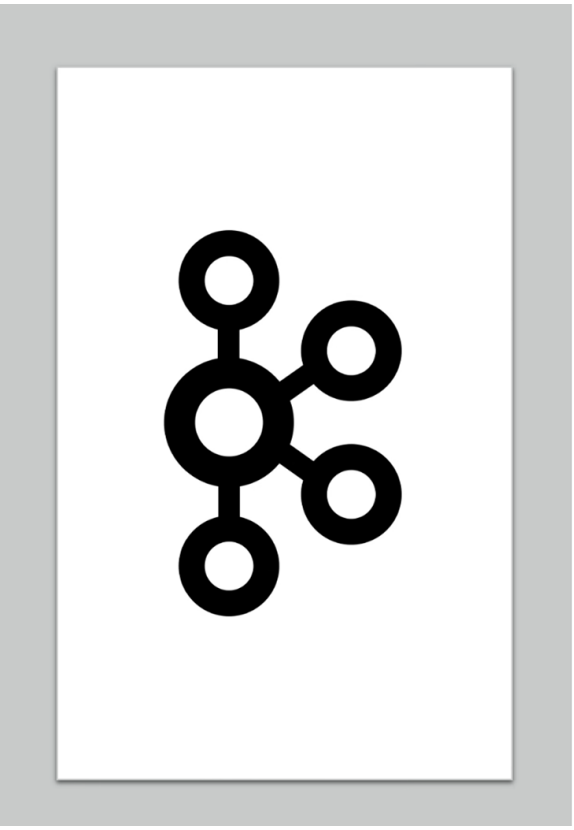
EMPLOYEE_TABLE

ID	NAME	DEPT	SALARY
1	john	IT	50000
2	sam	Bank	80000
3	joe	Admin	20000

PAYMENT_TABLE

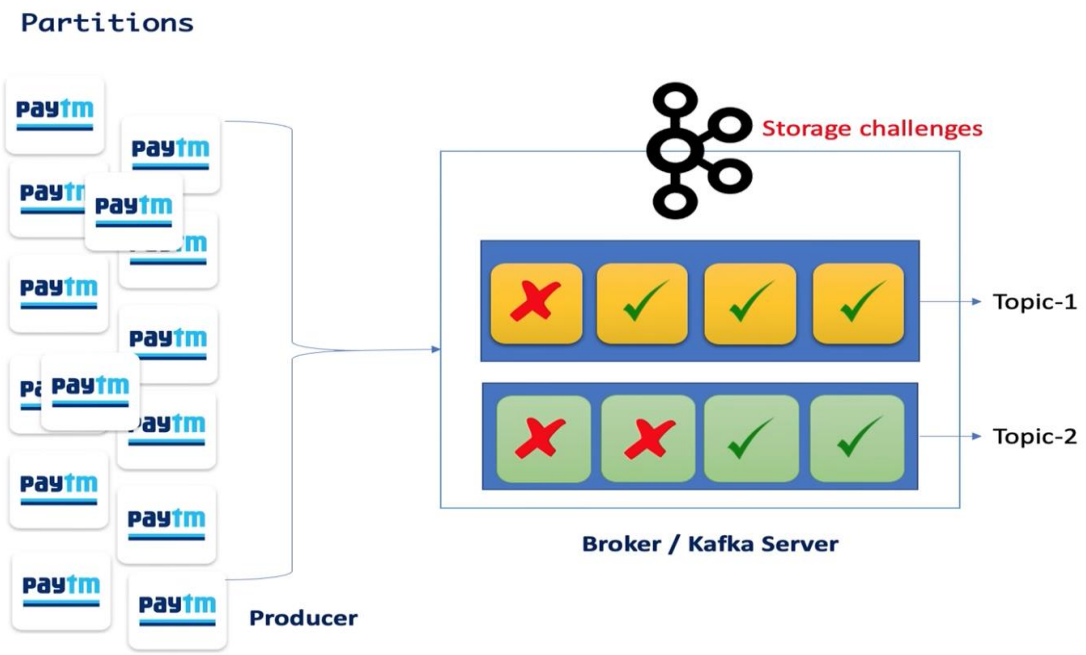
PAYMENT_ID	CNAME	PAY MODE	SRC AC	DEST AC	AMOUNT
df364bdjfb	Basant	CREDIT	HDFC12	SBI239837	100000
bcsr72386d	Santosh	DEBIT	ICICI893	HDFC3626	890012
dfv834tjsd8	Krishna	CHEQUE	SBI2367	IDBI23726	23273582



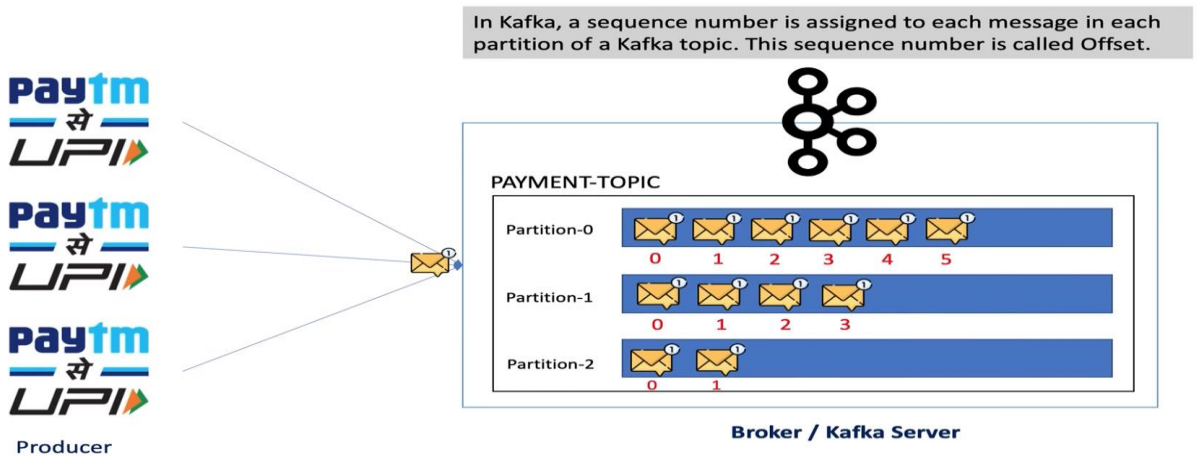


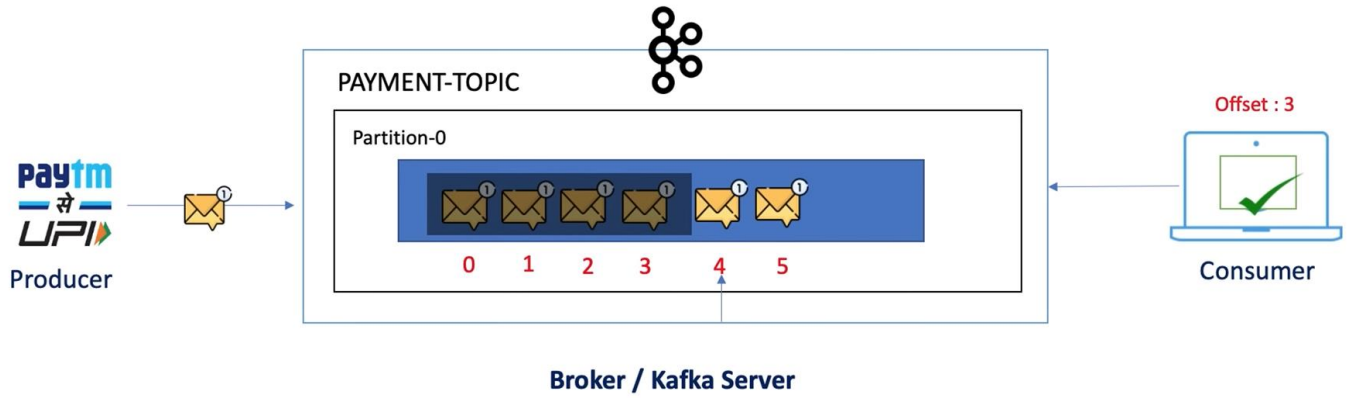
TOPIC

It specifies the category of the message or the classification of the message. Listeners can then just respond to the messages that belong to the topics they are listening on.

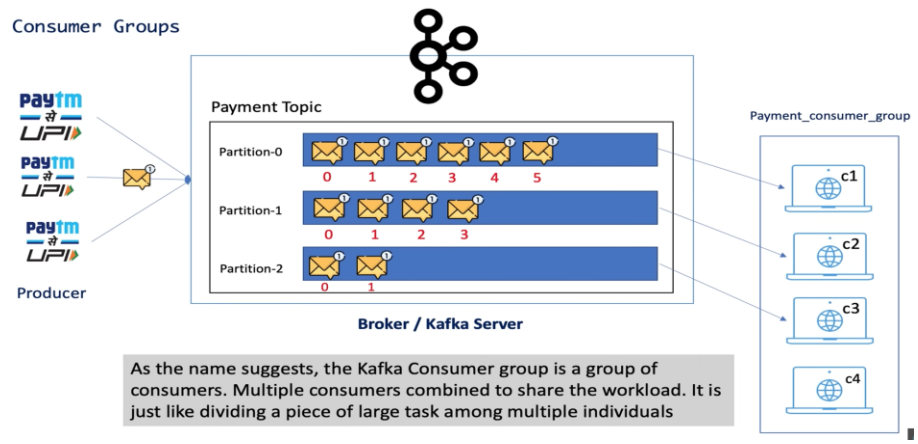


Offset :

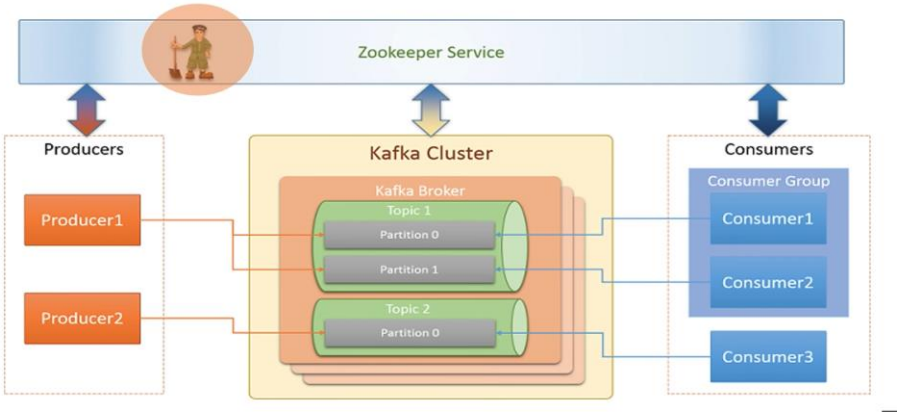




It will read from offset 4.



Zookeeper :



ZooKeeper is a **distributed coordination service** used by Kafka (before version 2.8) to manage metadata and health of Kafka brokers.

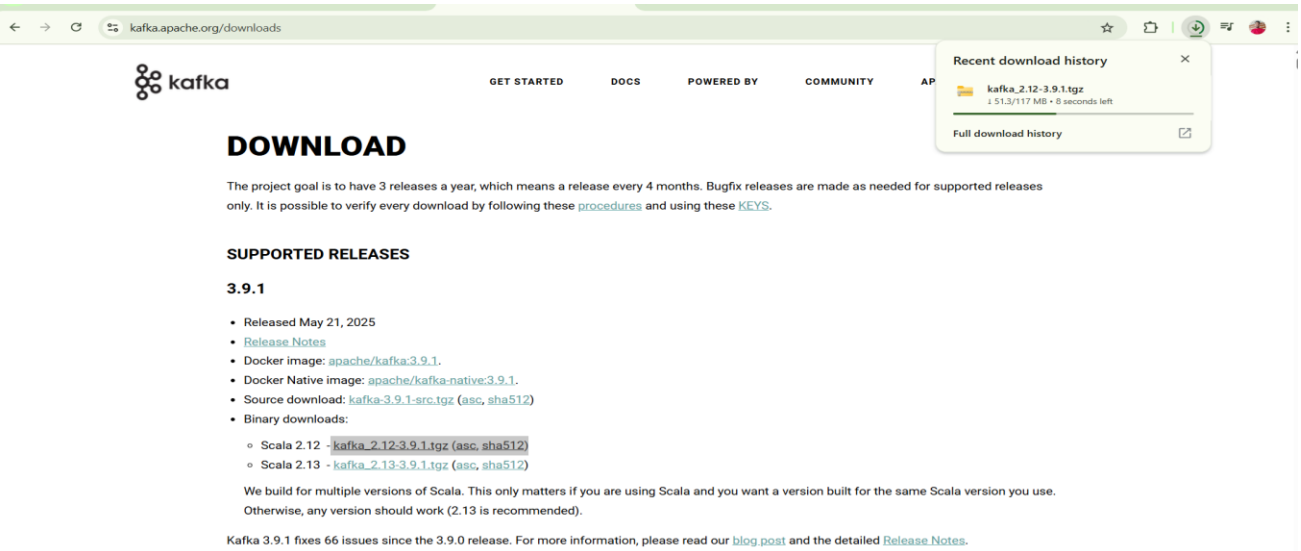
Kafka Installation :

Kafka Installation

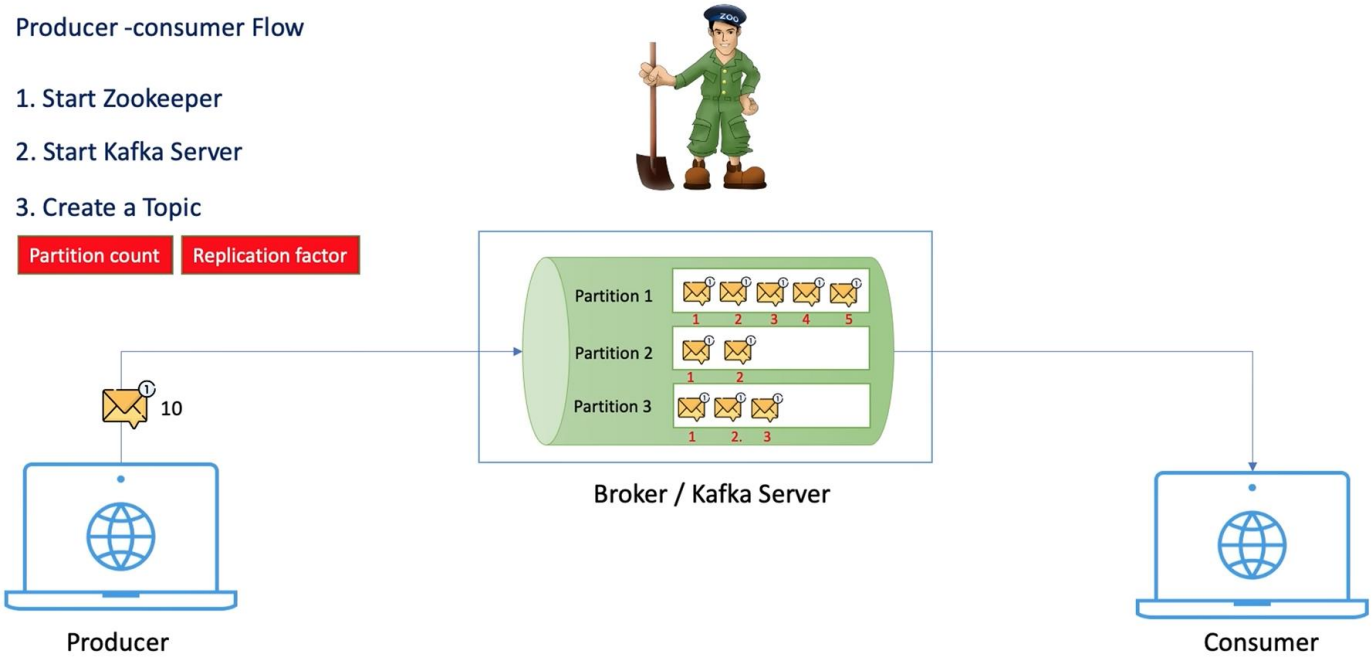
Open Source : Apache Kafka

Commercial distribution : Confluent Kafka

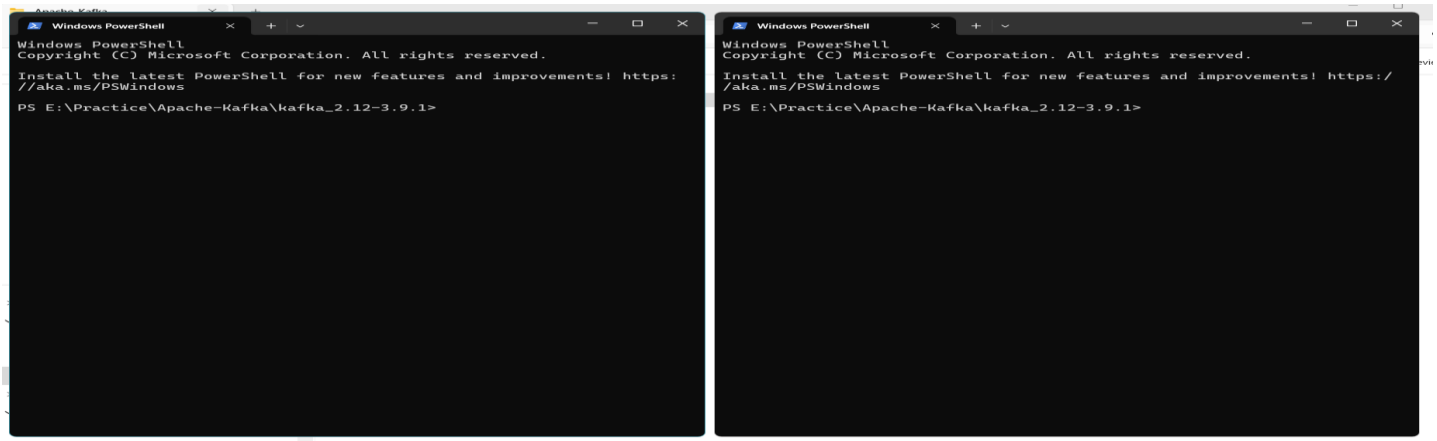
Managed Kafka service : confluent & AWS



Kafka – CLI :



➔ **Just open two terminals**



First, we need to start Zookeeper:

E:\Kafka\kafka_2.12-3.9.1>bin\windows\zookeeper-server-start.bat config\zookeeper.properties

Start Kafka Server/Broker:

E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-server-start.bat config\server.properties

Create Topics:

```
E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic topic2 --partitions 3 --replication-factor 1
Created topic topic2.

E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --list topics
topic1
topic2

E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --describe topic1
Topic: topic1   TopicId: SKke5fS1RKaTM_2qIsBLig PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: topic1   Partition: 0    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A
        Topic: topic1   Partition: 1    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A
        Topic: topic1   Partition: 2    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A
Topic: topic2   TopicId: VKJnb_h_SMKmJ6UmZ8fRrA PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: topic2   Partition: 0    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A
        Topic: topic2   Partition: 1    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A
        Topic: topic2   Partition: 2    Leader: 0       Replicas: 0     Isr: 0  Elr: N/A        LastKnownElr: N/A

E:\Kafka\kafka_2.12-3.9.1>
```

Start Producer CLI: Like sender

E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic topic1

```
E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic topic1
>Hi
>Naveen
>Hru
>Are you oka
>Thank you
>Yes
>
```

Start Consumer CLI: Like receiver

E:\Kafka\kafka_2.12-3.9.1>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic topic1

```
E:\Kafka\kafka_2.12-3.9.1\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic1 --from-beginning
Hi
Naveen
Hru
Are you oka
Thank you
Yes
```

Producer & Consumer Spring Boot example application:

Step 1: **Start zookeeper server CLI.**

Step 2: **Start Kafka-server CLI.**

Step 3: **Create Producer application.**

Pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

Project Structure:

- Kafka-Producer [boot] [Practice master]
  - src/main/java
    - com.producer
    - com.producer.controller
      - EventMessageController.java
    - com.producer.service
      - KafkaMessagePublisher.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-17]
  - Maven Dependencies
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

Application.properties :

application.properties

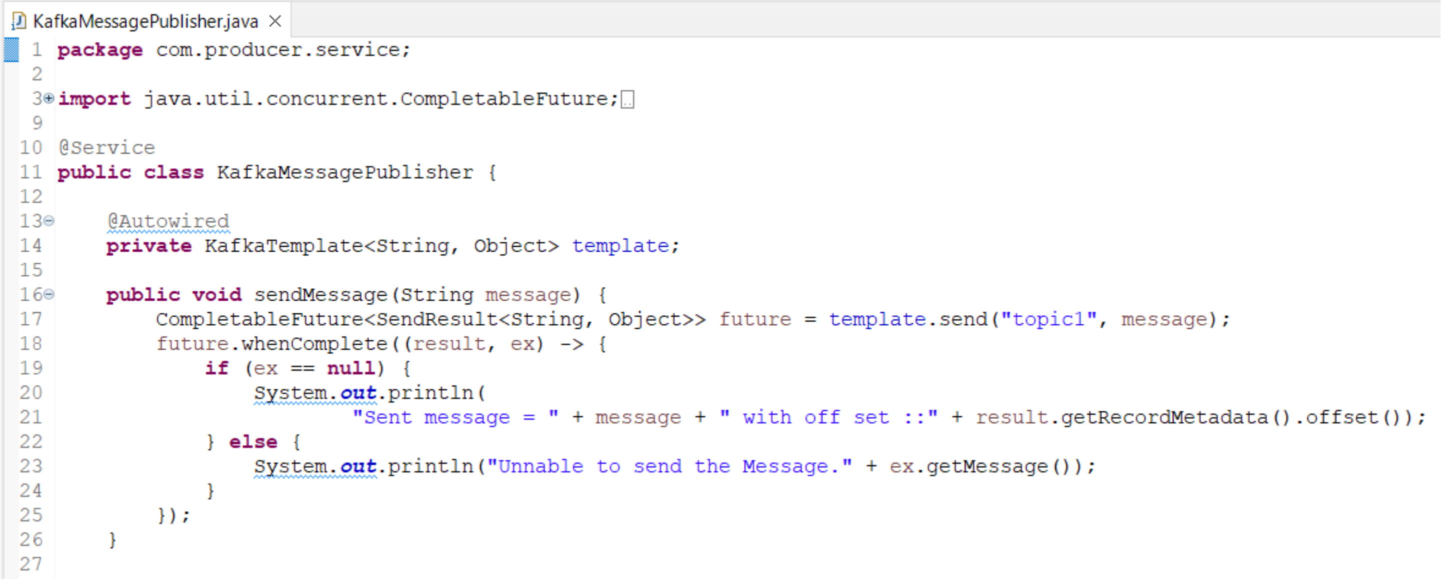
```
spring.application.name=Kafka-Producer

server.port=9797

spring.kafka.producer.bootstrap-servers=localhost:9092

```

Producer Controller:

EventMessageController.java

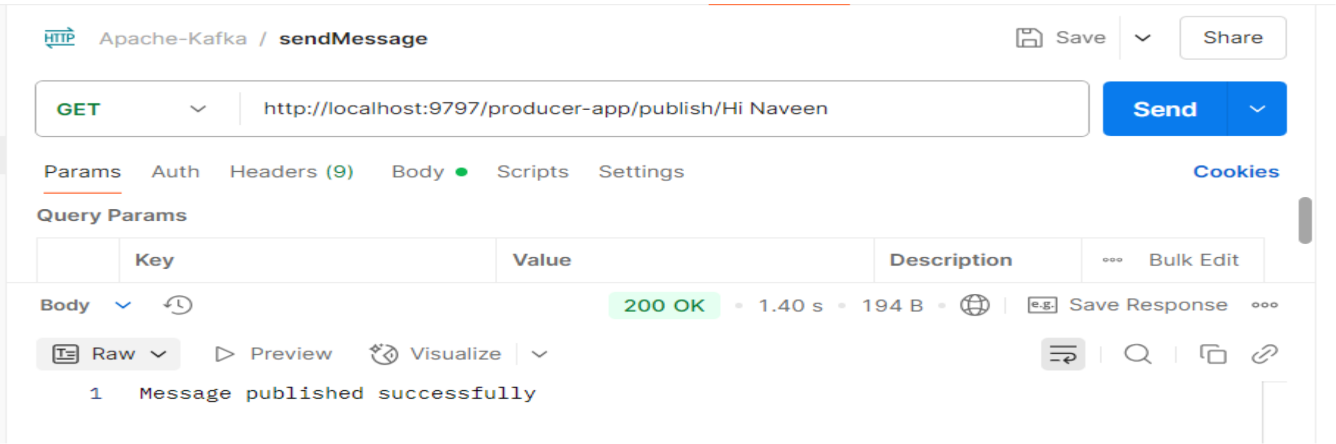
```
package com.producer.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@RequestMapping("/producer-app")
public class EventMessageController {

    @Autowired
    private KafkaMessagePublisher kafkaMessagePublisher;

    @GetMapping("/publish/{messaage}")
    public ResponseEntity<?> publishMessage(@PathVariable String messaage) {
        try {
            for (int i = 0; i < 10000; i++) {
                kafkaMessagePublisher.sendMessage(messaage);
            }
            return ResponseEntity.ok("Message published successfully");
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

}
```

Producer Service:



```java
package com.producer.service;

import java.util.concurrent.CompletableFuture;

@Service
public class KafkaMessagePublisher {

    @Autowired
    private KafkaTemplate<String, Object> template;

    public void sendMessage(String message) {
        CompletableFuture<SendResult<String, Object>> future = template.send("topic1", message);
        future.whenComplete((result, ex) -> {
            if (ex == null) {
                System.out.println(
                    "Sent message = " + message + " with off set ::" + result.getRecordMetadata().offset());
            } else {
                System.out.println("Unnable to send the Message." + ex.getMessage());
            }
        });
    }

```

Postman:

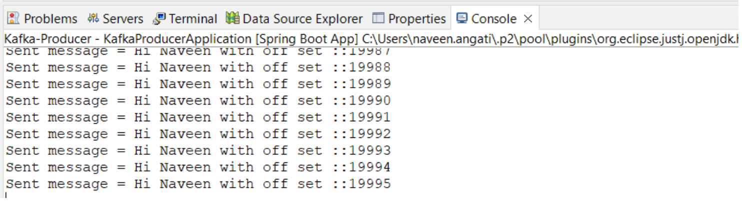


GET http://localhost:9797/producer-app/publish/Hi Naveen

200 OK • 1.40 s • 194 B

```
1   Message published successfully
```

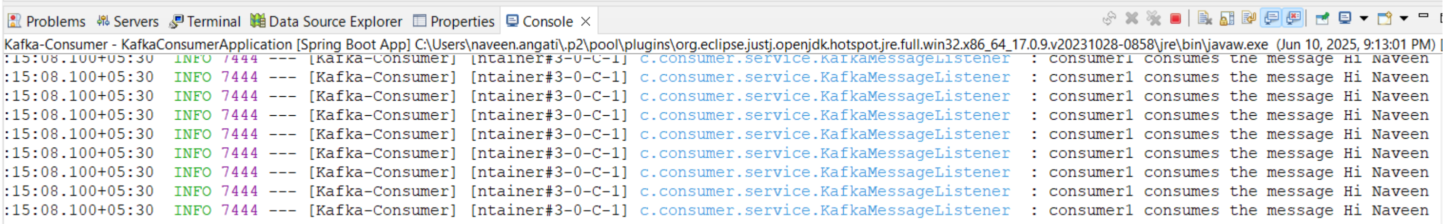
Producer Console:



Kafka-Producer - KafkaProducerApplication [Spring Boot App] C:\Users\naveen.angati\.p2\pool\plugins\org.eclipse.justj.openjdk.

```
Sent message = Hi Naveen with off set ::19987
Sent message = Hi Naveen with off set ::19988
Sent message = Hi Naveen with off set ::19989
Sent message = Hi Naveen with off set ::19990
Sent message = Hi Naveen with off set ::19991
Sent message = Hi Naveen with off set ::19992
Sent message = Hi Naveen with off set ::19993
Sent message = Hi Naveen with off set ::19994
Sent message = Hi Naveen with off set ::19995
```

Consumer Console:



Kafka-Consumer - KafkaConsumerApplication [Spring Boot App] C:\Users\naveen.angati\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Jun 10, 2025, 9:13:01 PM)

```
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
:15:08.100+05:30  INFO 7444 --- [Kafka-Consumer] [ntainer#3-0-C-1] c.consumer.service.KafkaMessageListener  : consumer1 consumes the message Hi Naveen
```

When we are sending Object or Other type of data to Kafka server we need to write these properties :

1.Producer Application:

```
spring.application.name=Kafka-Producer

server.port=9797

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

```
@PostMapping
public ResponseEntity<?> publishMessage(@RequestBody Consumer consumer) {
    try {
        //for (int i = 0; i < 10000; i++) {
            kafkaMessagePublisher.sendMessage(consumer);
        //}
        return ResponseEntity.ok("Message published successfully");
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```

2.Consumer Application:

application.properties

```
spring.application.name=Kafka-Consumer

server.port=8055

spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=group1

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

# Optional (recommended for POJOs)
spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

🎯 Purpose:

These configurations control how Kafka messages are serialized and deserialized (converted to and from bytes) when sending and receiving.

📩 Producer Properties

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

🔹 What they do:

- key-serializer: Converts the message key (usually a string like a user ID) into bytes.
- value-serializer: Converts your Java object (POJO) into JSON, then into bytes — using JsonSerializer.

✅ So when your Spring app sends messages, the value is serialized as JSON.

📥 Consumer Properties:

spring.kafka.consumer.key-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=*

🔹 **What they do:**

- **key-deserializer:** Converts the key from bytes to String.
- **value-deserializer:** Converts the **JSON** (byte array) back into your Java object **(POJO).**
- **trusted.packages**=*: Tells Spring that it's okay to deserialize **JSON** into any class (useful for avoiding deserialization security issues).

✅ So when your Spring app receives messages, it turns the **JSON** back into a Java object.

Disadvantages of Kafka:

1.Complex Setup and Management

2.Kafka requires multiple components: **Brokers**, **ZooKeeper** (or KRaft), Topic configurations, etc.

3.Managing partitions, replication, and scaling is not beginner friendly.

4.Until **Kafka 2.8**, **ZooKeeper** is mandatory.

5.**ZooKeeper** adds extra infrastructure and can be a single point of failure if not handled properly.

6.**Kafka** guarantees message order only within a partition.

7.If your **topic** has multiple **partitions**, you can lose global **ordering**.

Disadvantages:

1.**High Throughput** - Can handle millions of messages per second, even with limited hardware.

2.**Scalability** - Kafka easily scales horizontally by adding more brokers or partitions.

3.**Fault Tolerance** - Kafka automatically recovers from failures

4.**Decoupling of Services** - Enables asynchronous, event-driven architecture — producers and consumers don't depend on each other.

5.**Multiple Consumers Can Read Same Data** - Many services can independently consume the same messages without conflict.