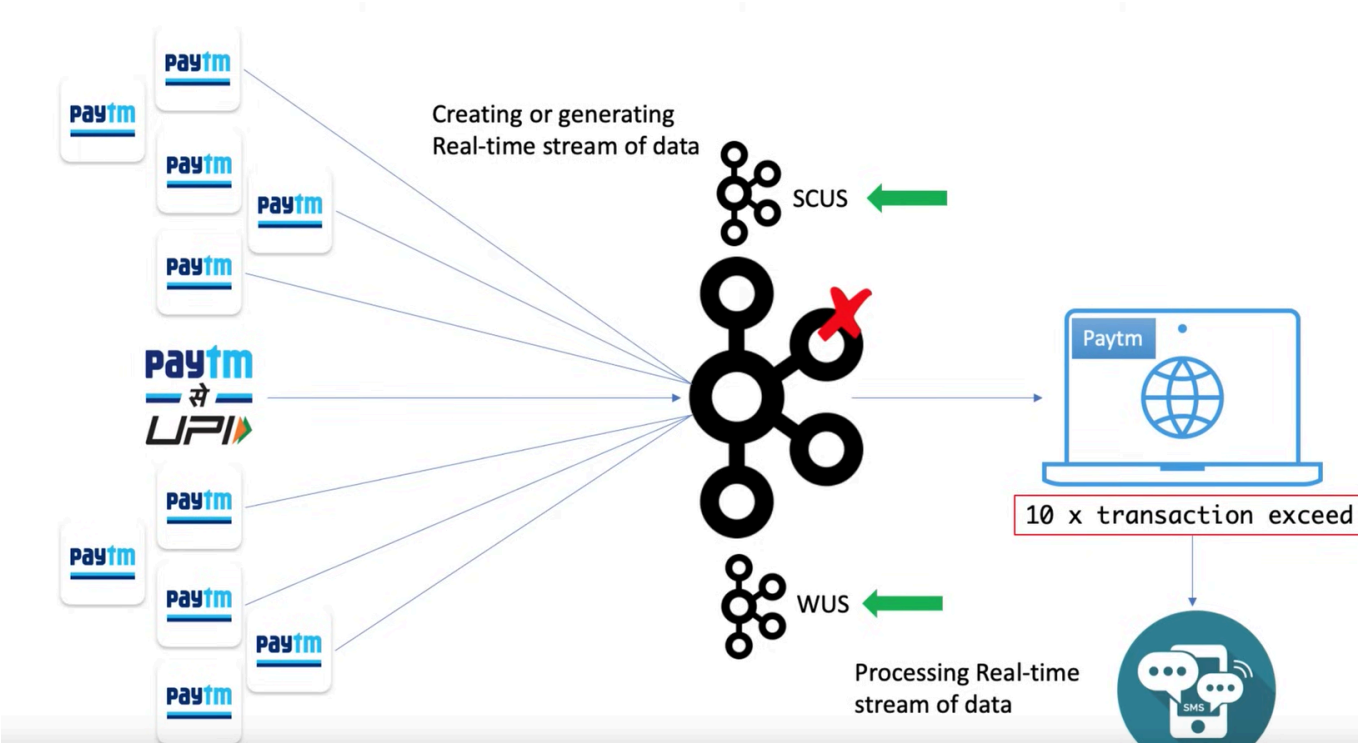
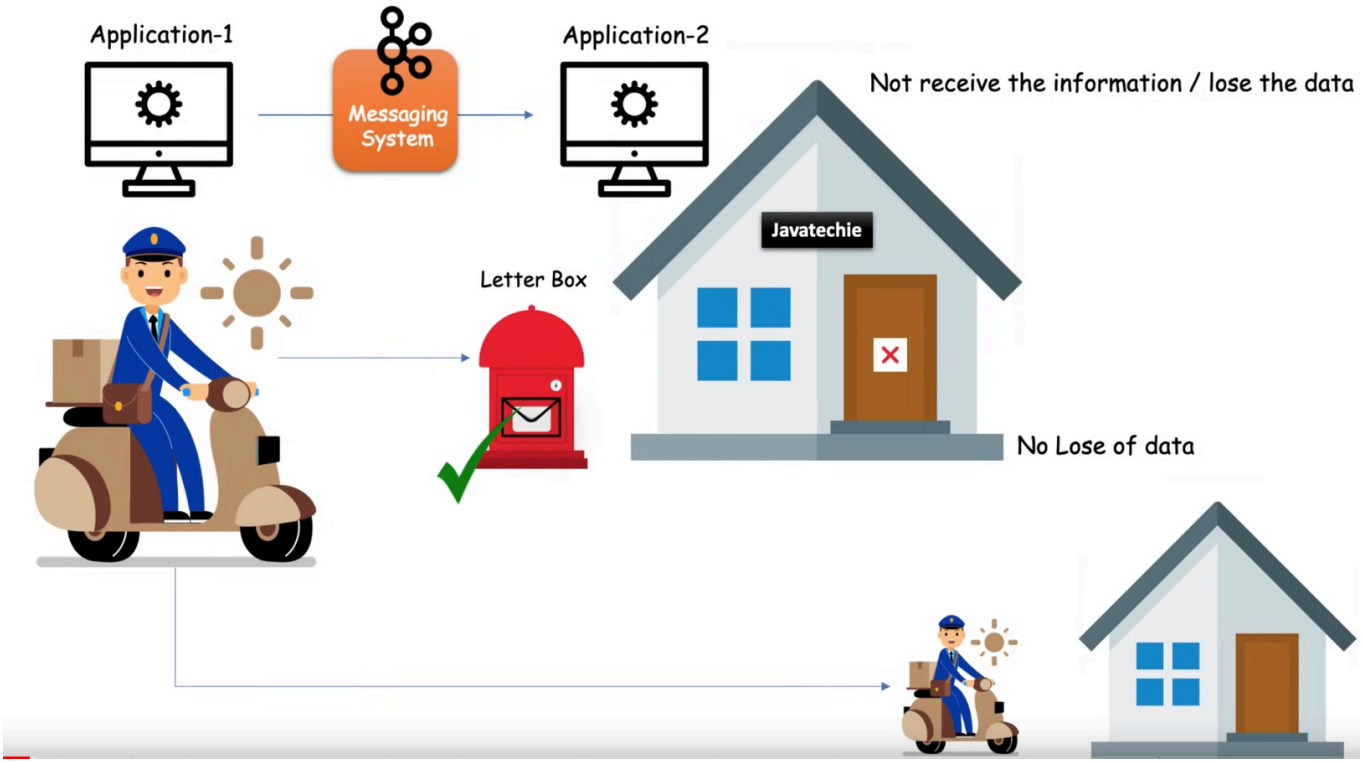
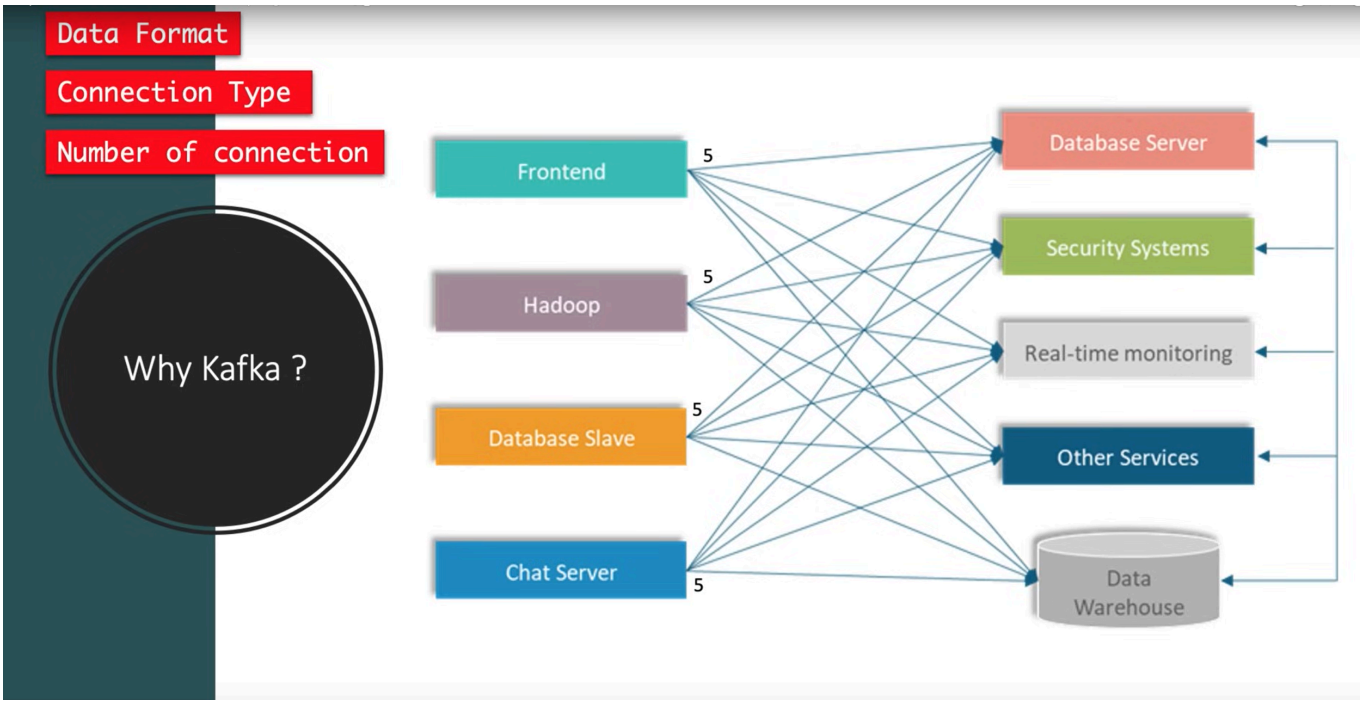
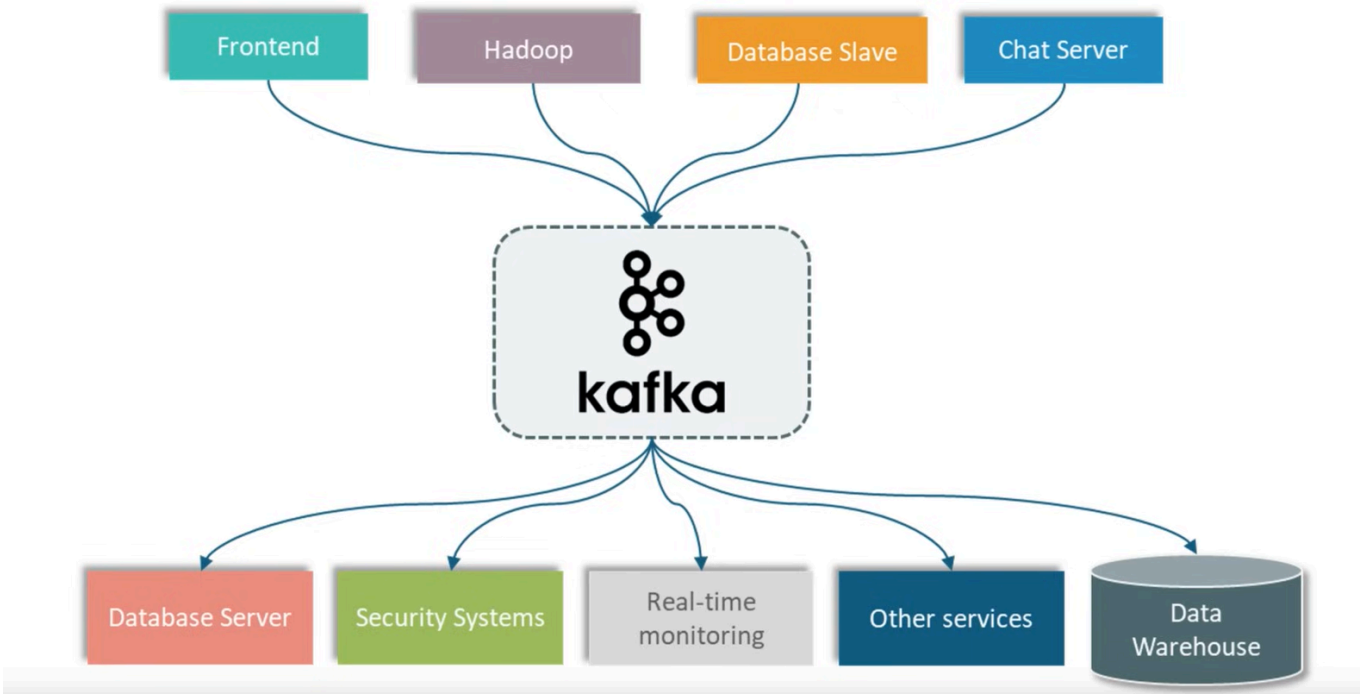
Kafka Notes



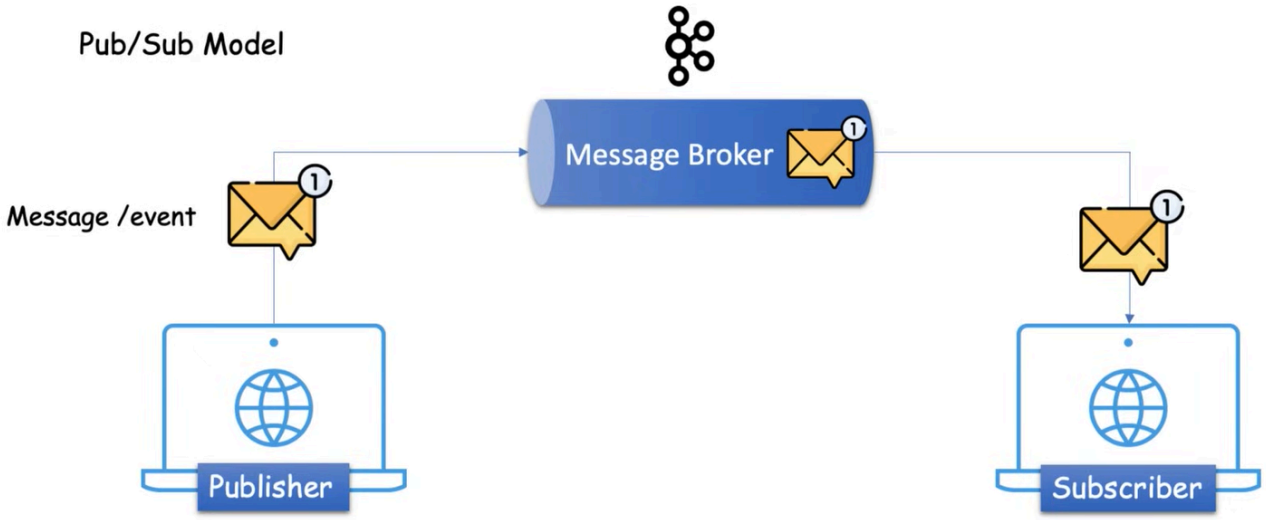
built by LinkedIn




Data Format
Connection Type
Number of connection
Why Kafka ?
Frontend
Hadoop
Database Slave
Chat Server
5
5
5
5
Database Server
Security Systems
Real-time monitoring
Other Services
Data Warehouse

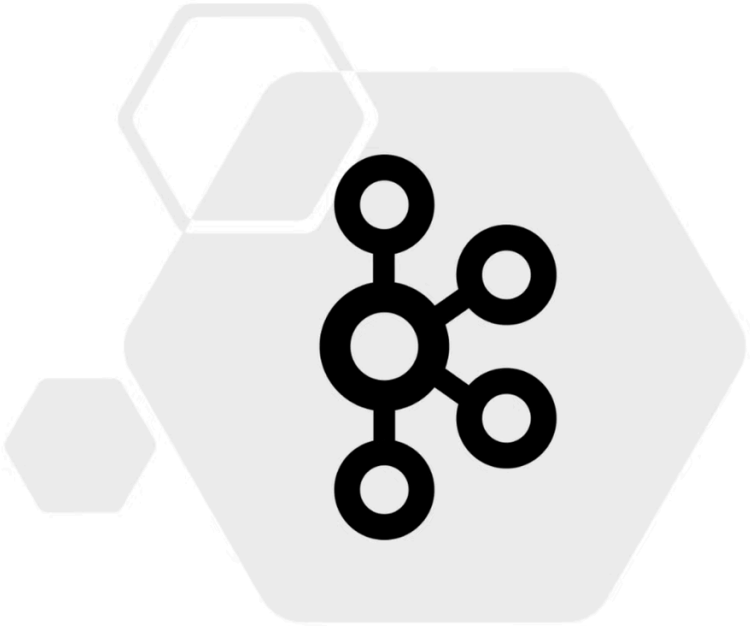

Frontend
Hadoop
Database Slave
Chat Server
kafka
Database Server
Security Systems
Real-time monitoring
Other services
Data Warehouse

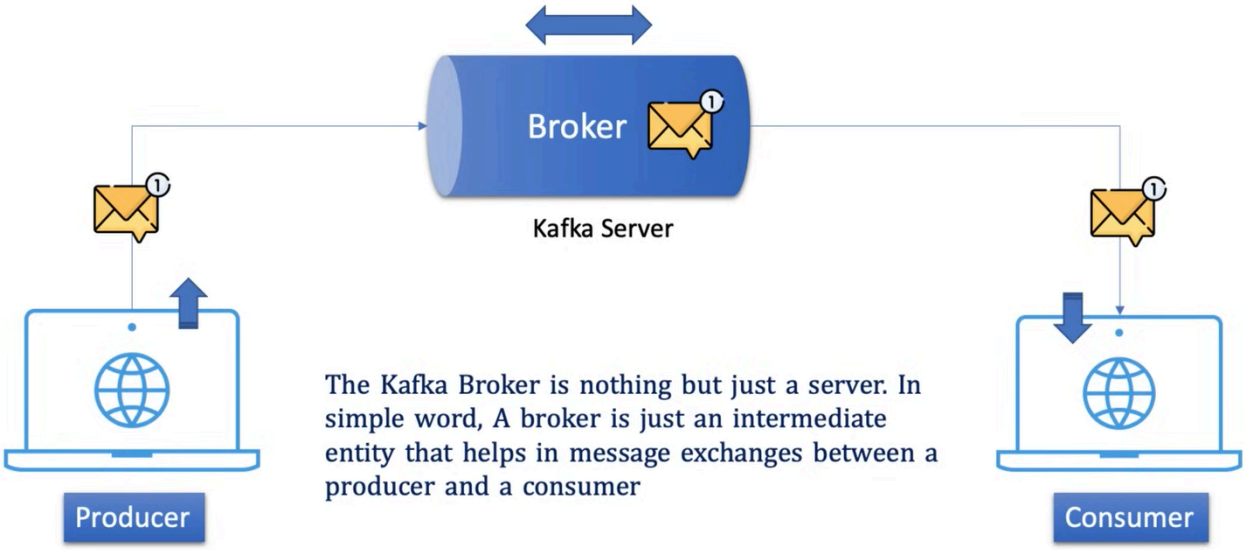
How does it work (High-level overview)



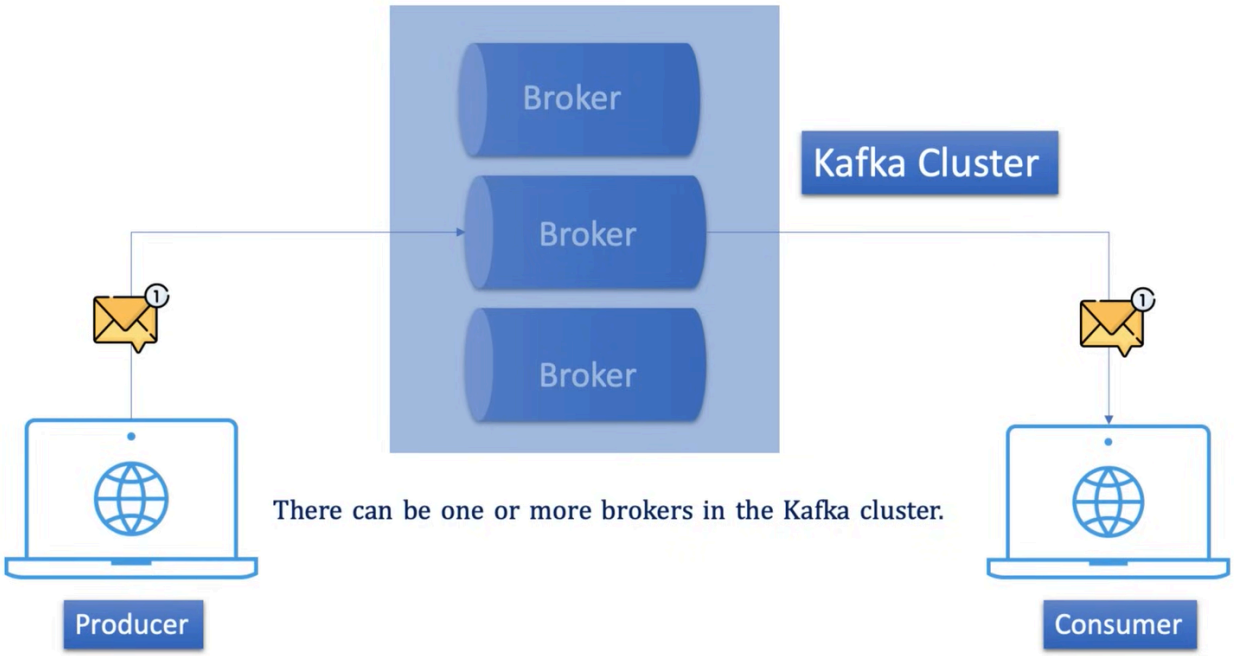
Kafka Components

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

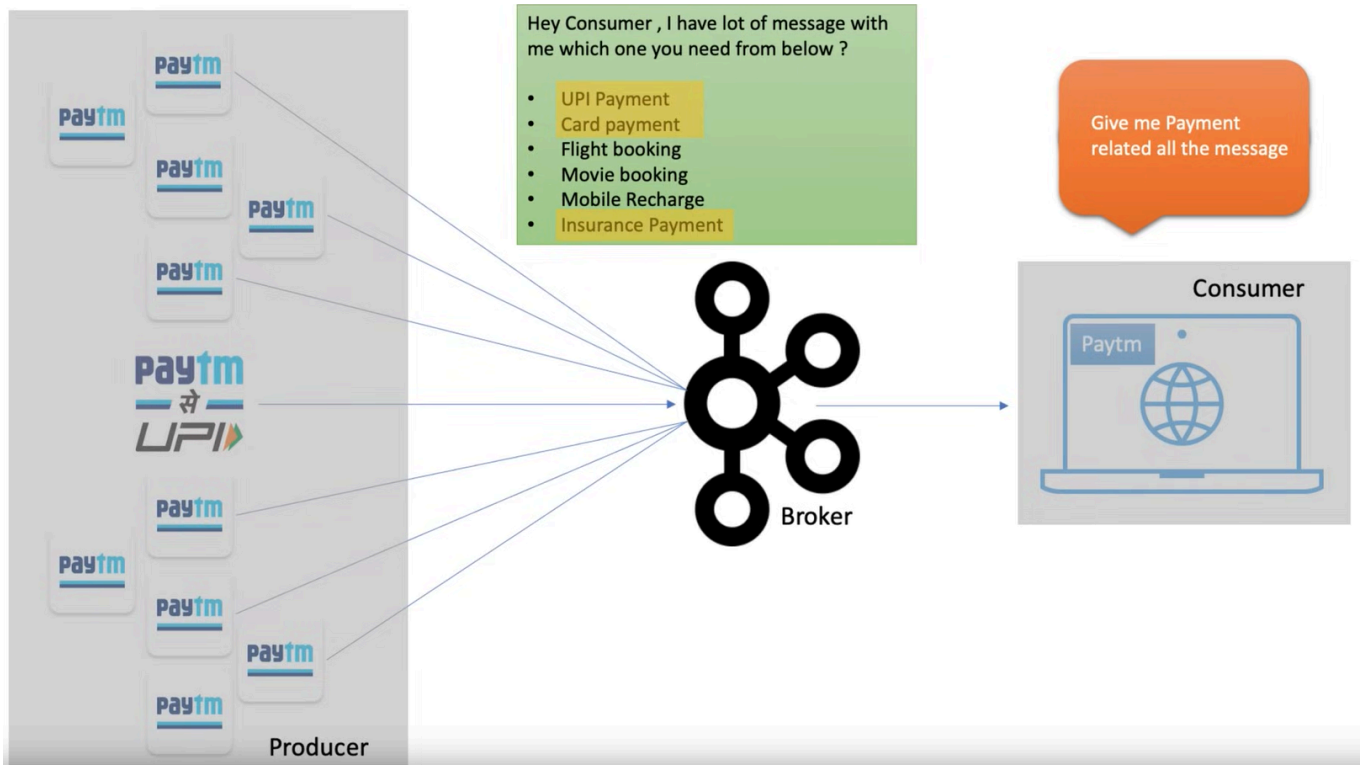


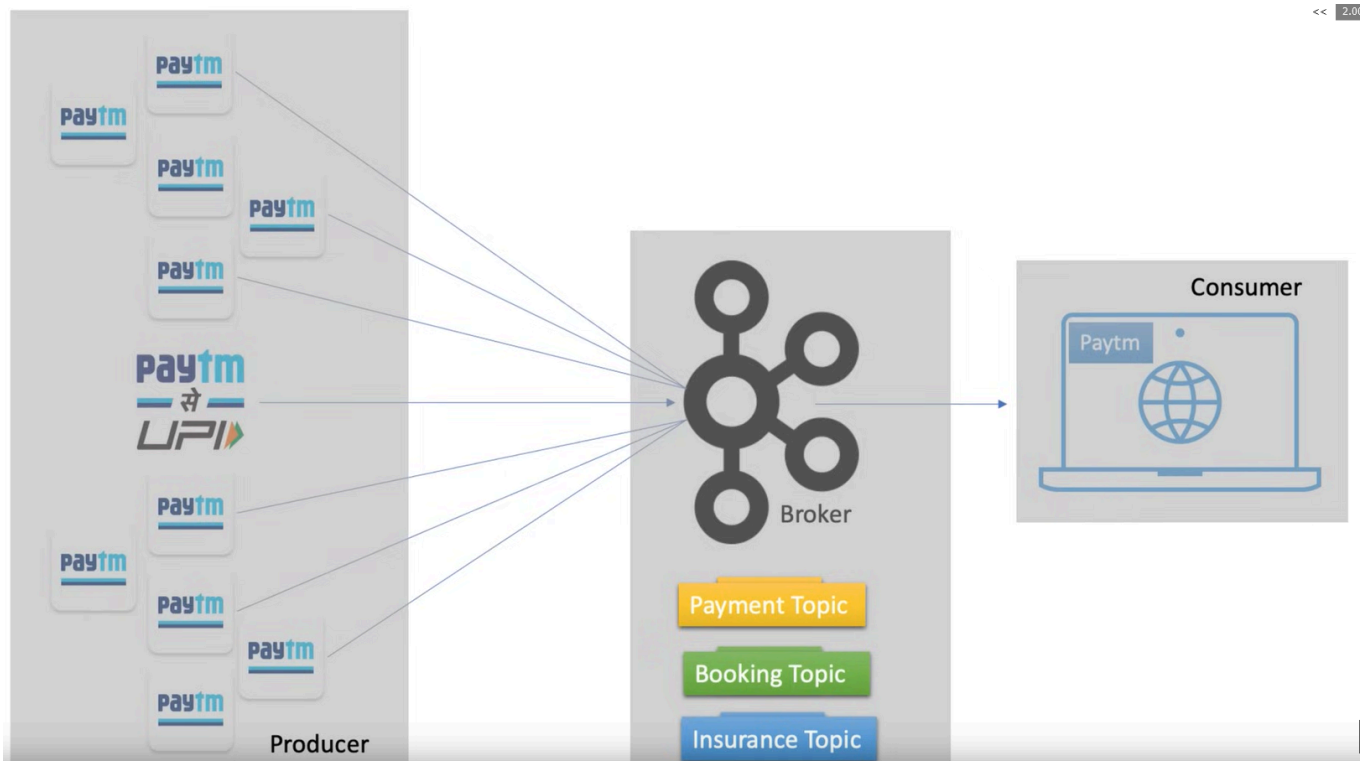


The Kafka Broker is nothing but just a server. In simple word, A broker is just an intermediate entity that helps in message exchanges between a producer and a consumer



There can be one or more brokers in the Kafka cluster.


Hey Consumer , I have lot of message with me which one you need from below ?
UPI Payment
Card payment
Flight booking
Movie booking
Mobile Recharge
Insurance Payment
Give me Payment related all the message
Consumer
Paytm
Broker
Producer

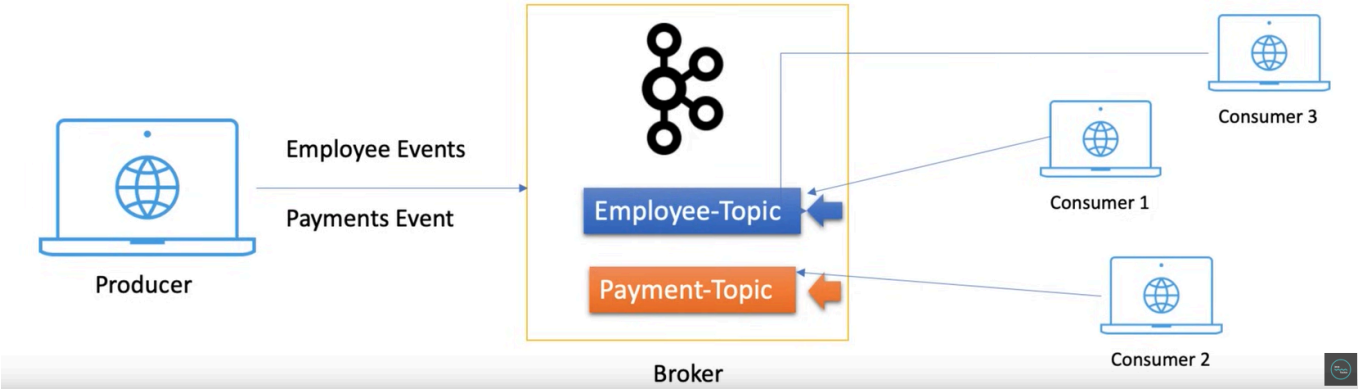

Consumer
Paytm
Broker
Payment Topic
Booking Topic
Insurance Topic
Producer

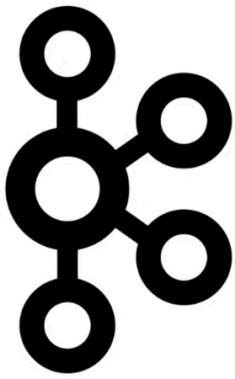
EMPLOYEE_TABLE

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

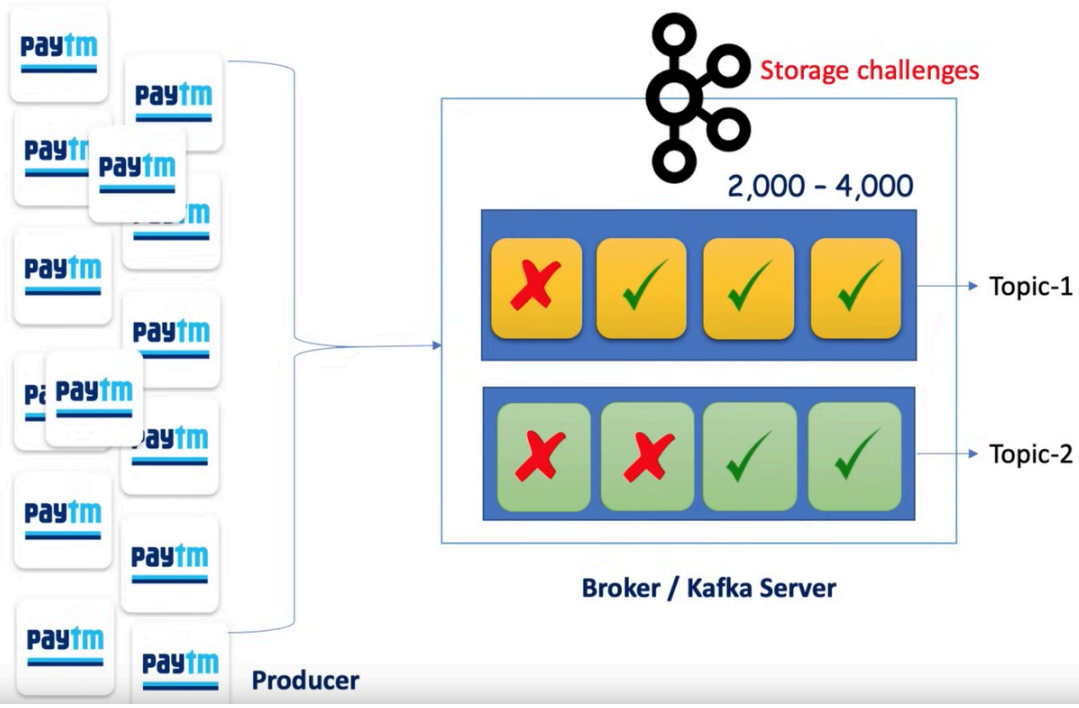




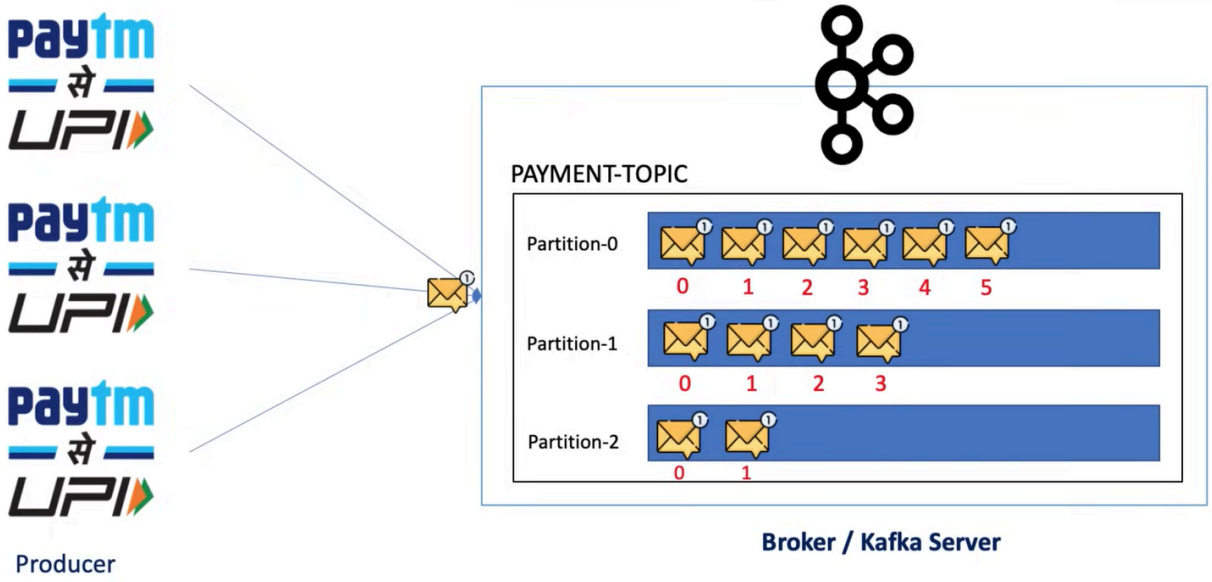
TOPIC

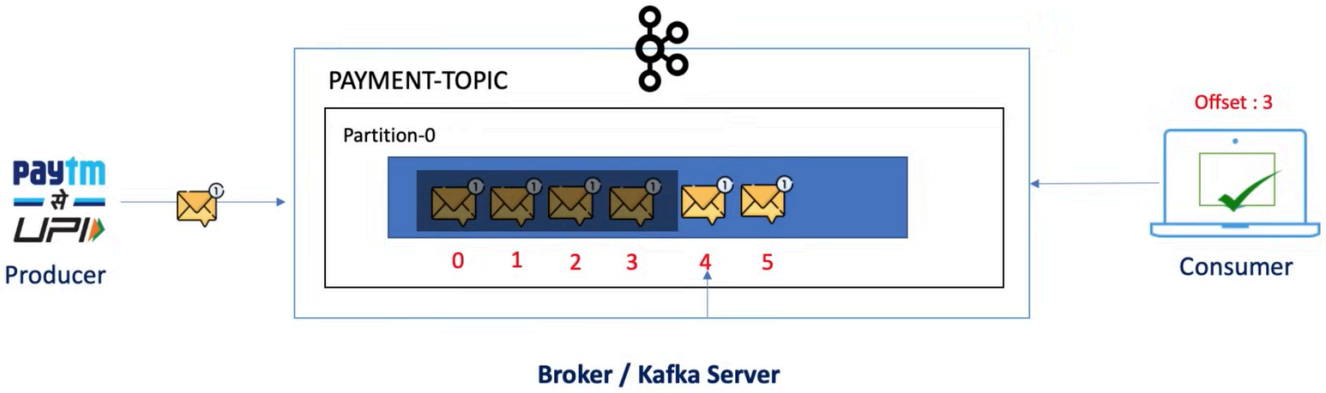
It specifies the category of the message or the classification of the message. Listeners can then just respond to the messages that belong to the topics they are listening on.

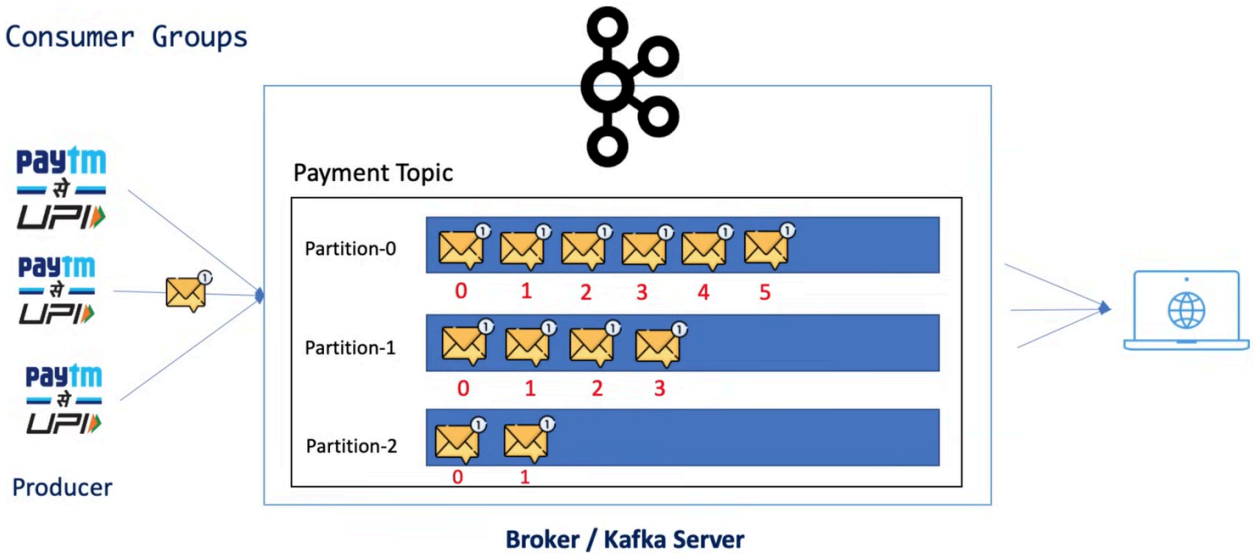
Partitions

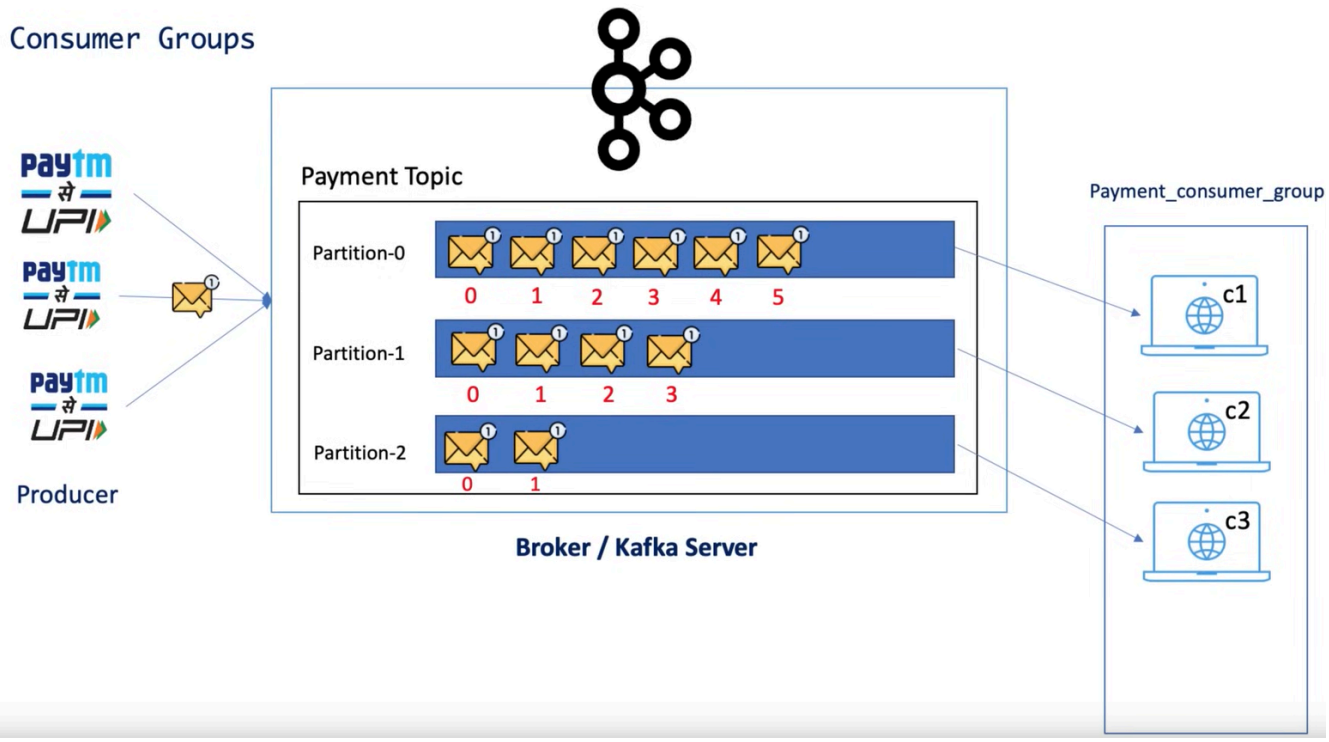


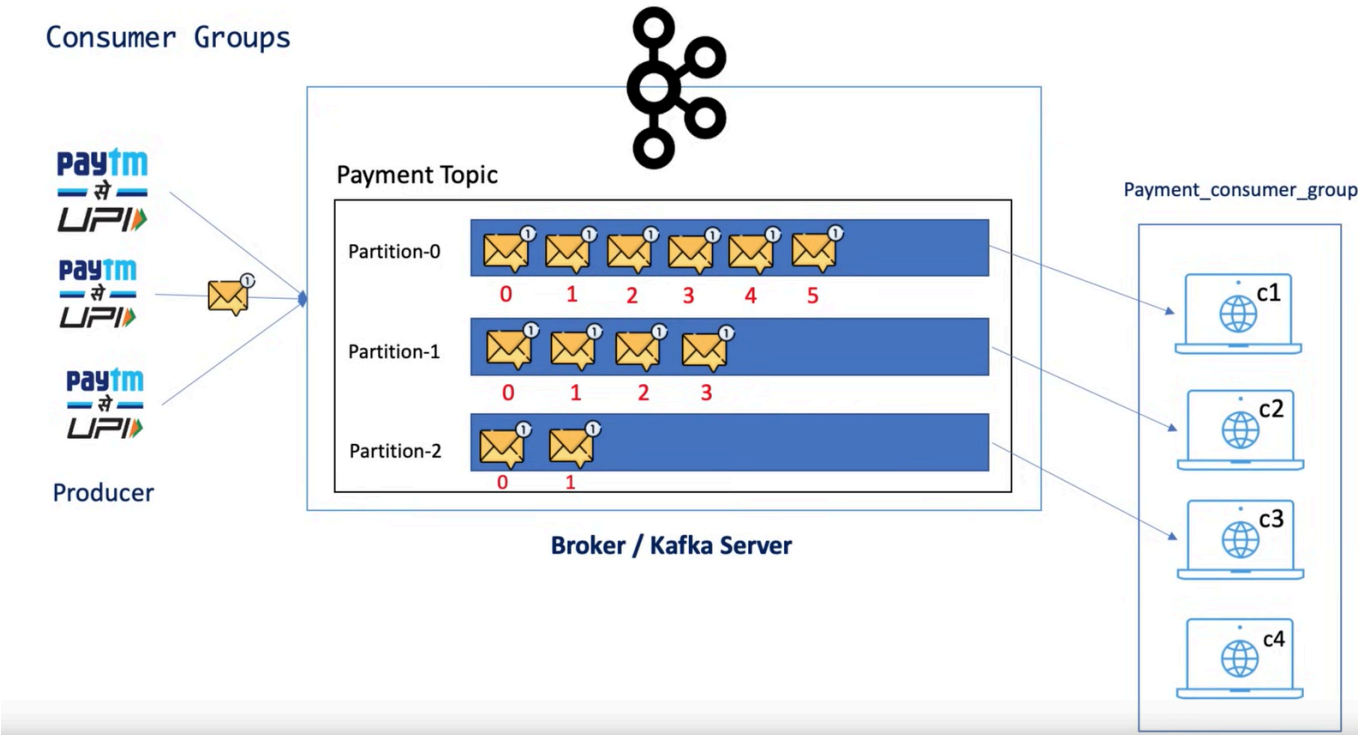
In Kafka, a sequence number is assigned to each message in each partition of a Kafka topic. This sequence number is called Offset.




PAYMENT-TOPIC
Partition-0
0 1 2 3 4 5
Offset : 3
Producer
Consumer
Broker / Kafka Server

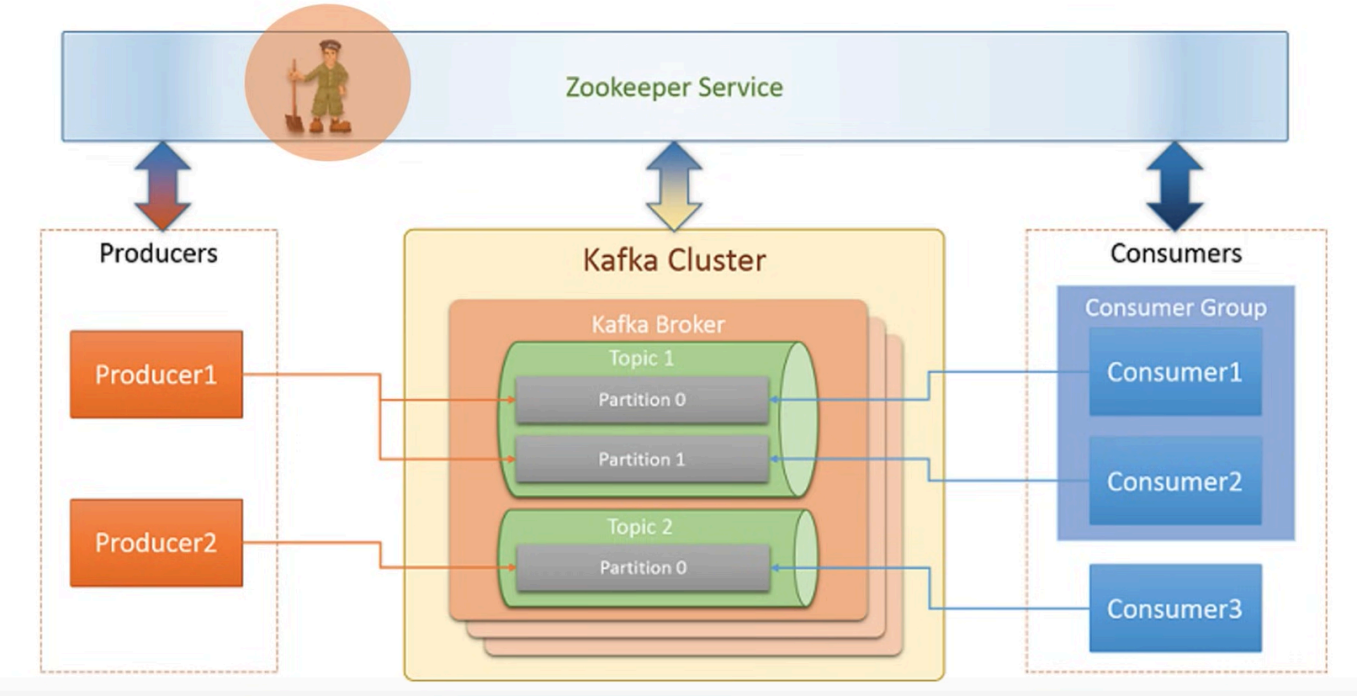

Consumer Groups
Payment Topic
Partition-0
0 1 2 3 4 5
Partition-1
0 1 2 3
Partition-2
0 1
Producer
Broker / Kafka Server


Consumer Groups
Payment Topic
Partition-0
0 1 2 3 4 5
Partition-1
0 1 2 3
Partition-2
0 1
Payment_consumer_group
c1
c2
c3
Producer
Broker / Kafka Server



As the name suggests, the Kafka Consumer group is a group of consumers. Multiple consumers combined to share the workload. It is just like dividing a piece of large task among multiple individuals

Zookeeper is a prerequisite for Kafka. Kafka is a distributed system, and it uses Zookeeper for coordination and to track the status of Kafka cluster nodes. It also keeps track of Kafka topics, partitions, offsets, etc.

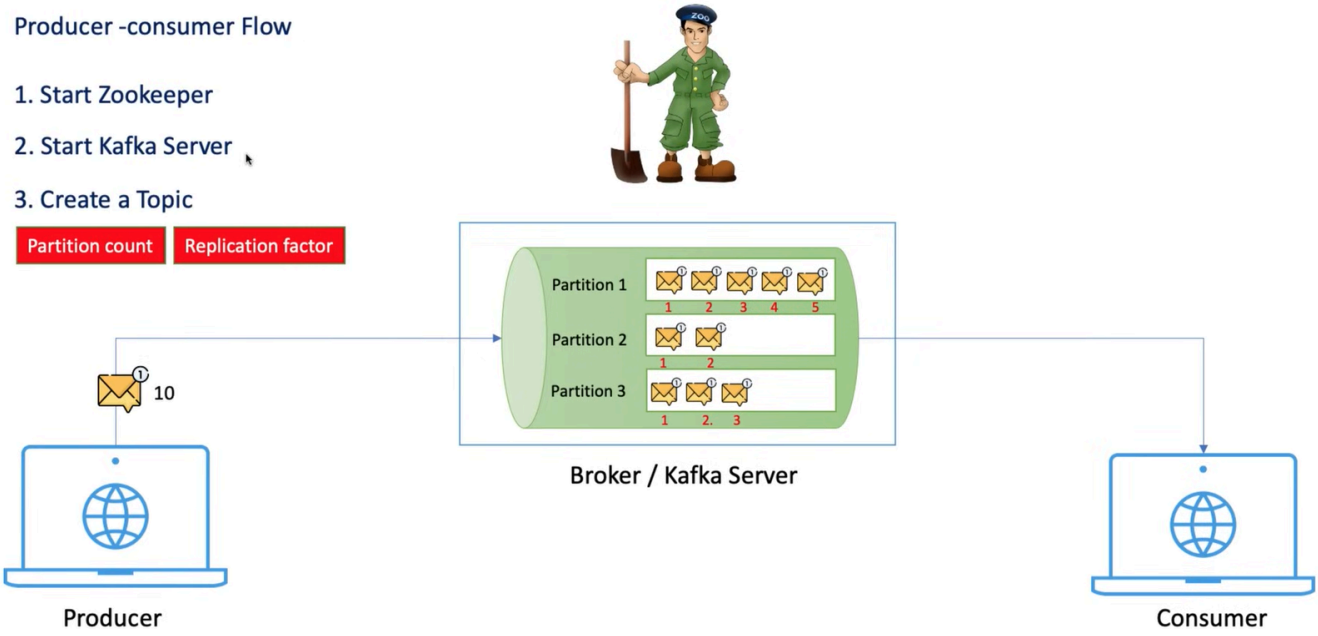


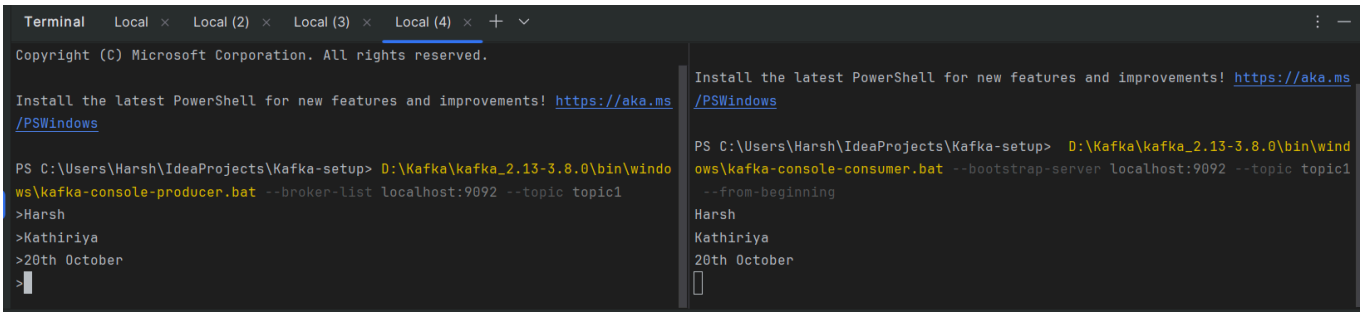
Kafka Installation

Open Source : Apache Kafka

Commercial distribution : Confluent Kafka

Managed Kafka service : confluent & AWS





Headers In Kafka:

- metadata that can be associated with Kafka message/record.
- provides additional context about the message without altering its payload.
- allows to include extra information as key-value pairs , which can help with routing, filtering, logging and tracing messages.
- uses key-value pair structure:
  - key is String
  - value is byte[], allows to store any type of data by serializing it into bytes.
- Headers allow to:
  - Enable Message Filtering
  - Headers can provide data for routing decisions when multiple consumers process messages differently.
  - Enable Tracing and Logging
  - Add Protocol Information

```
public class ProducerRecord<K, V> {
private final String topic;
```

```
private final Integer partition;
private final Headers headers;
private final K key;
private final V value;
private final Long timestamp;
}
```

- Here, key is used to uniquely identify a message within a partition. It's optional but if provided, Kafka uses it to determine the specific partition within the topic where the message should go.
- By default, Kafka uses the hash of key to determine the partition.
- if No key is assigned, Kafka assigns the message to partitions in a round-robin manner.
- Example: In a retail application, the `key` could be a customer ID, which ensures that all messages for the same customer are stored in the same partition.
- Value is the actual content/payload of the message - this is the main data that we want to transmit.
- It could be a JSON string, a serialized object or any other details of an order.

```
// Create a ProducerRecord
ProducerRecord<String, String> record = new ProducerRecord<>(topic, key, value);

// Send the record using Kafka Producer (Assuming producer is already created and configured) producer.send(record);
```

Feature	ThreadPoolExecutor	ThreadPoolTaskExecutor
Package	`java.util.concurrent`	`org.springframework.scheduling.concurrent`
Integration	Standalone Java library	Integrated with Spring Framework
Configuration	Configured via constructor parameters	Configured using setters and Spring context
Thread Naming	No automatic naming	Automatic naming based on prefix
Use Case	Low-level concurrency management	High-level task execution in Spring apps
Ease of Use	More complex due to low-level API	Simpler, user-friendly Spring configuration

2024-10-26T14:26:57.290+05:30 INFO 3108 --- [ad | producer-1] org.apache.kafka.clients.NetworkClient : [Producer clientId=producer-1] Node -1 disconnected.

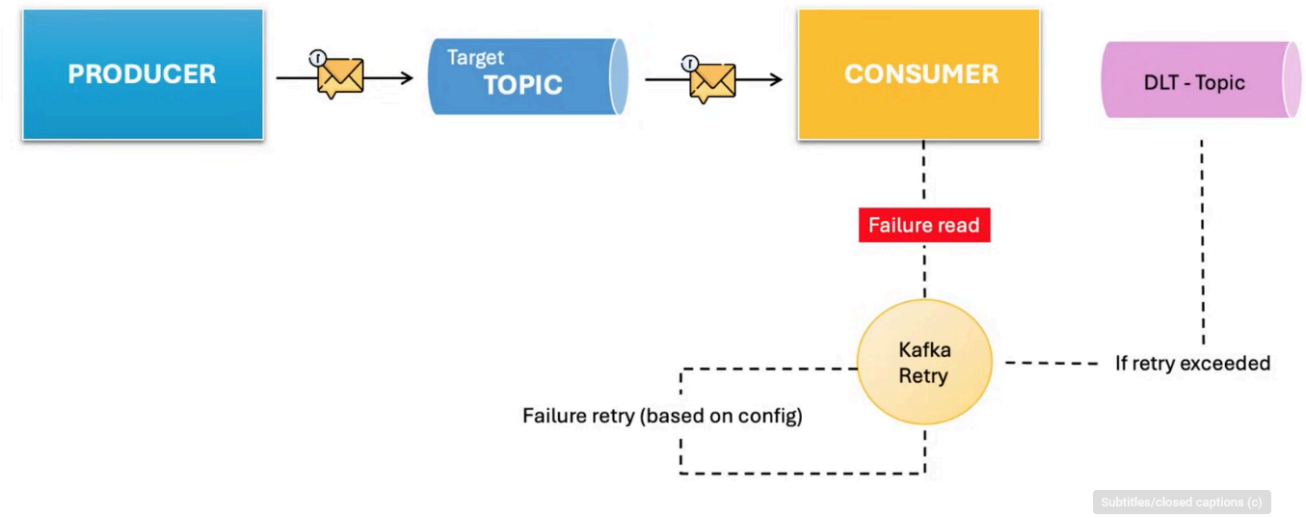
- Whenever we send message to topic, it always accepts byte array.
- when we send message over network, it must be serialized and while reading it must be deserialized.
- When we send json object to topic, Json serializer is used to serialize the object
- If we want to access secure Kafka cluster (HTTPS) , need to define all SSH related certificates in configuration file.

Message Routing with specific Partition:

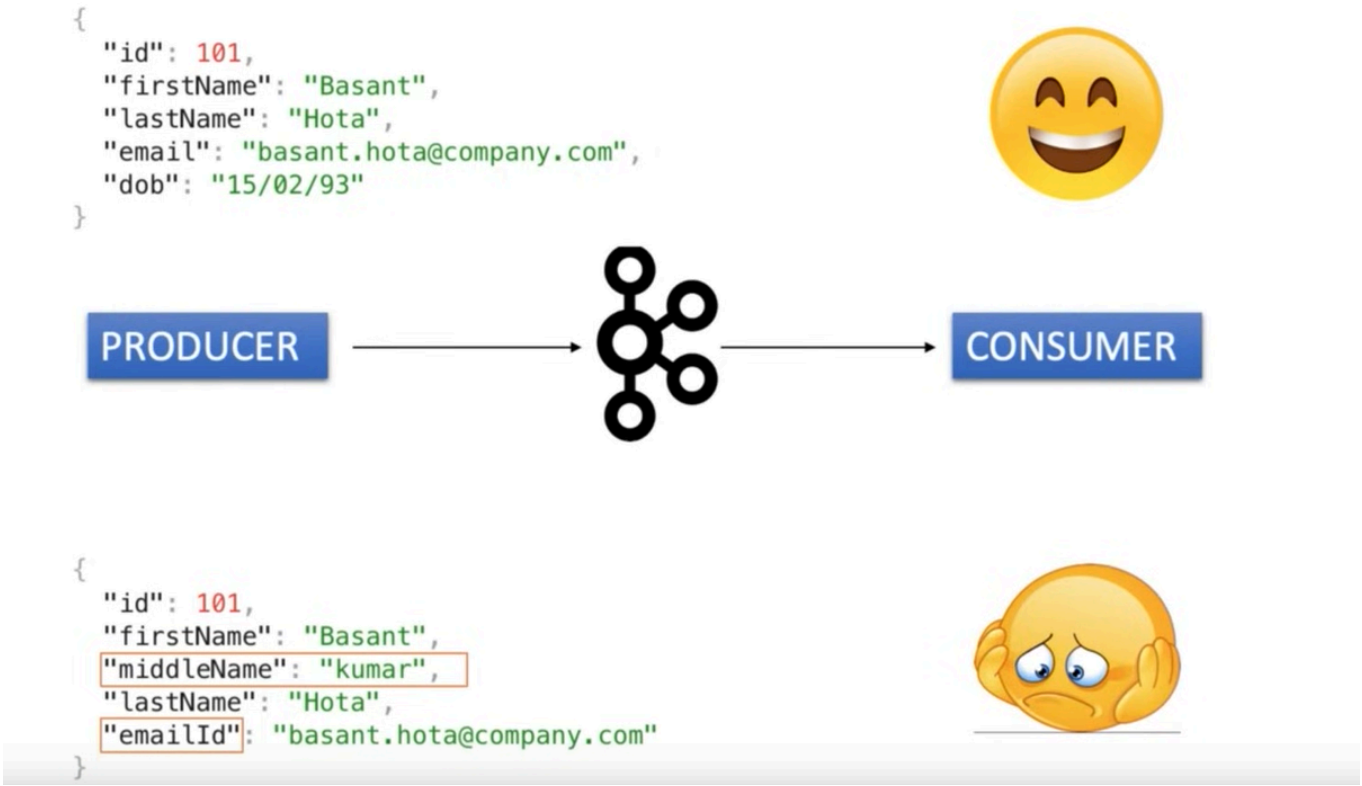
- Now, Let's say we want our producer to send all the messages to some particular partition.
- Or we want consumer to consume from only one specific partition.
- then we need to specify the partition.

Kafka E2E integration Test:

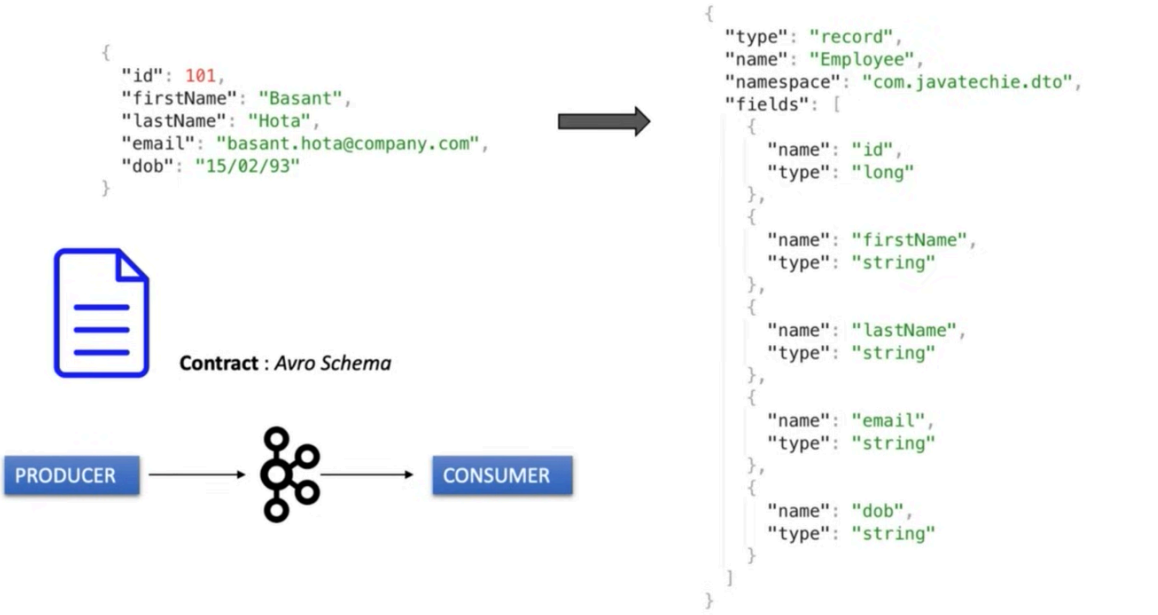
Kafka Error Handling:



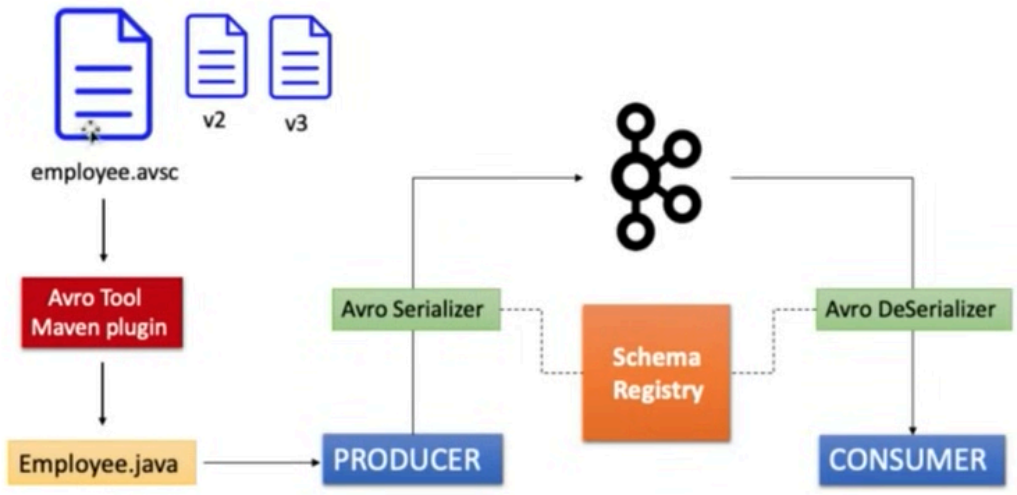
Kafka Schema Registry & Avro:



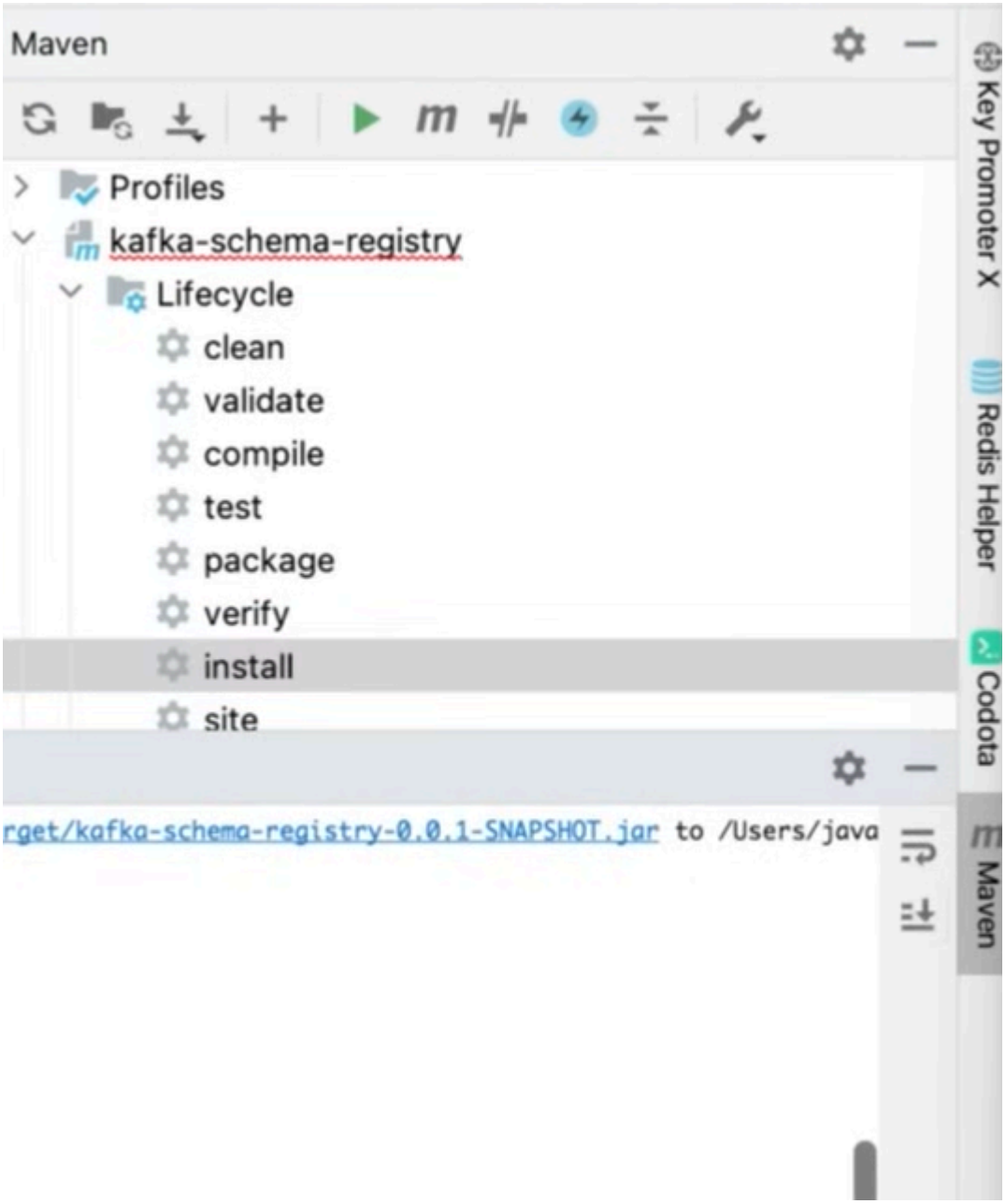
- If there is some change in data format then all consumers will get affected.
- 1st solution:
  - write new producer and consumer, so that new data will be produced and consumed by new instance without impacting existing flow.
  - but that is not a feasible solution
- 2nd solution:
  - so Kafka introduced Avro schemas and to store these schema, schema registry.
  - Avro schema: contract between producer and consumer. It represents data that we are going to serialize and deserialize.



- here, right side shown is Avro schema.
- If any changes made in the schema, it will be called new version.
- Schema Registry: store the schema, so.....
- when Avro serializer serializes the record then it first validates and stores schema into the registry.
- when Avro deserializer deserializes the record then it takes schema from the registry and validates and then deserializes it back to the object.
- versioning is supported in schema registry.

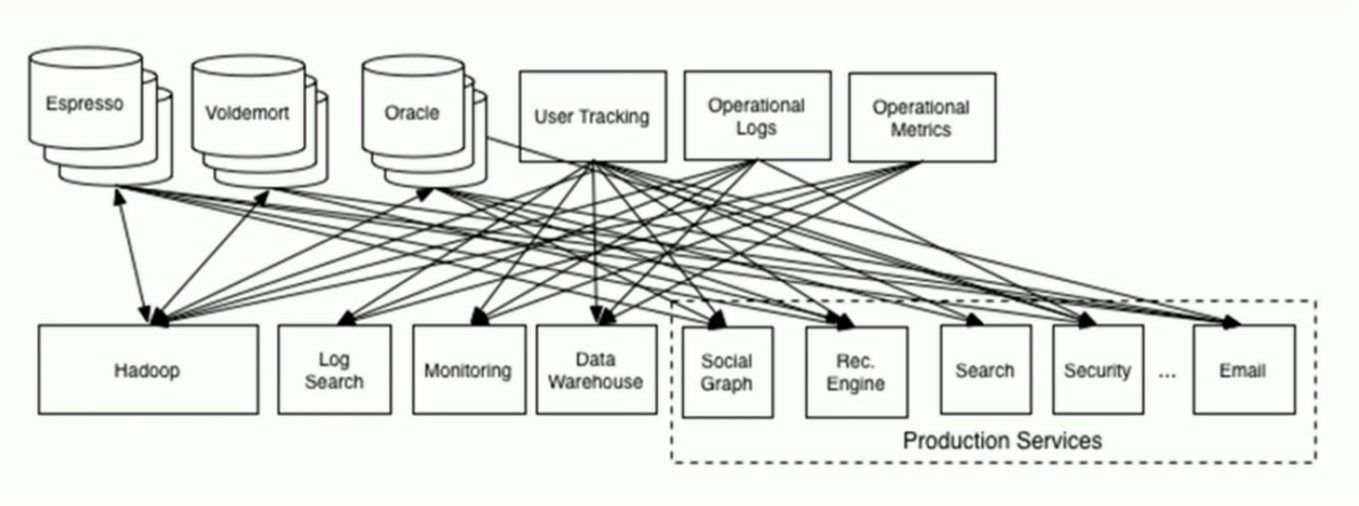


- after creating `empoyee.avsc` and adding all the dependencies, run `./gradlew generateAvro`
- which will create a avro object for defined schema.

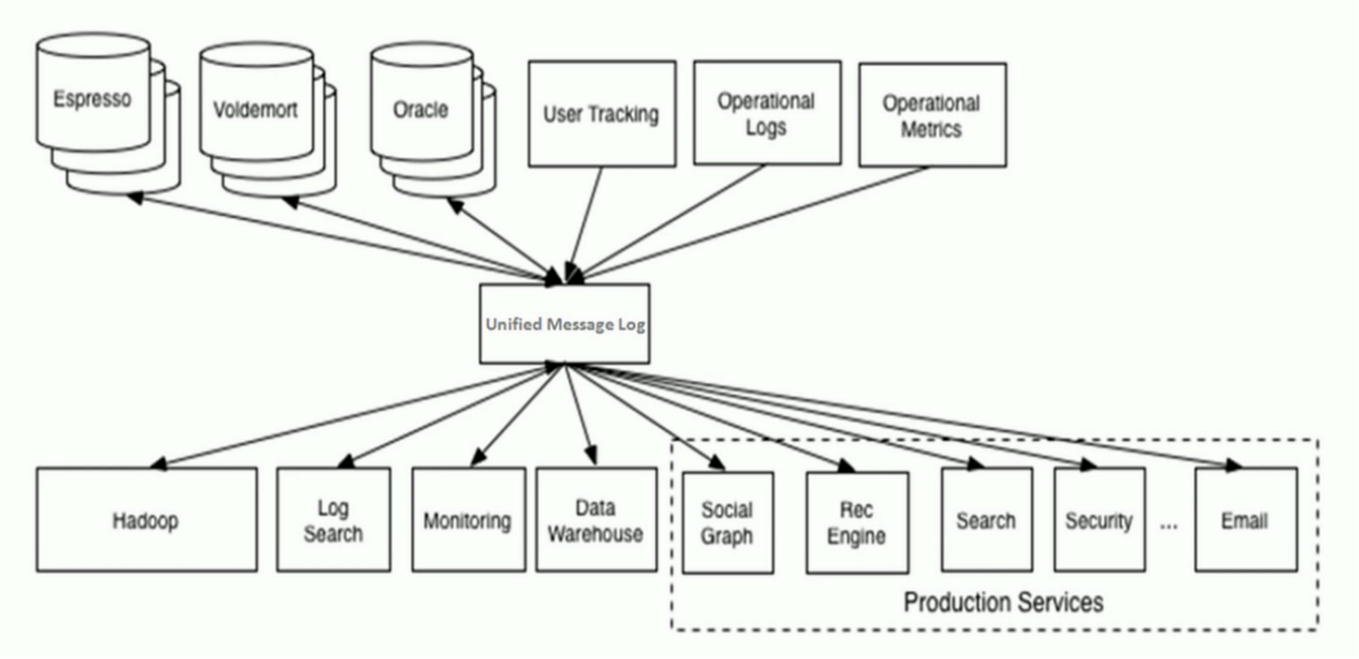


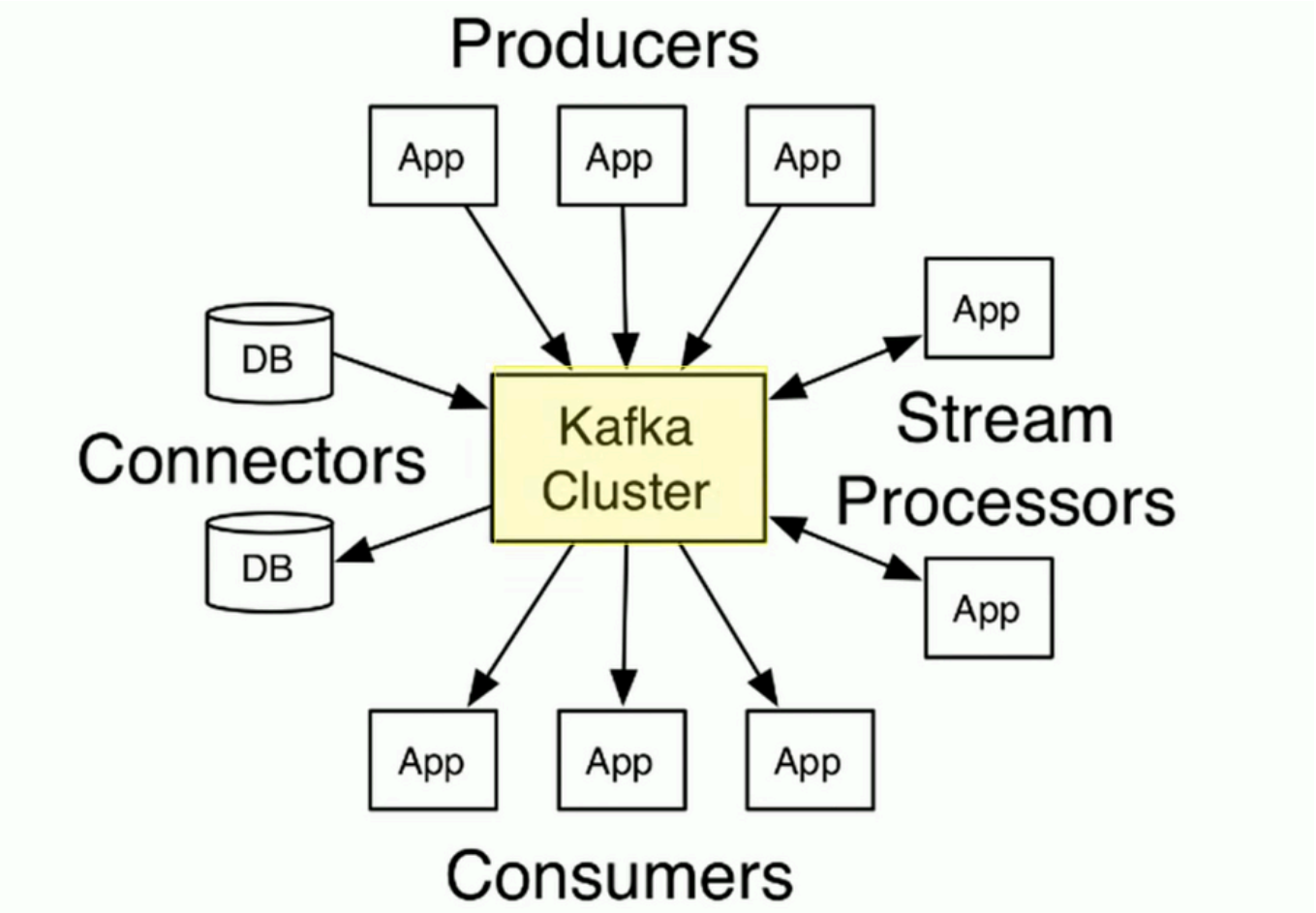
- so it will create DTO employee.java class for us that we defined in .avsc
- Initially, when we run the project for the first time, the schema wouldn't have been registered. first time, when we send a record to topic, it will be serialized using Avro schema and schema will be registered in the registry.
- from second time onwards whenever we make changes in the schema, create it's object (`./gradlew generateAvro`) stop the application, rerun.
- while adding new field, make sure to give some default value to it.
- otherwise what will happen that the new schema being registered won't be compatible with an earlier schema and will throw an error.

Problem:


Espresso
Voldemort
Oracle
User Tracking
Operational Logs
Operational Metrics
Hadoop
Log Search
Monitoring
Data Warehouse
Social Graph
Rec. Engine
Search
Security
...
Email
Production Services

Solution:


Espresso
Voldemort
Oracle
User Tracking
Operational Logs
Operational Metrics
Unified Message Log
Hadoop
Log Search
Monitoring
Data Warehouse
Social Graph
Rec Engine
Search
Security
...
Email
Production Services



Kafka Cluster:

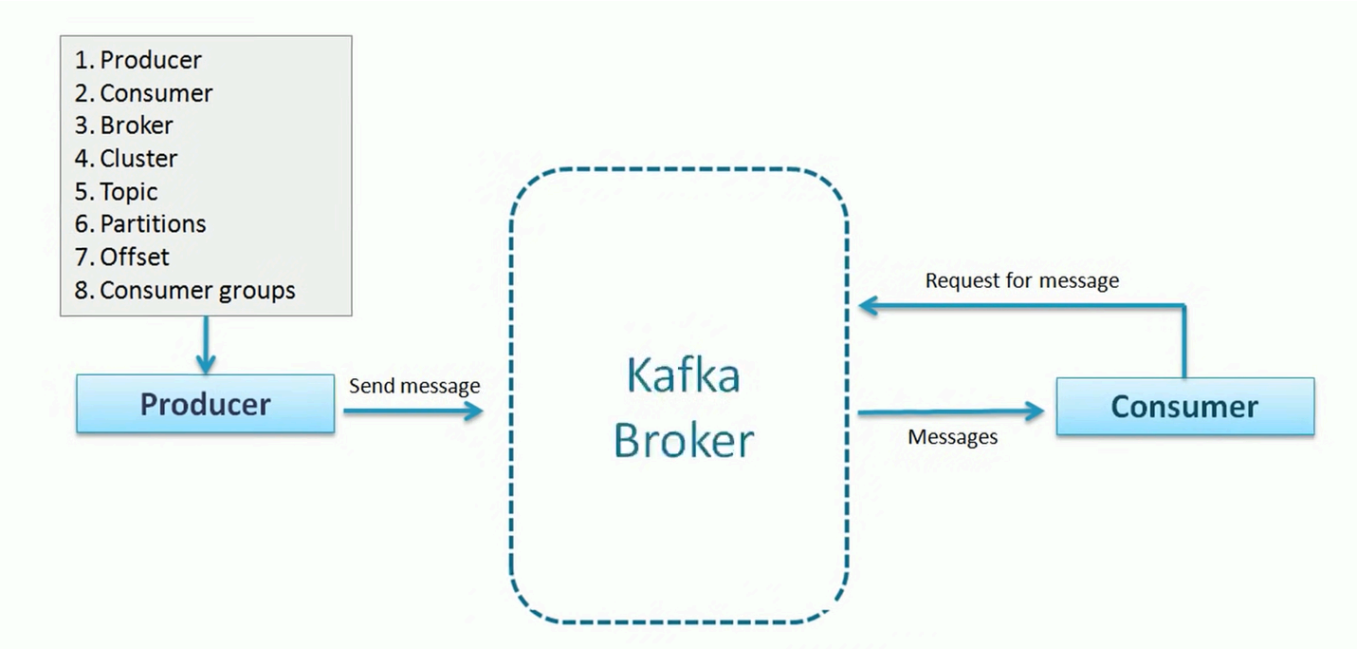
- Bunch of brokers running in a distributed manner, they take messages from the Producers and store in the Kafka logs.
- Consumer applications read messages from the Kafka cluster, process it and forward it to whatever they want

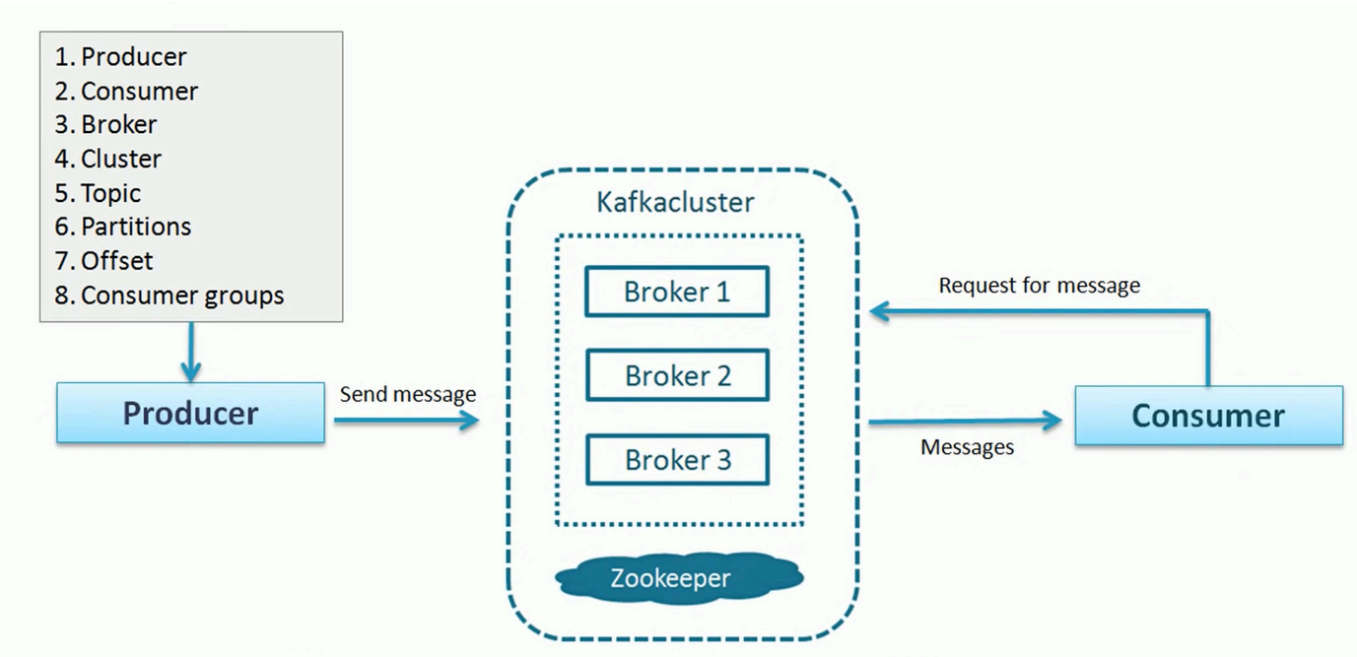
Stream:

- continuous flow of data / continuous massages
- stream processing applications read stream of messages from kafka, process them and either store them back to the kafka or send them directly to other system.

Connectors:

- ready to use connectors to import data from DBs into Kafka or export data from Kafka to DBs.





usually, different type of data is stored in different topics.

Partition:

- let's say data is so huge that storing it in a single topic of a machine is not possible.
- so, we do partitioning across the multiple machines of cluster.

Offset:

- A sequence Id given to messages as they arrive in a partition
- Kafka stores message in a order they arrive.

To Locate a message, we need 3 things: Topic Name, Partition number, offset.

Consumer group:

- let's say 100s of producers , so we've partitioned a topic .
- so, at source side we have multiple brokers to share the produced data's load.
- what about destination side, we have single consumer.
- so, we create a consumer group, and start executing multiple consumers and they divide the work.
- Kafka doesn't let 2 consumers to read from the same partitions simultaneously.

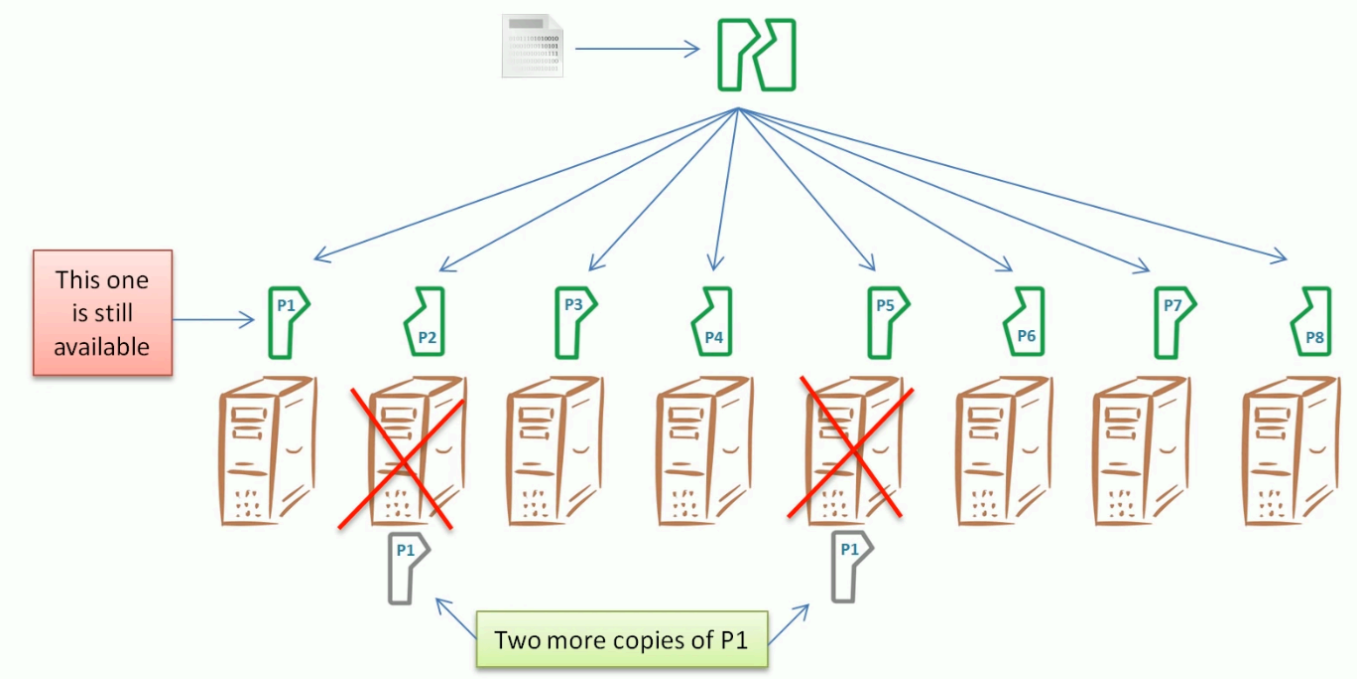
=> so, partitioning and consumer group are tools for scalability.

Zookeeper:

- provides coordination services for a distributed systems.

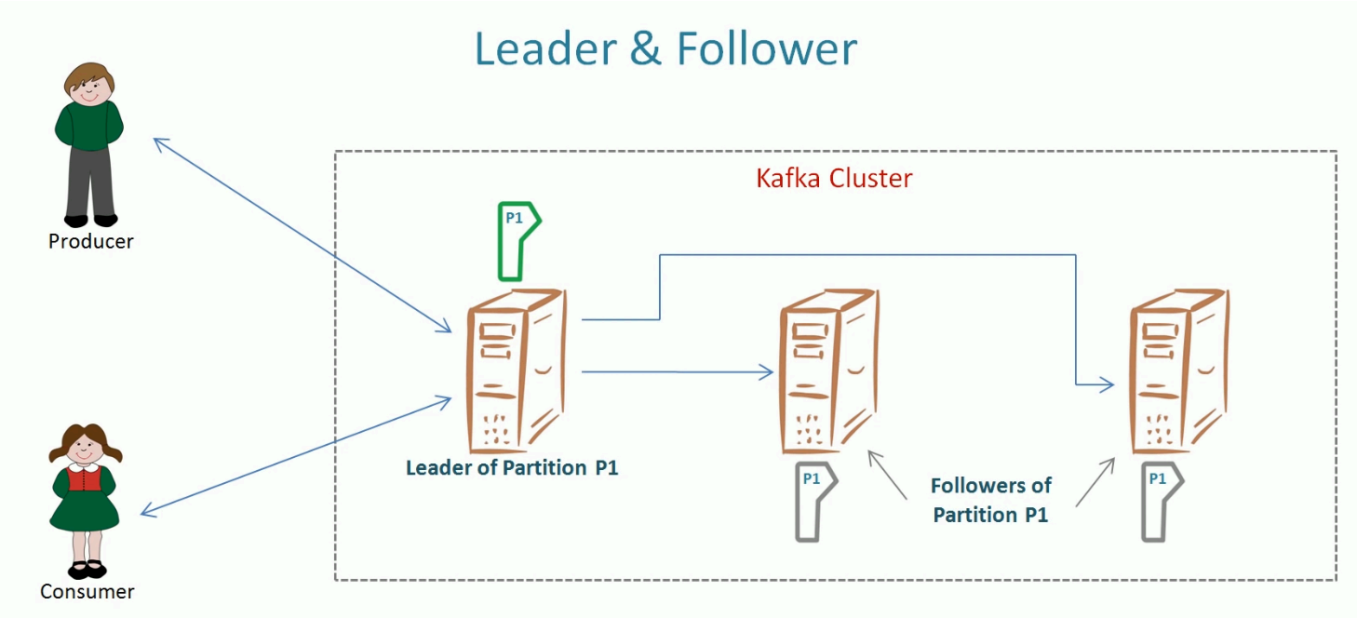
Fault Tolerance in Kafka

- Enabling a system to continue operating properly in the event of the failure of some of its components.



Replication factor:

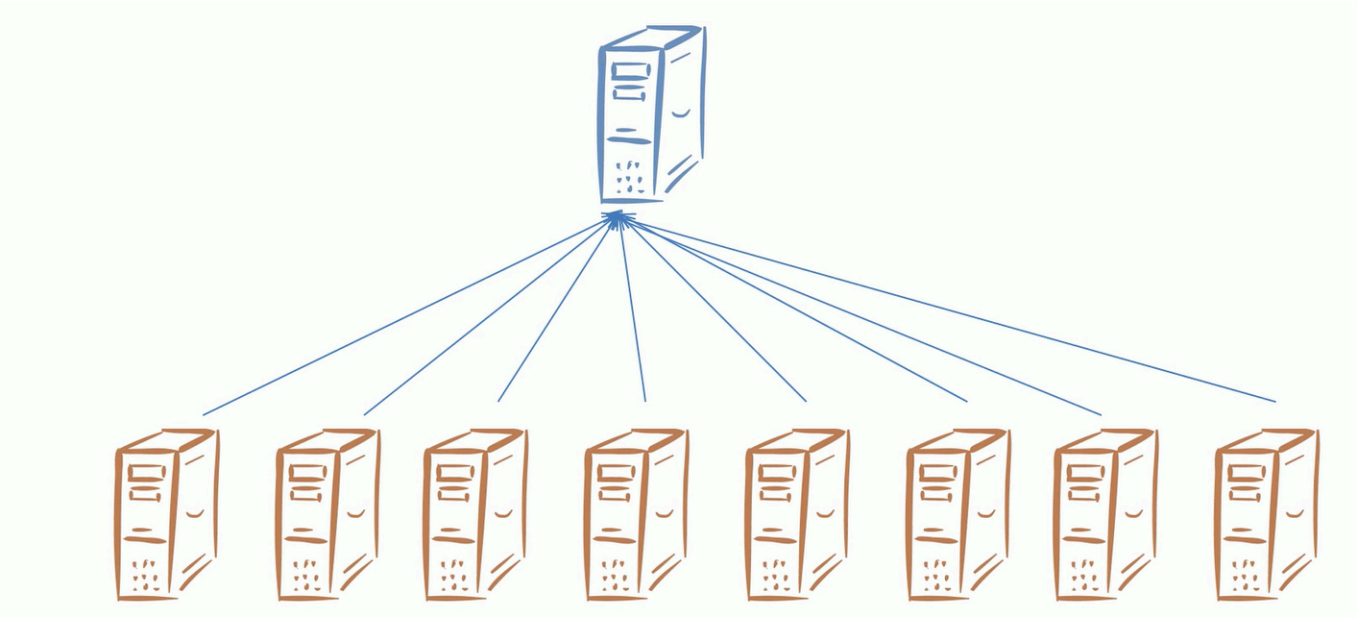
- Number of total copies
- Replication factor 3 means maintaining 2 copies + original one
- so, Kafka implements fault tolerance by applying replication to the partitions.
- RF is defined for the topic.



- 3 node kafka cluster means we have 3 machines in the cluster.
- to start the multiple brokers on the same machine, we need to copy the server.properties time and change its name.
- by using the same server.properties file, we can't start the 2 brokers on same machine.
- and then open the copied file, we have to change the following details:
  - change the broker.id , it is unique for each broker.
  - change the PORT number, to which broker binds itself.
  - logs.dirs - it is the main data directory of the broker and we don't want all brokers to write in the same directory so we need to change this value as well.

```
Topic:TestTopicXYZ      PartitionCount:2        ReplicationFactor:3     Configs:
        Topic: TestTopicXYZ     Partition: 0    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: TestTopicXYZ     Partition: 1    Leader: 2       Replicas: 2,0,1 Isr: 2,0,1
```

- Isr : In-sync replicas
- we have a leader for each partition of the topic, it means all the producer/consumer request related to that particular partition will be handled by the assigned leader.
- zookeeper.connect is hostname with port number
- Kafka uses zookeeper for various coordination purposes.
- so, zookeeper is the connecting link for all brokers to form a cluster.



Important parameters:

- delete.topic.enable
- auto.create.topic.enable
- default.replication.factor
- num.partitions
- log.retention.ms -> default is 7 days
- log.retention.bytes -> this applies to partition size (not topic size)

- we can use Kafka to solve various integration problems as we saw earlier.
- to create series of validations , transformations and build complex data pipelines.
- can use to record some information for later consumptions like recording click history.
- can use it to log transactions and create applications to respond in real time.
- can use it to collect data from mobile in an IOT applications.

---

steps to create producer:

- first create a producer object for which we need properties map with at least 3 properties: bootstrap-servers (list of kafka brokers), key-serializer and value-serializer
- then create a producer record object using topic name, key and value.
- then send this record to producer.
- Kafka only accepts an array of bytes, so we need serializer.

Producer Workflow: