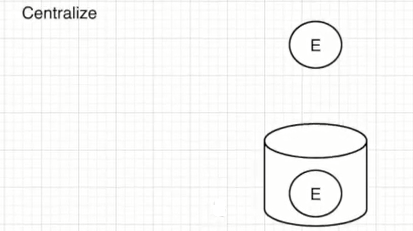
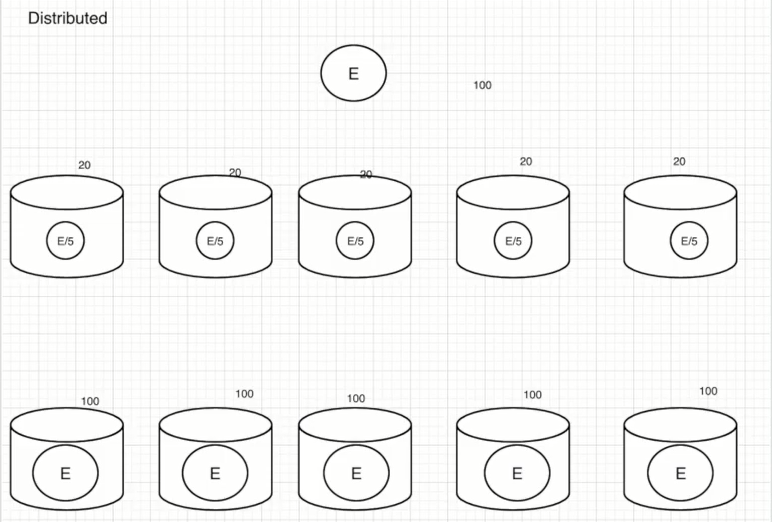
**What is Kafka?**

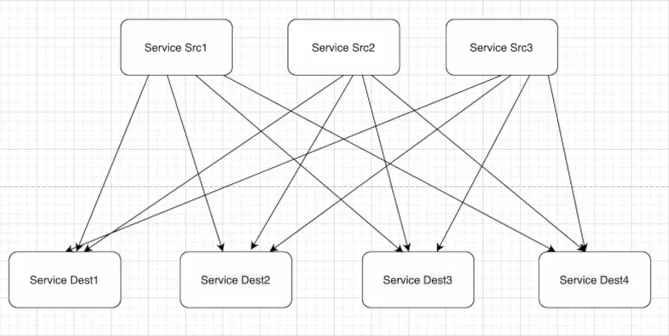
* Kafka is a fast, scalable, fault-tolerant distributed message streaming platform that uses publish and subscribe mechanism to stream the records.
* It’s a publish-subscribe messaging system which lets you exchanging of data between applications, servers, and processors as well.
* Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation.
* Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.
* Currently used by many big enterprises like LinkedIn, Airbnb, Netflix, Uber, Walmart etc.

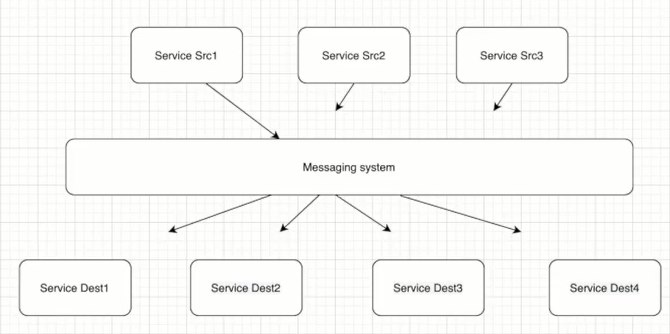




**What is a messaging system?**

* A messaging system is a simple exchange of messages between two or more persons, devices, etc.
* A publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message.
* In Apache Kafka, a sender is known as a producer who publishes messages, and a receiver is known as a consumer who consumes that message by subscribing it.



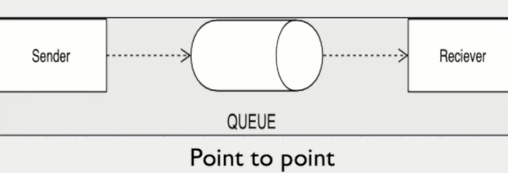


There are two types of messaging patterns available:

* Point to point messaging system
* Publish-subscribe messaging system

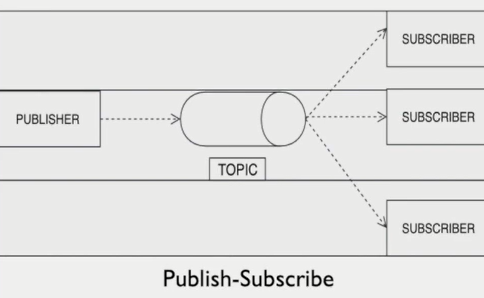
**a) Point to Point Messaging System**

* Messages are persisted in a queue.
* A particular message can be consumed by a maximum of one receiver only.
* After the receiver reads the message in the queue, the message disappears from that queue.
* There is no time dependency laid for the receiver to receive the messages.
* When the Receiver receives the message, it will send an acknowledgment back to the Sender.

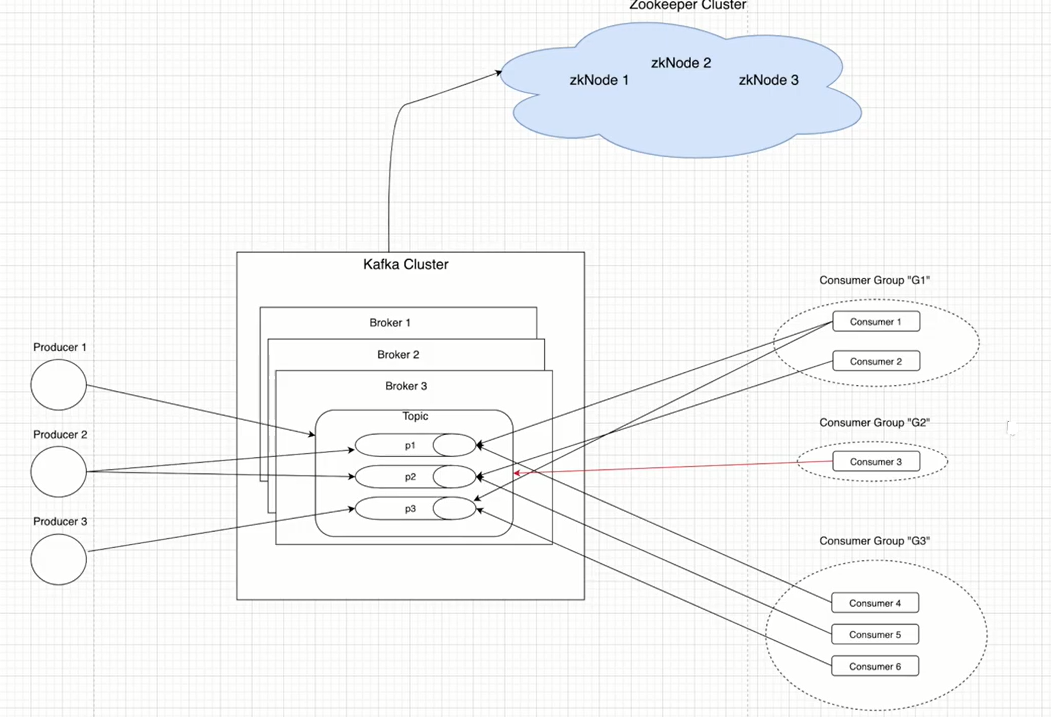


**b) Publish-Subscribe Messaging System**

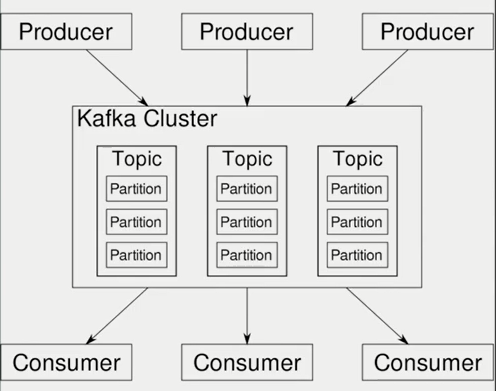
* Messages are persisted in a Topic.
* A particular message can be consumed by any number of consumers.
* There is time dependency laid for the consumer to consume the message.
* When the subscriber receives the message it doesn’t send an acknowledgement back to the publisher.



**Kafka Architecture:** Below we are discussing four core APIs of Apache Kafka.

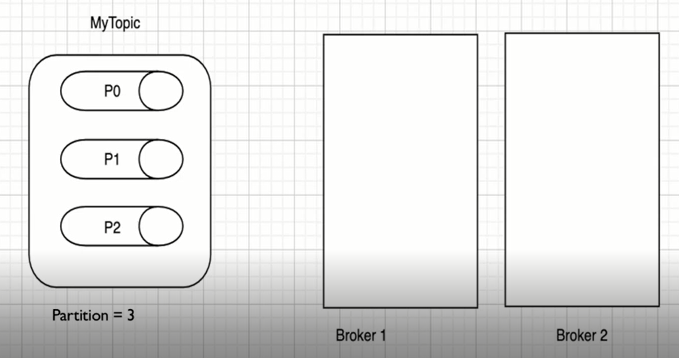


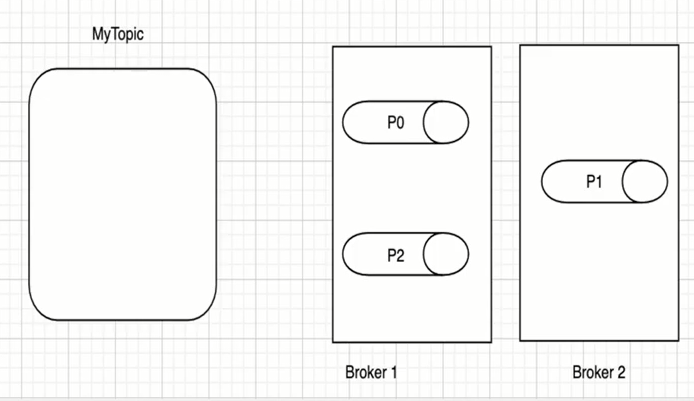
**1) Topics:** A stream of messages belonging to a particular category is called a topic. It is a logical feed name to which records are published. Similar to a table in DB (Records are considered messages here). Unique identifier of a topic is its name. We can create as many topics as we want.

****

**Partitions:**

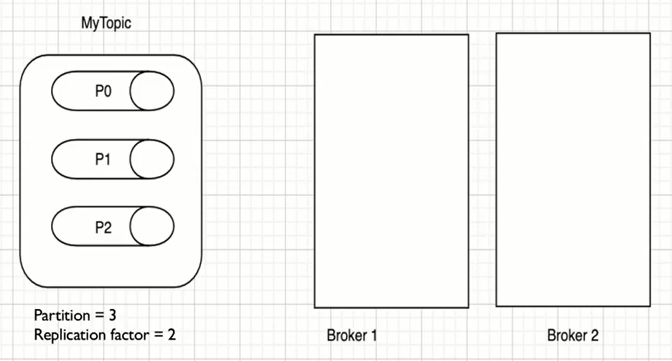
Topics are split into partitions. All the messages within a partition are ordered and immutable. Each message within a partition has a unique id associated knows as Offset.

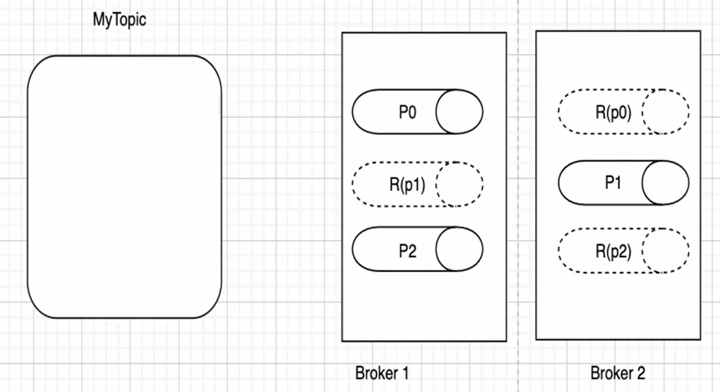
****

****

**Replica/Replication:**

Replicas are backup of a partition. Replicas are never read or write data. They are used to prevent data loss (Fault Tolerant).





**2) Producers:**

* Producers are applications which write/publish data to the topics within a cluster using the Producer APIs.
* Producers can write date either on the topic level (All the partition of that topic in a round robin manner) or specific partitions of that topic.

**3) Consumers:**

* Consumers are applications which read/consume data from the topics within a cluster using the Consumer APIs.
* Consumers can read date either on the topic level (All the partition of that) or specific partitions of that topic.
* Consumers are always associated with exactly one Consumer Group which is a group of Consumers that performs a task.

**4) Brokers:**

* Brokers are simple software processes who maintain and manage the published messages also known as Kafka Servers.
* Brokers also manage the consumer-offsets and are responsible for the delivery of messages to the right consumers.
* A set of brokers who are communicating with each other to perform the management and maintenance task are collectively known as Kafka Cluster.
* We can add more brokers in a already running Kafka cluster without and downtime which ensures horizontal scalability.

**5) Zookeeper:**

* Zookeeper is used to monitor Kafka Cluster and co-ordinate with each broker.
* Keeps all the metadata information related to Kafka Cluster in the form of Key-Value pair.

***Metadata Includes:***

Configuration Information

Health status of each broker.

* It is used for the controller election within Kafka Cluster.
* A set of Zookeepers nodes working together to manage other distributed systems is known as Zookeeper Cluster or Zookeeper Ensemble.

**6) Offset in Kafka:**

The position of the consumer in the log and which is retained on a per-consumer basis is what we call Offset.

**Kafka Features:**

**1) Scalable:** Horizontal scaling is done by adding new brokers to the existing clusters.

**2) Fault Tolerance:** Kafka Clusters can handle failures because of its distributed nature.

**3) Performance:** Kafka has high throughput for both publishing and subscribing messages.

**4) No Data Loss:** It ensures no data loss if we configure it properly.

**5) Zero Down Time:** It ensures zero downtime when required number of brokers are present in the cluster.

**Zookeeper:**

**a) Start the Zookeeper:**

cd $ZOOKEEPER\_HOME

bin/zkServer.sh start

**b) Validate your Zookeeper is running or not:**

echo stat | nc localhost 2181

**c) Stop the Zookeeper:**

bin/zkServer.sh stop

**Kafka:**

**a) Start the Kafka:**

cd $KAFKA\_HOME

bin/kafka-server-start.sh config/server.properties

**b) Validate your Kafka is running or not:**

echo dump | nc localhost 2181 | grep brokers

**c) Stop the Kafka:**

bin/kafka-server-stop.sh

**1) Create Topic:**

bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic myTopic

--partitions 1 --replication-factor 1

**2) List Topic:**

bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

**3) Describe Topic:**

bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic myTopic

**4) To create a Producer:**

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic myTopic

**5) To create a Consumer:**

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic myTopic

--from-beginning

**6) View Consumer Groups:**

bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

**7) Describe Consumer Groups:**

bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group console-consumer-55106

**8) Once the data is consumed we can see one \_\_consumer\_offsets created where all the offsets information is stored.**

bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

**9) Multiple Producer One Consumer:**

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic myTopic

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic myTopic

**10) Multiple Consumer One Producer:**

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic myTopic

--from-beginning

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic myTopic

**11) Let's verify how many Consumer Groups are now available:**

bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

**12) Create Consumer Group with User Defined Group:**

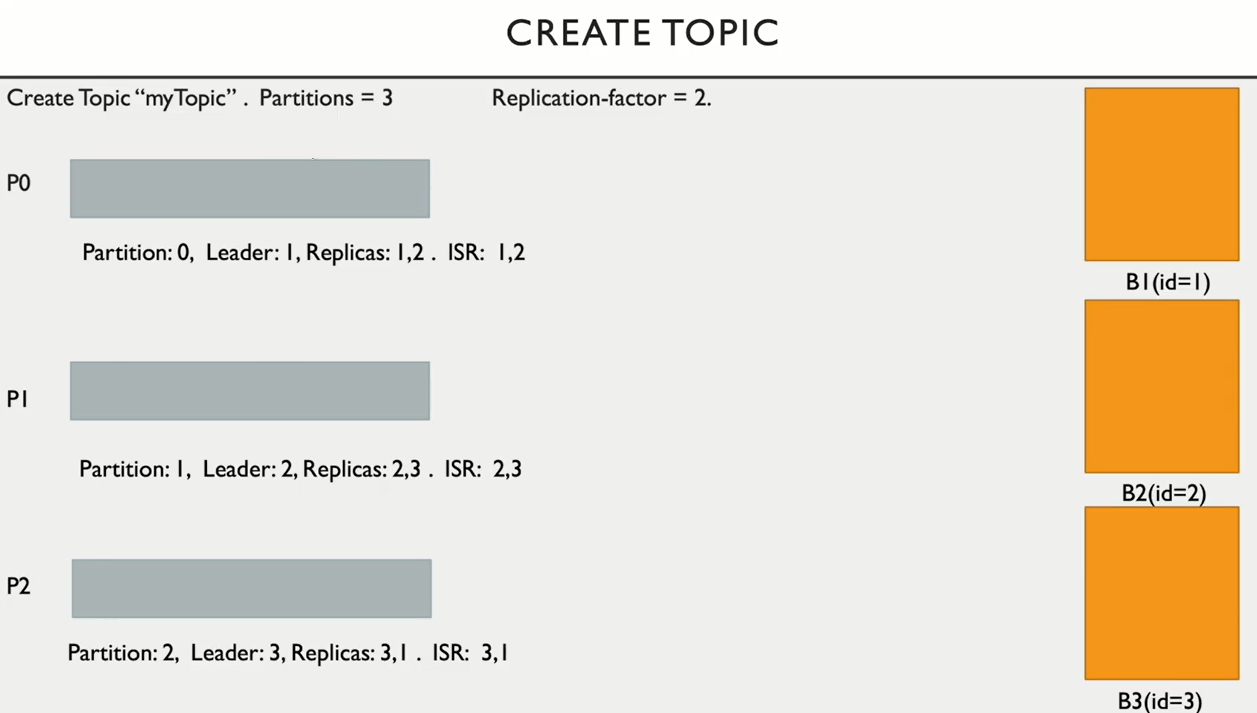
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic myTopic

--group myConsumerGroup

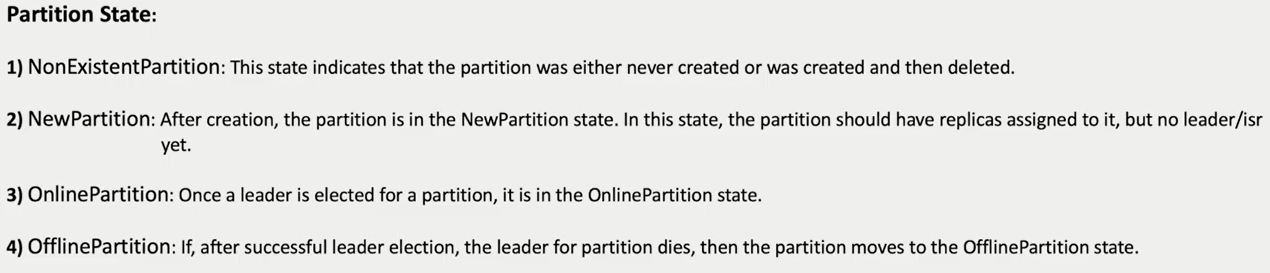
**Push data from Producer & see message in your User Defined Consumer Group**

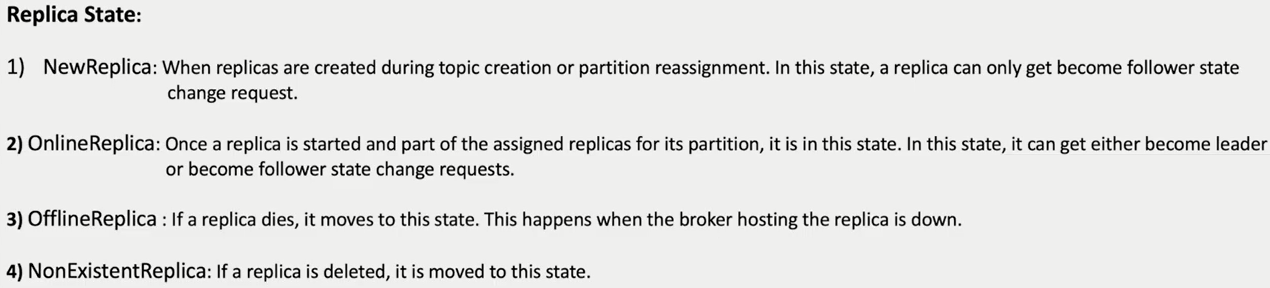
**List Consumer Groups:**

bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 –list

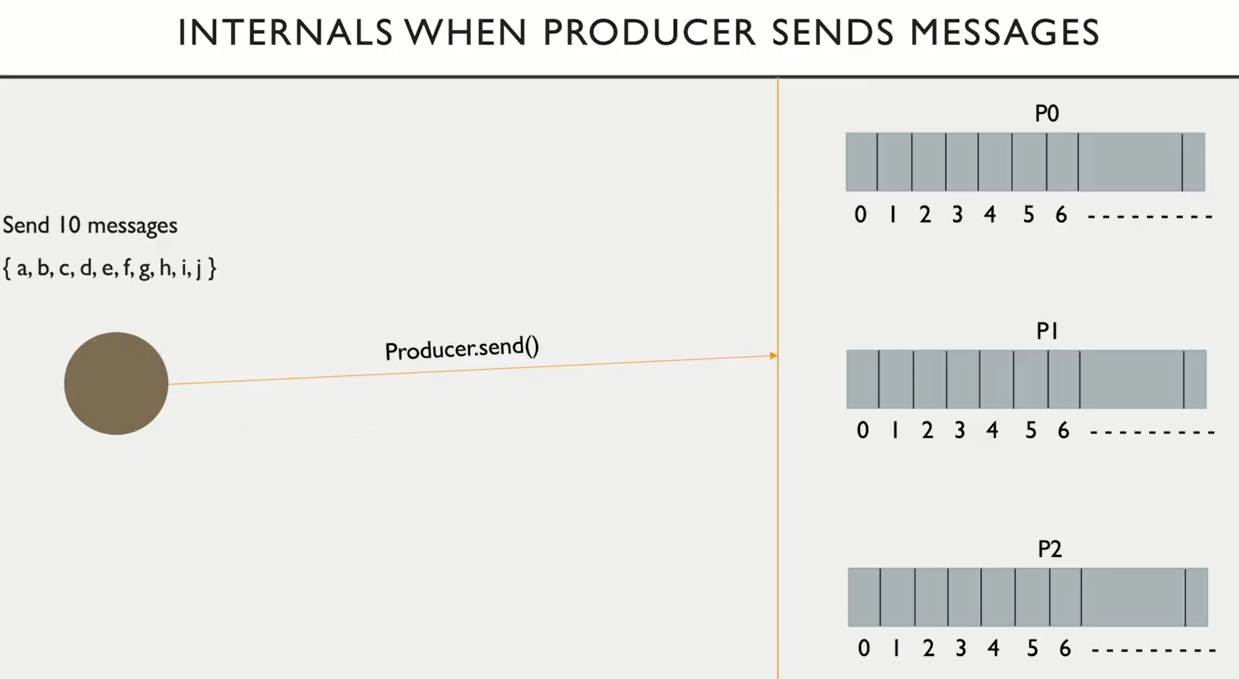
****

**What happens internally when we create topic:**

****



**Internals of Producer:**



**Offsets:**

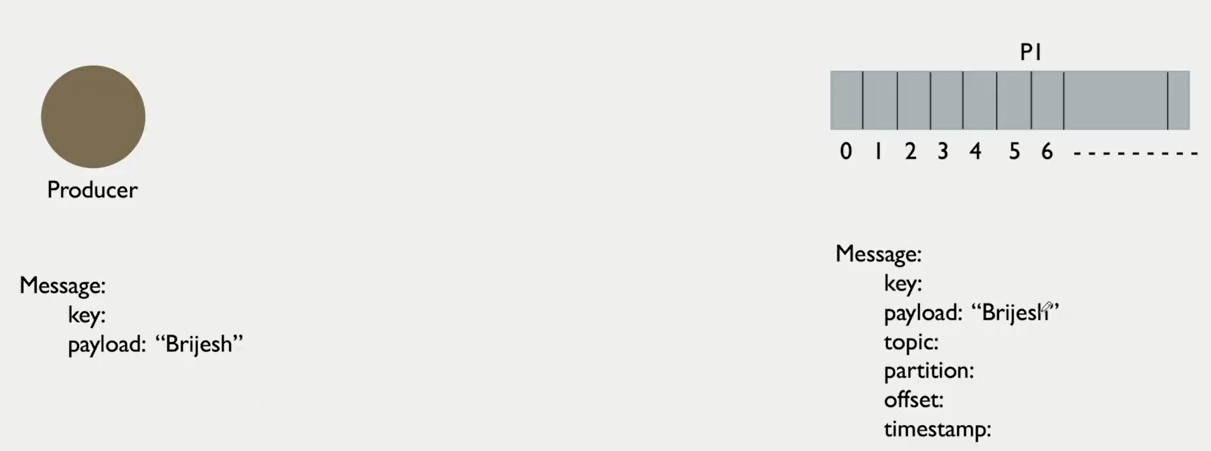
The records in the partitions are assigned a sequential id number called offset that uniquely identifies each record within the partition.

**1) Log-end offset:** Offset of the last message written to a log/partition.

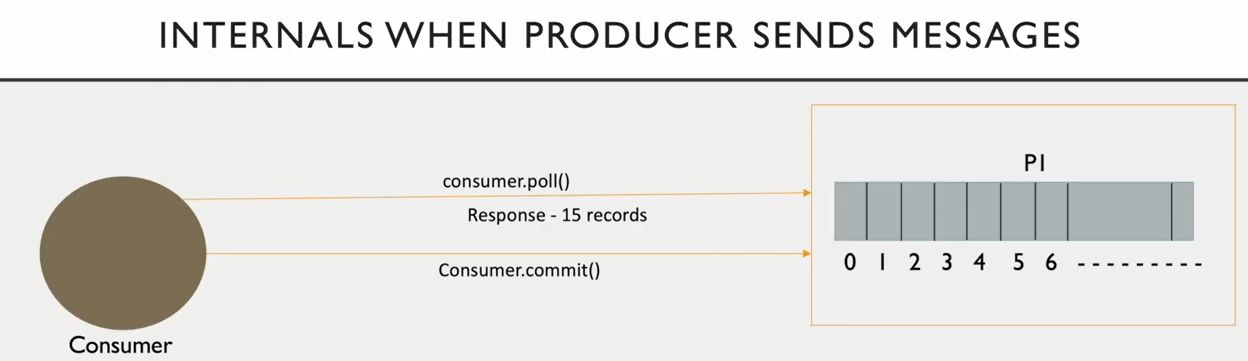
**2) Current offset:** Pointer to the last record that Kafka has already sent to a consumer in the most recent poll.

**3) Committed offset:** Marking an offset as consumed is called committing an offset.

**Metadata** i.e. information about data when messages are sent to Topic.



**Note:** When Key is null it always send the messages in Round-Robin fashion. If key has some value then the messages are directly sent to the required partition.



**max.poll.response (15 records)**

**Offsets Cont’d…**

**2) Current offset:** Pointer to the last record that Kafka has already sent to a consumer in the most recent poll.

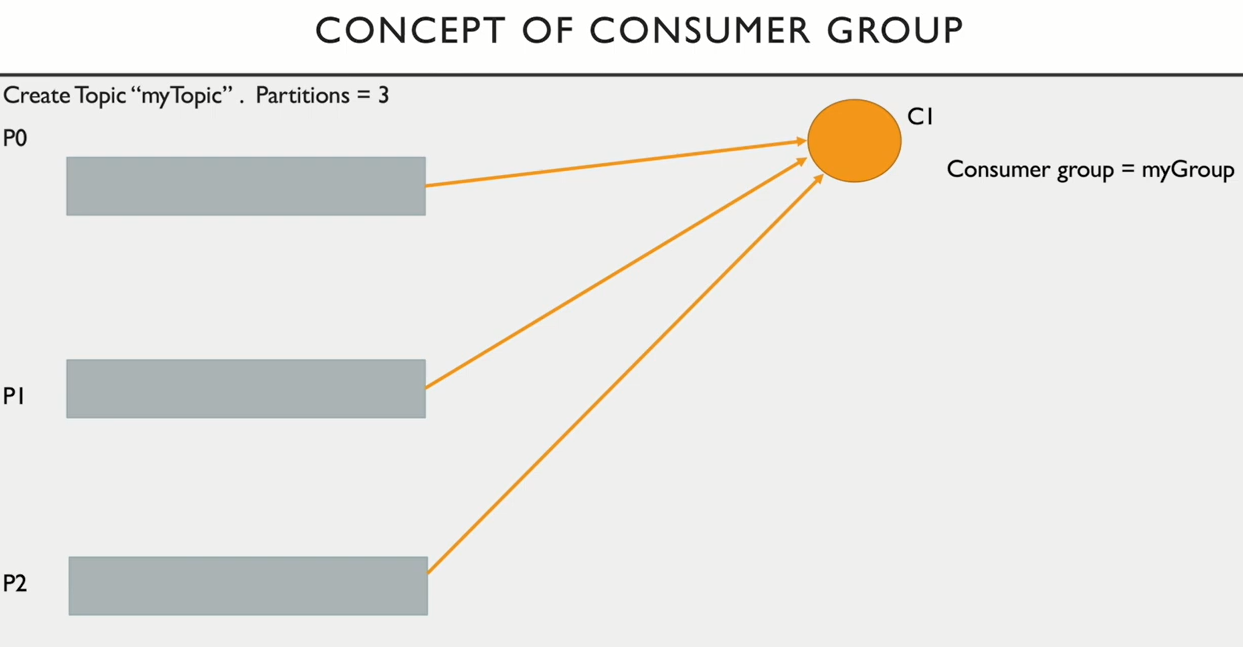
**3) Committed offset:** Marking an offset as consumed is called committing an offset.

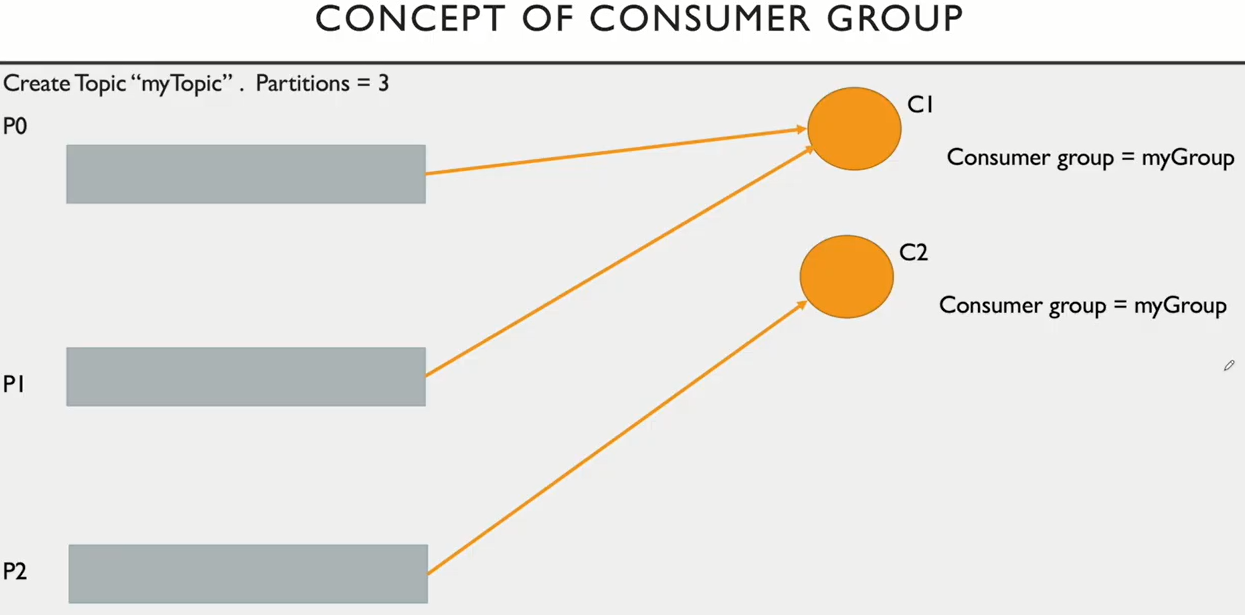
**Kafka Consumer Group:**

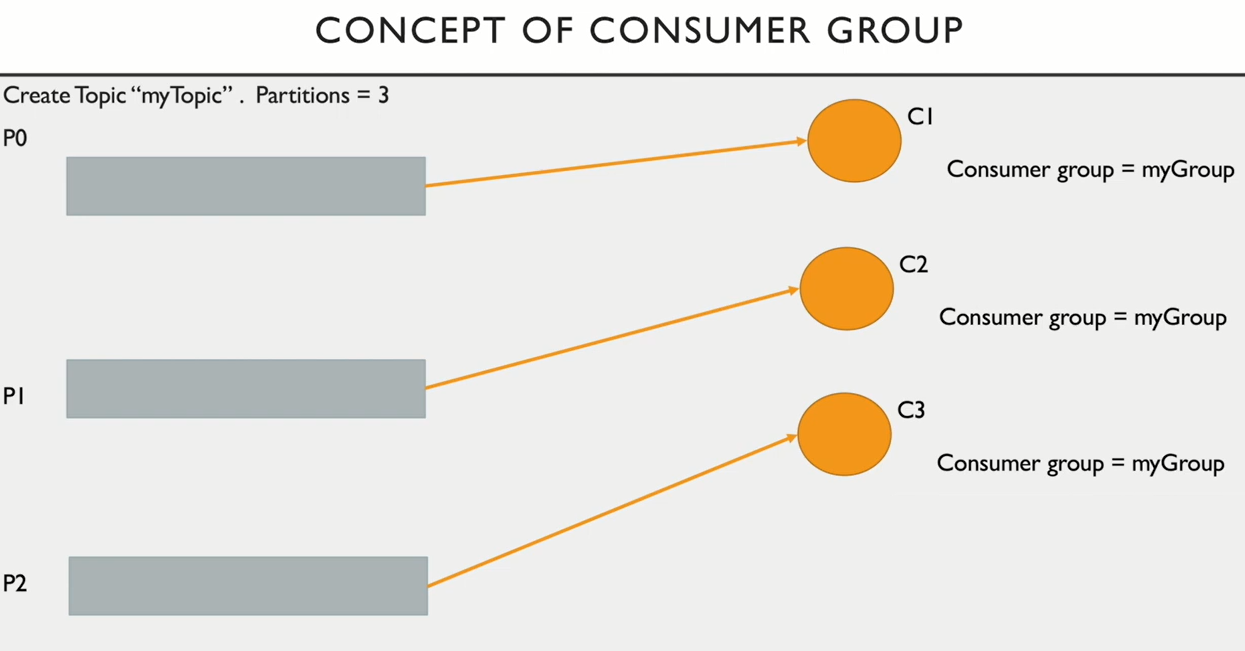
Consumer group is a logical entity in Kafka ecosystem which mainly provides parallel processing/scalable message consumption to consumer clients.

**1)** Each consumer must be associated with some consumer group.

**2)** Make sure there is no duplication within consumers who are part of the same consumer group.







**Delete Topic:**

bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic testDelete