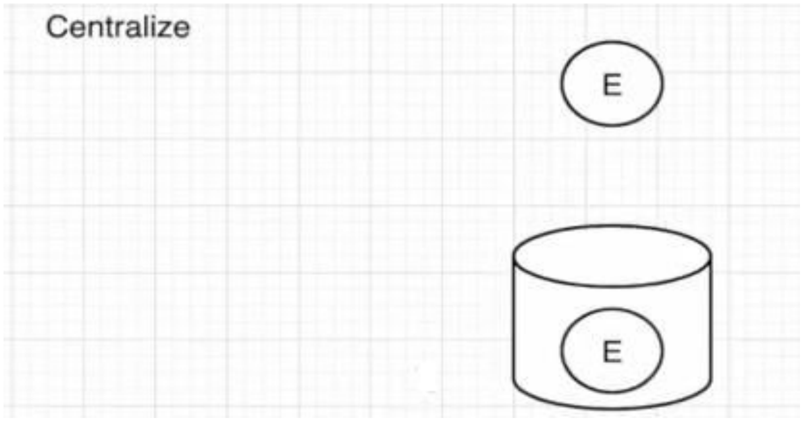
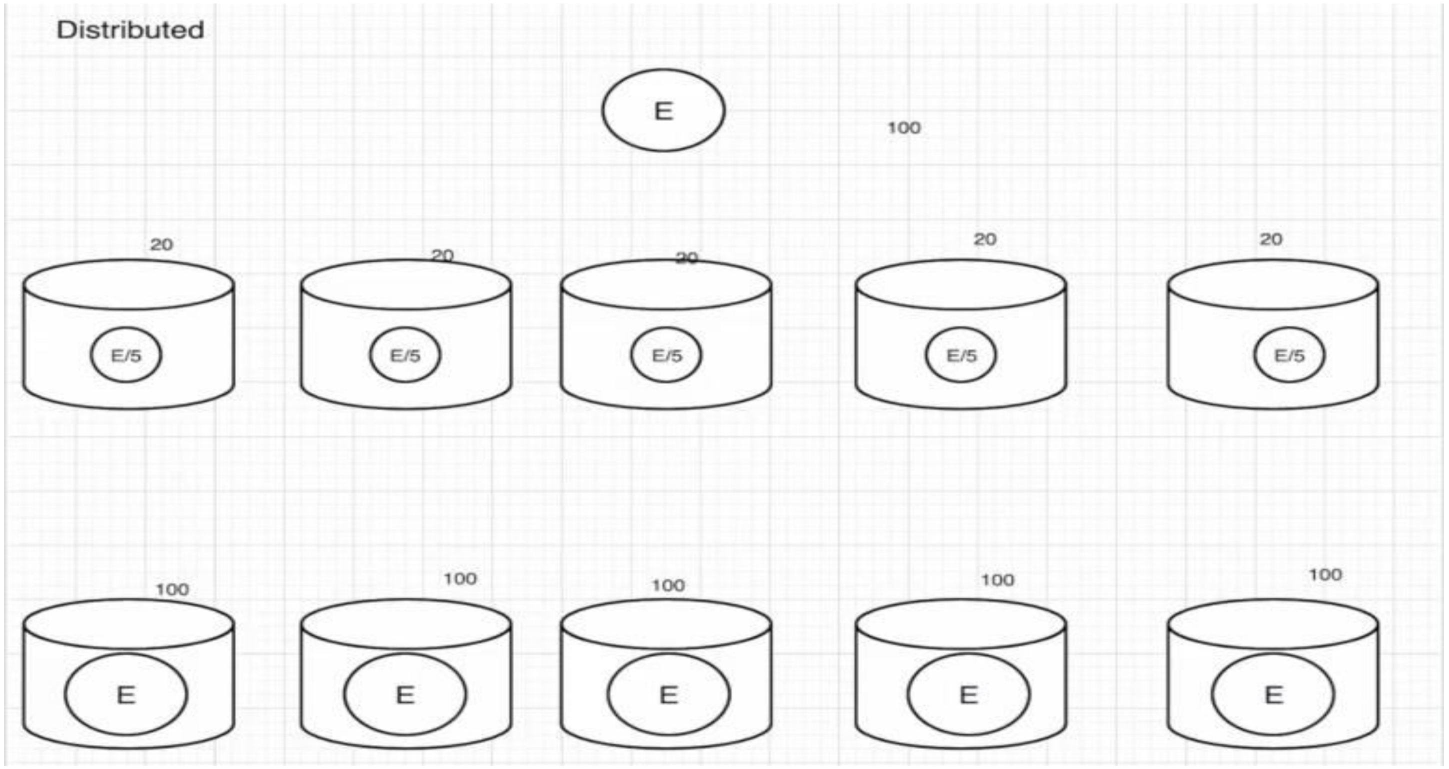
What is Kafka?

- 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.

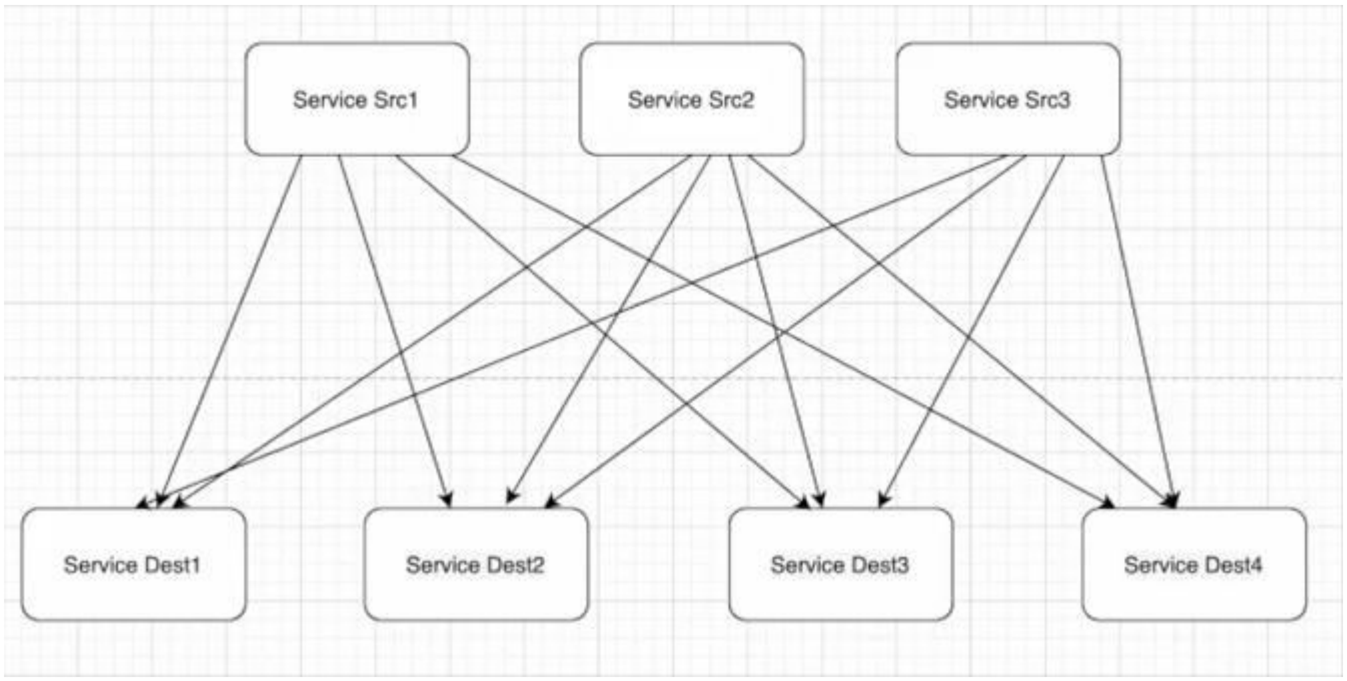


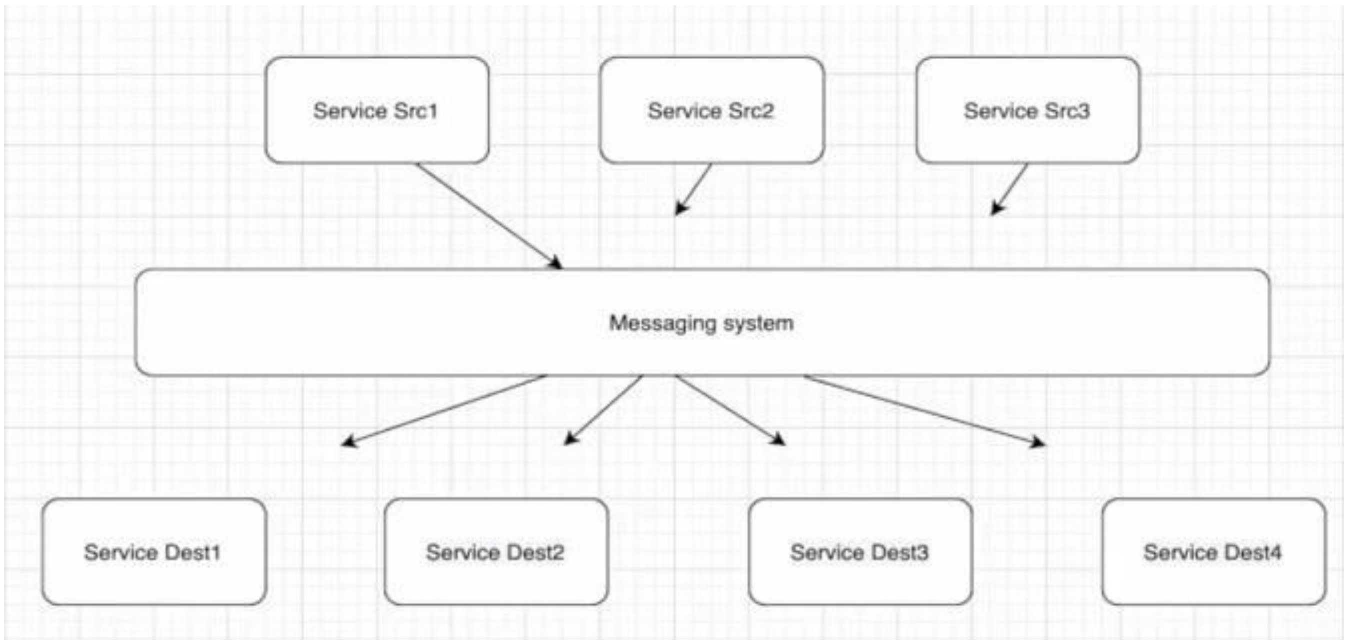


What is a messaging system?

- 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.



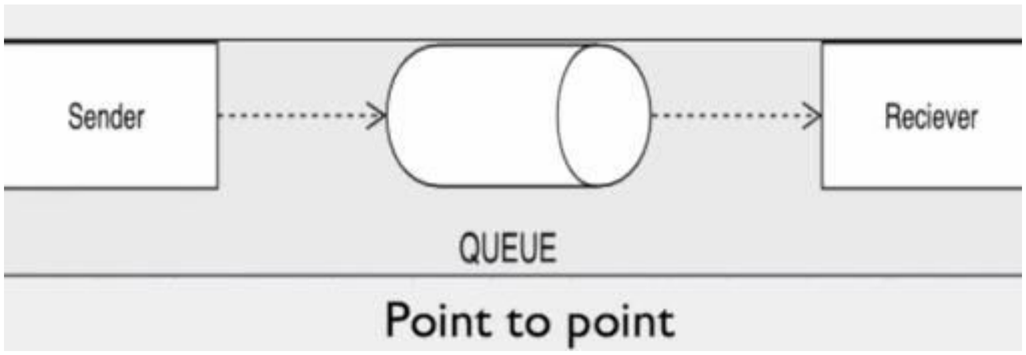


There are two types of messaging patterns available:

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

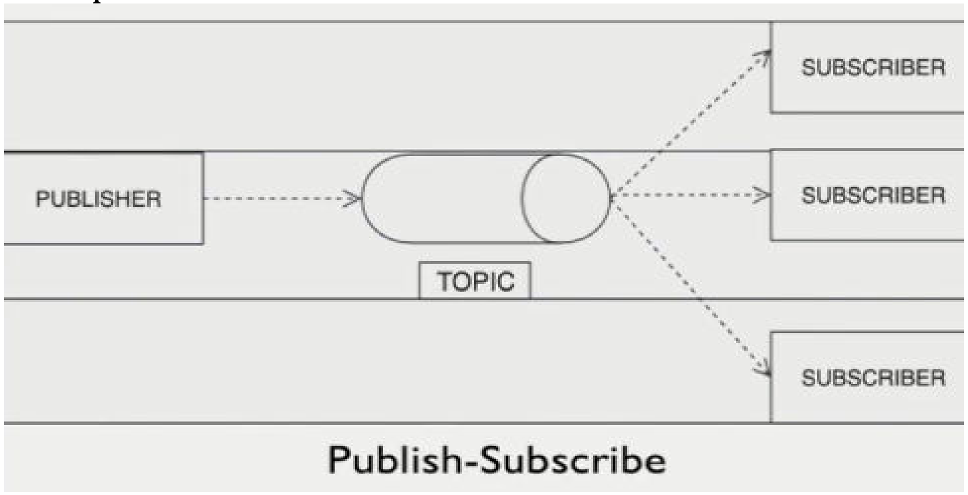
a) Point to Point Messaging System

- 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.

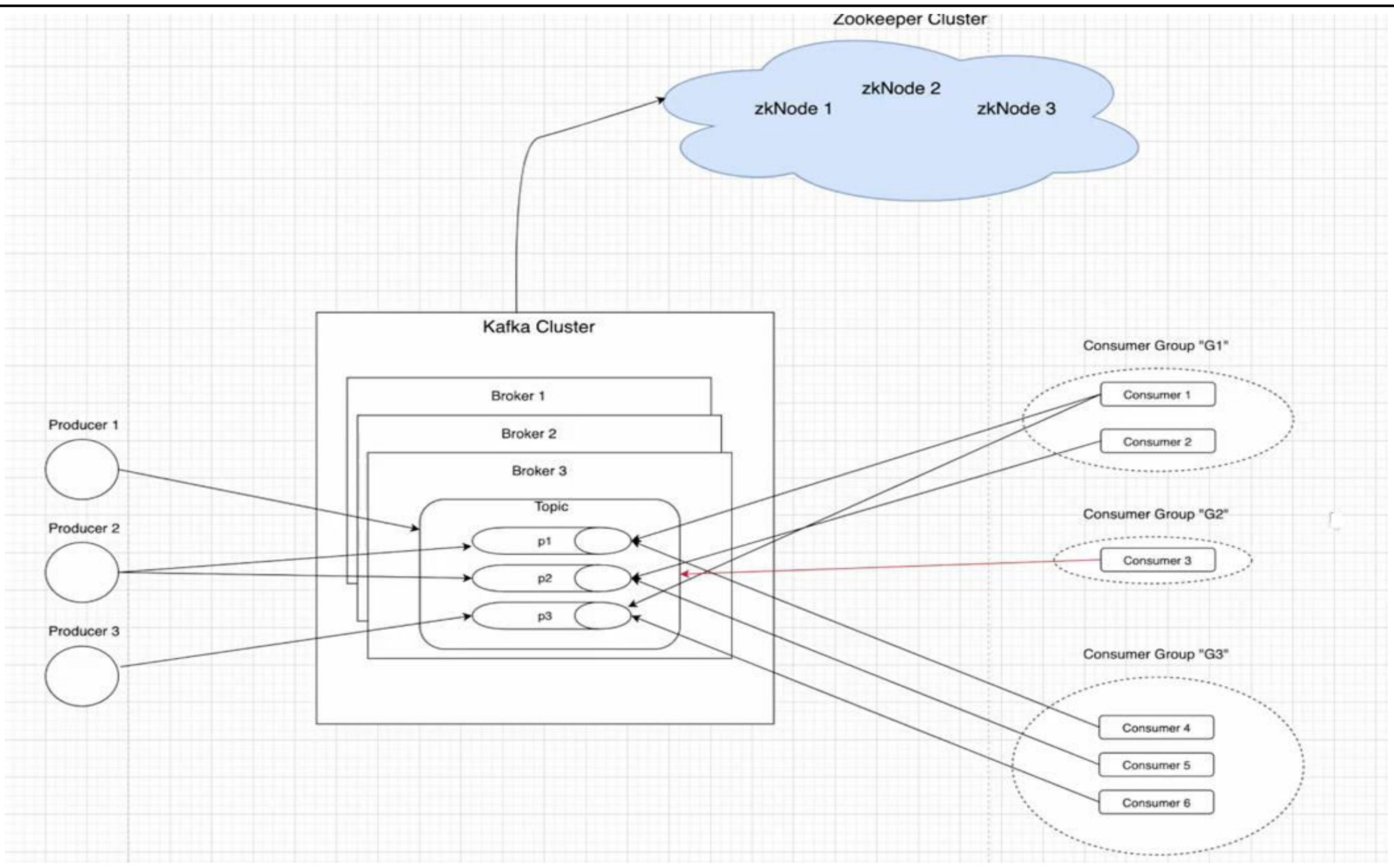


b) Publish-Subscribe Messaging System

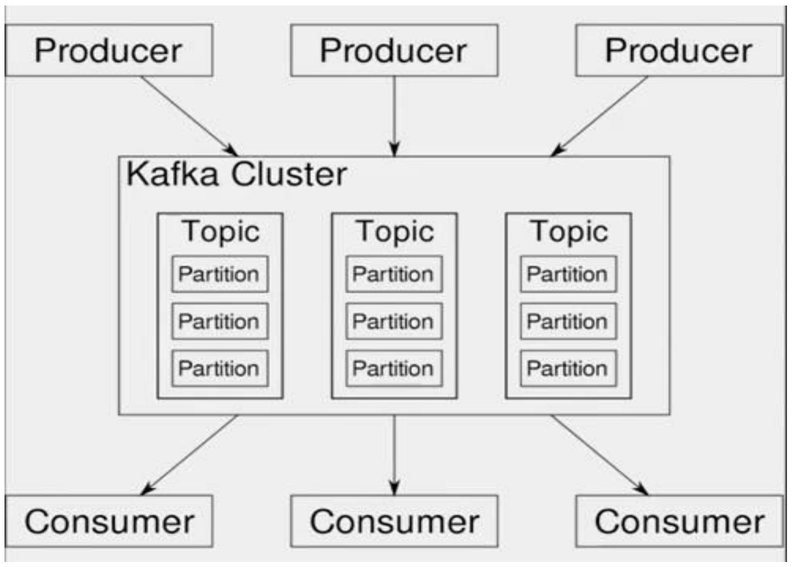
- Messages are persisted in a Topic.
- A particular message can be consumed by any number of consumers.
- There is time dependency laid for the consumer to consume the message.
- When the subscriber receives the message it doesn't send an acknowledgement back to the publisher.



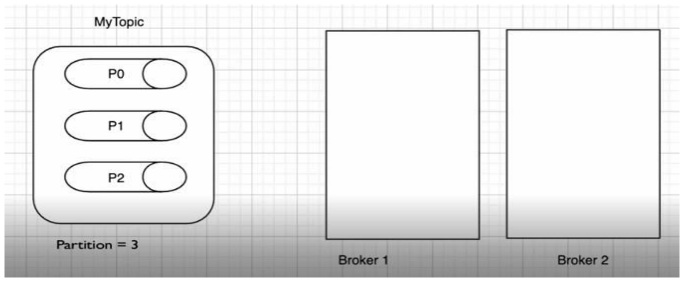
Kafka Architecture: Below we are discussing four core APIs of Apache Kafka.

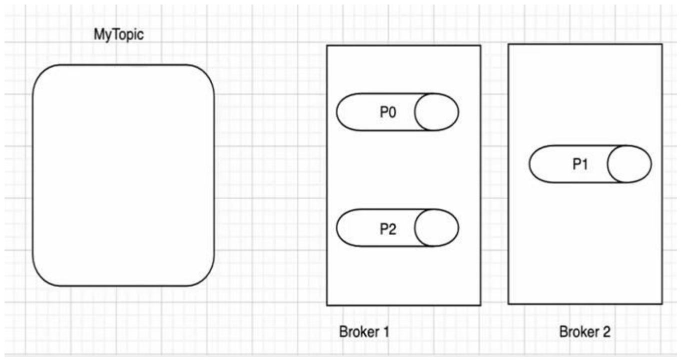


1) Topics: A stream of messages belonging to a particular category is called a topic. It is a logical feed name to which records are published. Similar to a table in DB (Records are considered messages here). Unique identifier of a topic is its name. We can create as many topics as we want.

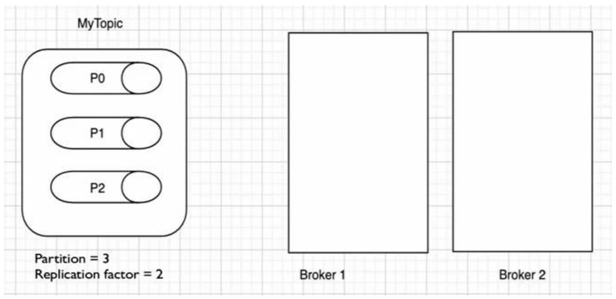


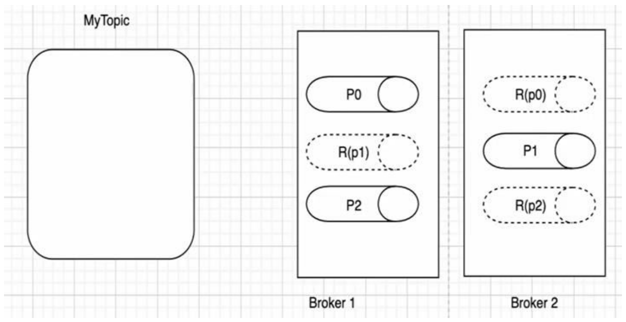
Partitions:
Topics are split into partitions. All the messages within a partition are ordered and immutable. Each message within a partition has a unique id associated knows as Offset.





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.

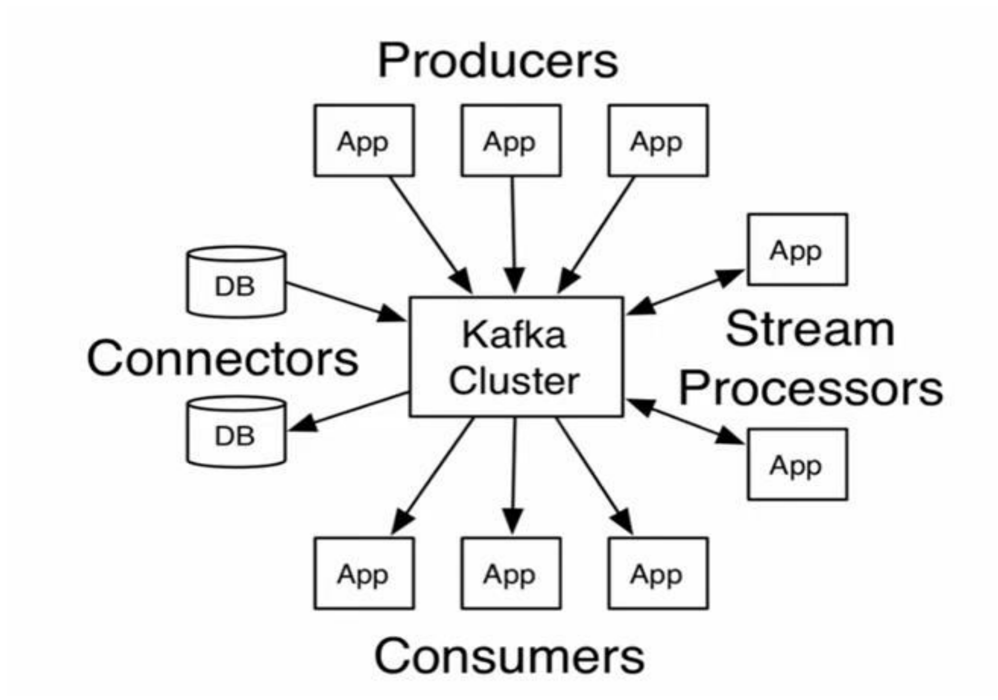
5) Zookeeper:

- 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.

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.

Kafka Core APIs:



6) Kafka Connector API:
The Connector API permits creating and running reusable producers or consumers that enables connection between Kafka topics and existing applications or data systems.

7) Kafka Streams API:
The Streams API permits an application to behave as a stream processor, consuming an input stream from one or more topics and generating an output stream to one or more output topics, efficiently modifying the input streams to output streams.

8) 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.
Zookeeper:
a) Start the Zookeeper:
b) Validate your Zookeeper is running or not:
c) Stop the Zookeeper:

Kafka:
a) Start the Kafka:
b) Validate your Kafka is running or not:
c) Stop the Kafka:

1) Create Topic:

2) List Topic:

3) Describe Topic:

4) To create a Producer:

5) To create a Consumer:

6) View Consumer Groups:

7) Describe Consumer Groups:

8) Once the data is consumed we can see one __consumer_offsets created where all the offsets information is stored.

9) Multiple Producer One Consumer:

10) Multiple Consumer One Producer:

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

12) Create Consumer Group with User Defined Group:

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

List Consumer Groups:

14) Multi-Node Zookeeper:

15) Multi-Node Kafka:

Kafka Controller Node:
In a Kafka Cluster, one of the broker serves as the controller which is responsible for managing the state of partitions and replicas, also it performs administrative tasks like reassigning partitions.

Command to see the controller:

16) Create Topic:

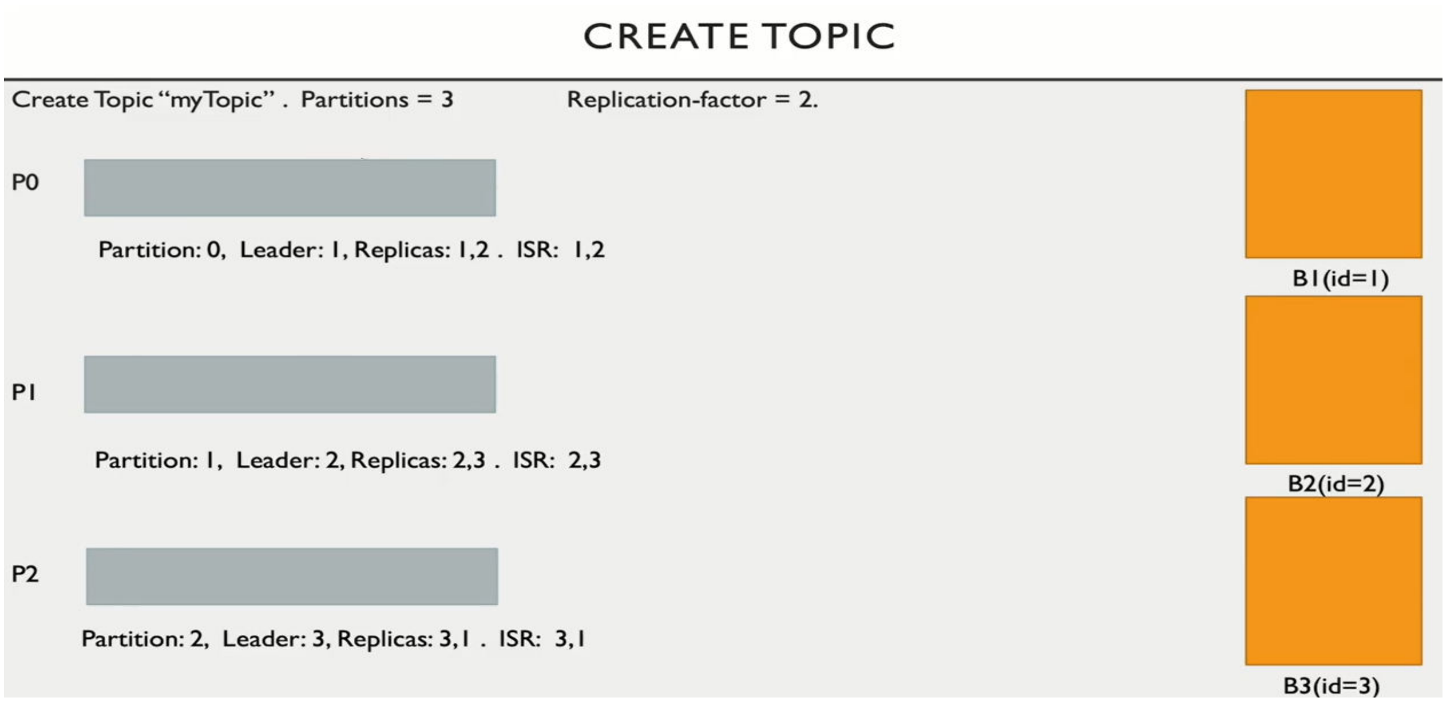
17) Describe Topic:

Note: If you stop one broker, then your insync replica will not be exact like before it will be a subset of replica, also it changes the leader.

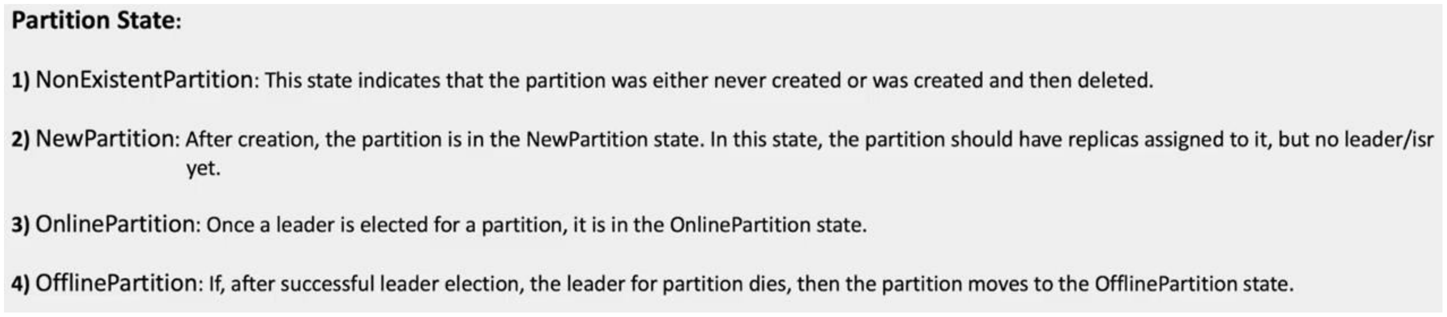
Again start your broker:

ISR will be updated again.

Internals of Topics, Partitions & Replicas:


CREATE TOPIC
Create Topic "myTopic" . Partitions = 3
Replication-factor = 2.
P0
Partition: 0, Leader: 1, Replicas: 1,2 . ISR: 1,2
P1
Partition: 1, Leader: 2, Replicas: 2,3 . ISR: 2,3
P2
Partition: 2, Leader: 3, Replicas: 3,1 . ISR: 3,1
B1(id=1)
B2(id=2)
B3(id=3)

What happens internally when we create topic:



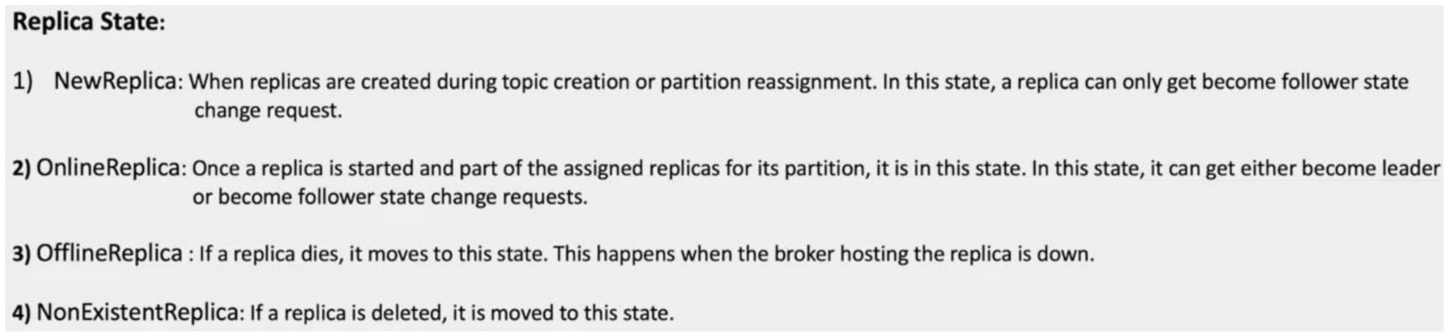
Partition State:

1) NonExistentPartition: This state indicates that the partition was either never created or was created and then deleted.

2) NewPartition: After creation, the partition is in the NewPartition state. In this state, the partition should have replicas assigned to it, but no leader/isr yet.

3) OnlinePartition: Once a leader is elected for a partition, it is in the OnlinePartition state.

4) OfflinePartition: If, after successful leader election, the leader for partition dies, then the partition moves to the OfflinePartition state.



Replica State:

1) NewReplica: When replicas are created during topic creation or partition reassignment. In this state, a replica can only get become follower state change request.

2) OnlineReplica: Once a replica is started and part of the assigned replicas for its partition, it is in this state. In this state, it can get either become leader or become follower state change requests.

3) OfflineReplica : If a replica dies, it moves to this state. This happens when the broker hosting the replica is down.

4) NonExistentReplica: If a replica is deleted, it is moved to this state.

Let's Create Topic with 1 Partition & 1 Replication-Factor:

Let's Increase the partitions.
Note: In entire cluster only 1 controller node will be there and we cannot decrease the partitions.

Alter Partitions:

Partition Reassignment:
1) Move partitions across borders.
2) Selectively move replicas of a partition to a specific set of brokers.
3) Increasing the replication factor.

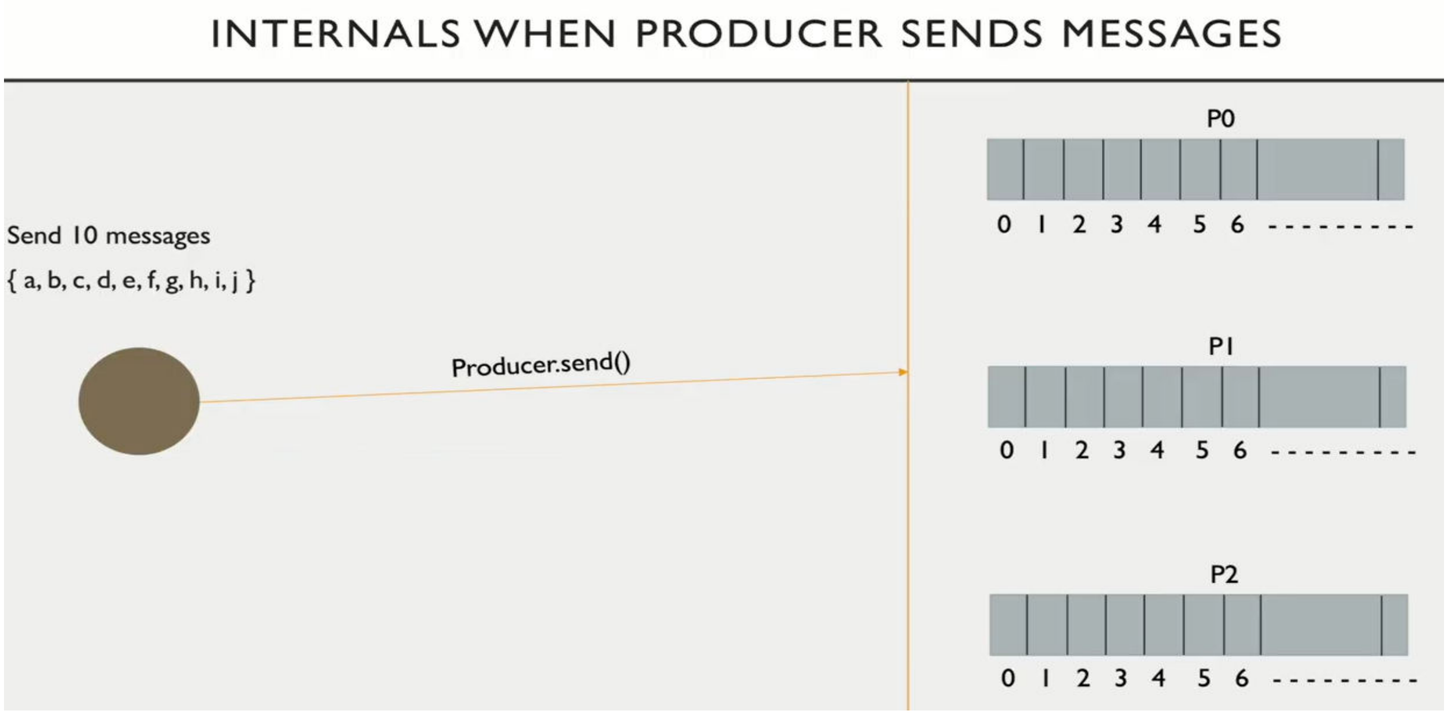
3 Steps Process:
1) Generate 2) Execute 3) Verify

a) Broker Movement:
Generate:
Execute:
Verify:
Verify reassignments:
b) Change/Increase the Replication-Factor:

c) Selectively send replicas to a set of brokers:

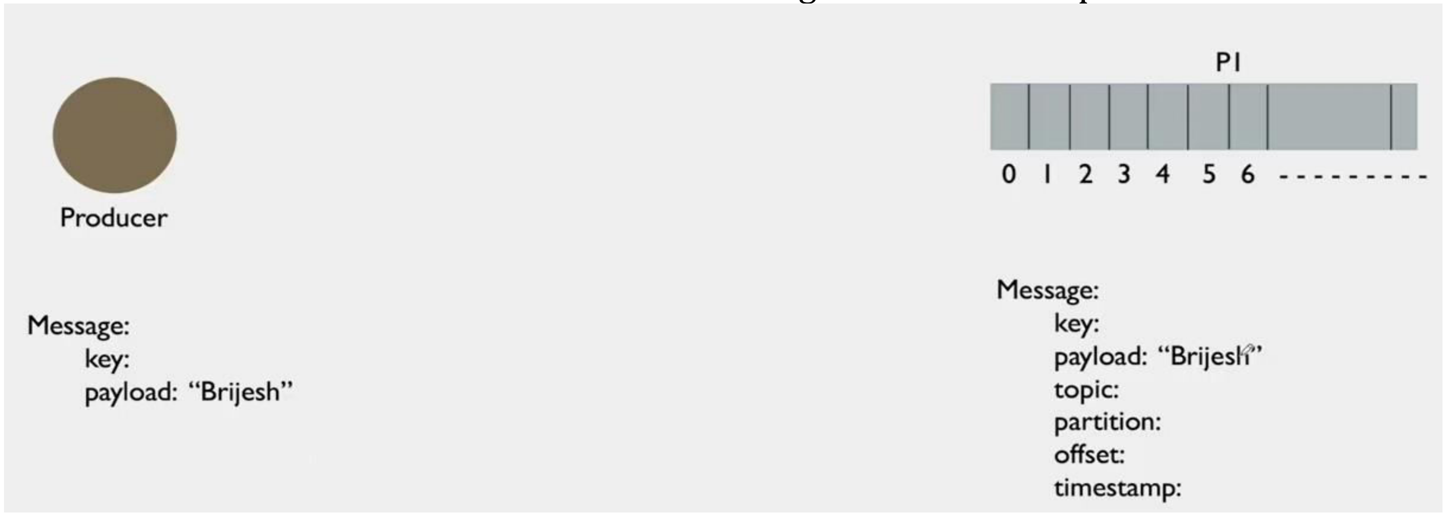
Verify reassignments:

Internals of Producer:



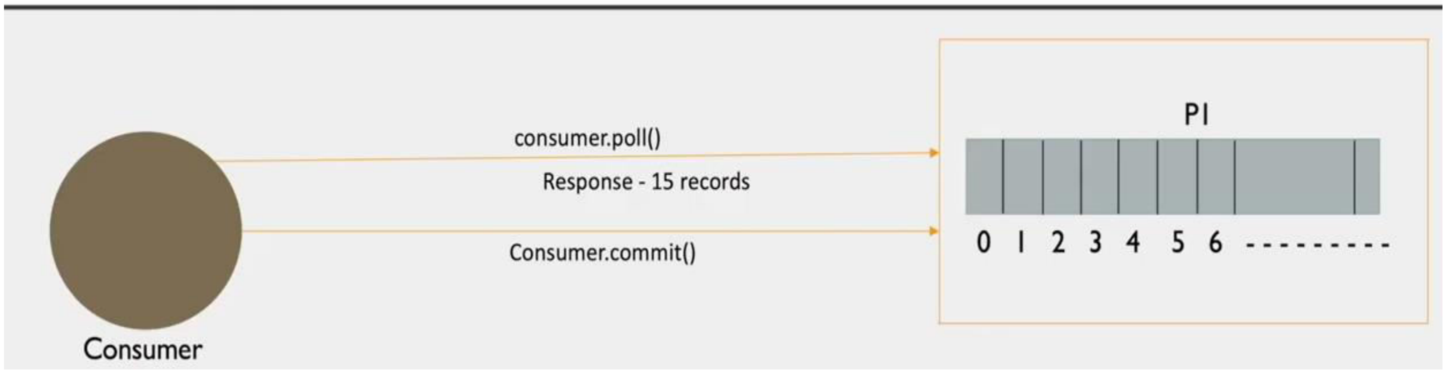
Offsets:
The records in the partitions are assigned a sequential id number called offset that uniquely identifies each record within the partition.
1) Log-end offset: Offset of the last message written to a log/partition.
2) Current offset: Pointer to the last record that Kafka has already sent to a 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.

INTERNALS WHEN PRODUCER SENDS MESSAGES

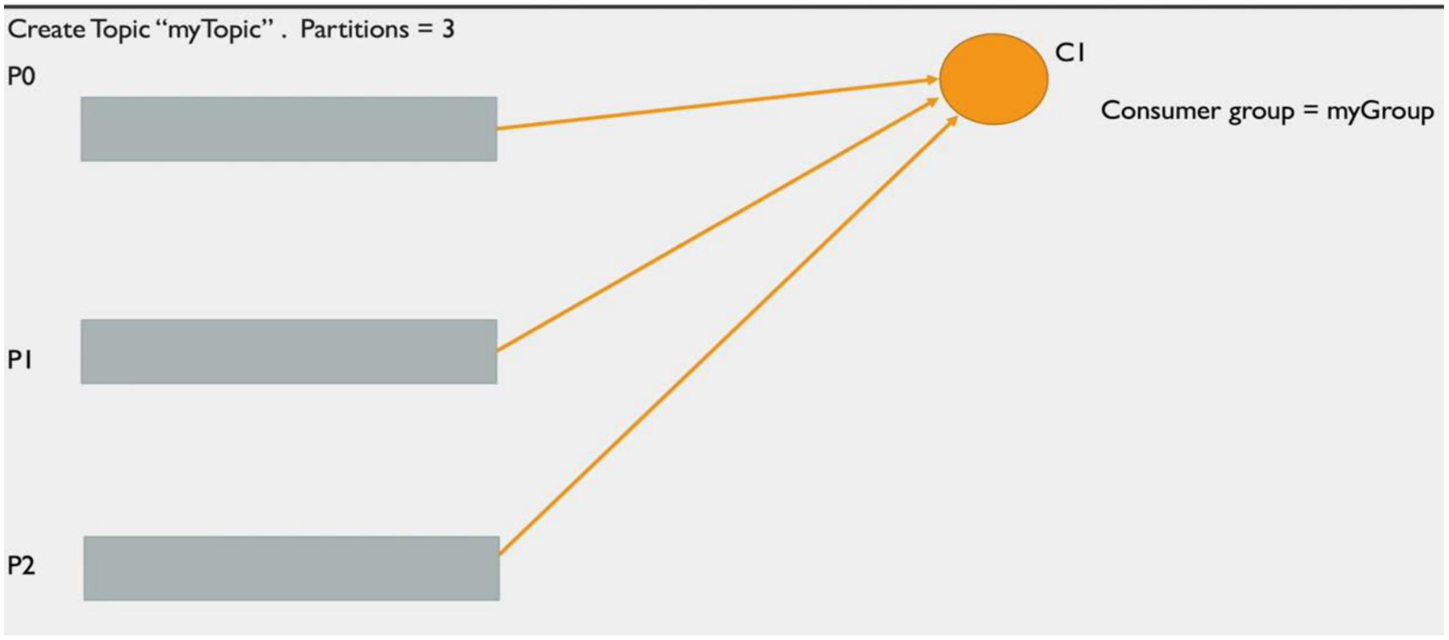


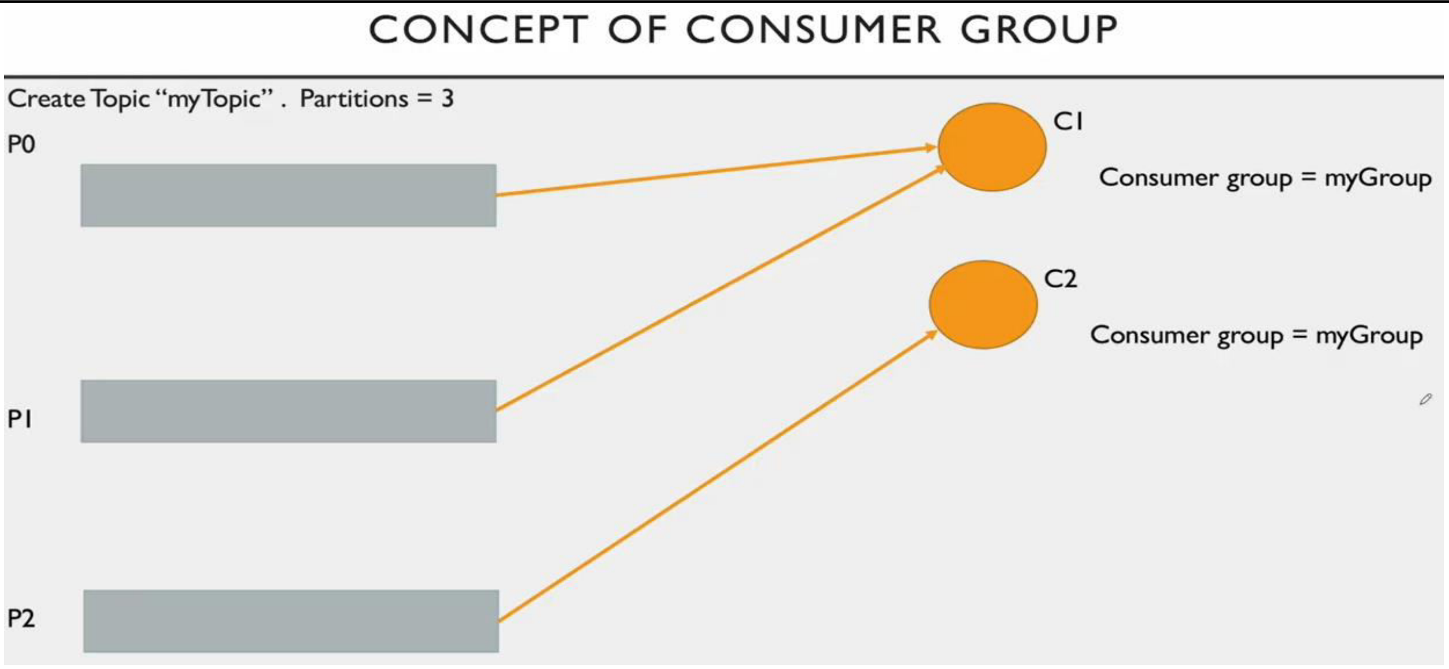
max.poll.response (15 records)

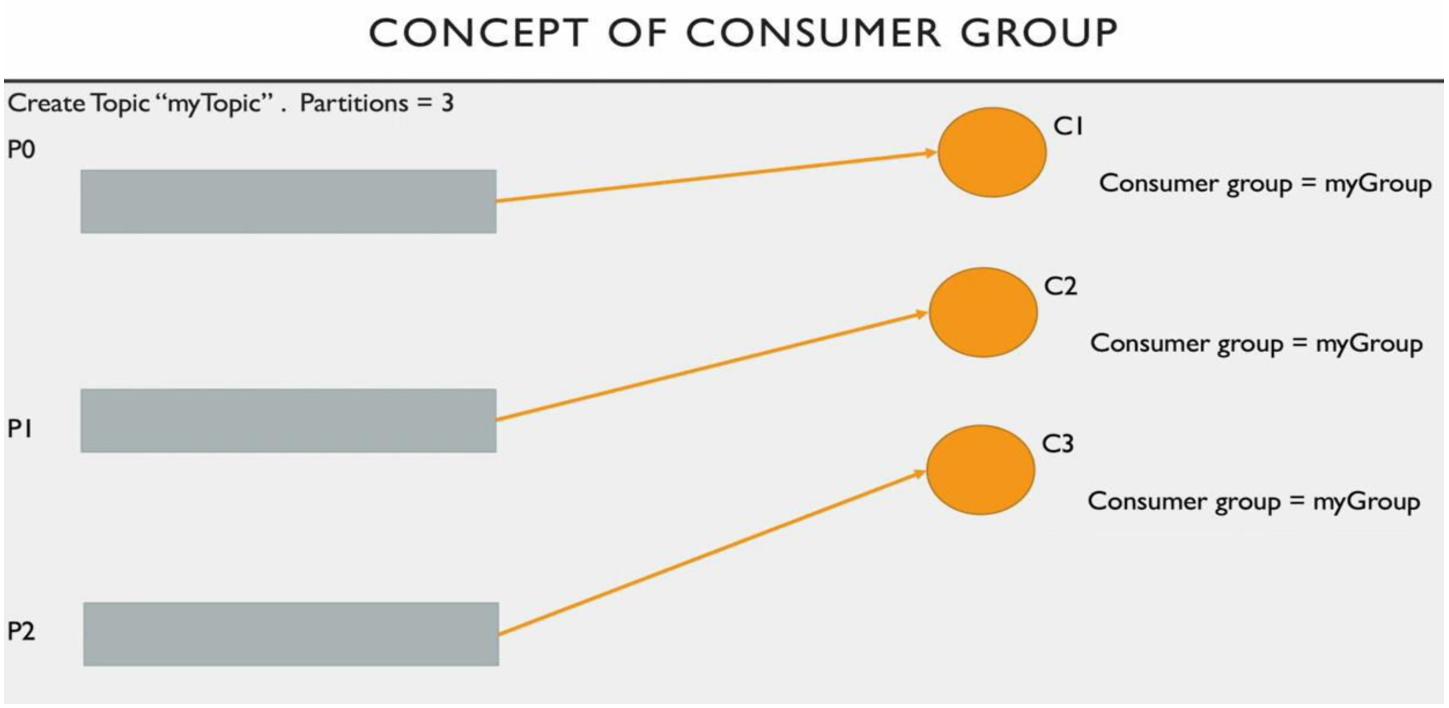
Offsets Cont'd…
2) Current offset: Pointer to the last record that Kafka has already sent to a consumer in the most recent poll.
3) Committed offset: Marking an offset as consumed is called committing an offset.

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.

CONCEPT OF CONSUMER GROUP




CONCEPT OF CONSUMER GROUP
Create Topic "myTopic" . Partitions = 3
P0
P1
P2
C1
Consumer group = myGroup
C2
Consumer group = myGroup



Consumer Group Rebalancing:
The process of re-distributing partitions to the consumers within a consumer group is known as Consumer Group Rebalancing.

Rebalancing of a consumer group happens in below cases:
1) A consumer joining the consumer group.
2) A consumer leaving the consumer group.
3) If partitions are added to the topics which these consumers are interested in.
4) If a partition goes in offline state.

Consumer Group Rebalancing:

Create Producer:

Create Consumer Group:

Produce Messages & see if the consumer is receiving or not.

Describe:

CONSUMER-ID

Create Consumer Group and check the Rebalancing:

Describe & See the partition balancing.

Create One more Consumer in the same Group & check Rebalancing:

Describe & See the partition balancing.

Delete Topic: