WISSEN

KAFKA

Know the basic building blocks used in Kafka and understanding the data In-Flow and In-Rest

www.wissen.com

KNOWLEDGE ▴ INSIGHT ▴ ACTION

Table of Contents

Abstract

Apache Kafka is an open-source distributed streaming platform. It provides a reliable, scalable, and robust solution for data streaming.

This paper explains the building blocks of Kafka and the journey of messages in and out of Kafka and refers to Kafka's version - 2.4.0.

Who should read this

Anyone interested in knowing the basic building blocks used in Kafka and understanding the data In-Flow and In-Rest in Kafka will find this paper useful.

Written by

Nikhil Kumar Summi

General Terms

Kafka Broker

It is a stateless instance of Kafka server running in a Kafka Cluster. It is capable of handling read and write operations on Kafka Topic(s). Each broker has a unique ID and can be responsible for partitions of one or more topic logs. Kafka Client (ex. Producers/Consumers) can bootstrap to a full Kafka cluster after connecting to any Broker.

Kafka Cluster

A Kafka cluster is made up of several brokers. They work in concert to form the cluster to achieve load balancing, reliable redundancy, and failover. To manage and coordinate stateless Brokers of a cluster, it uses Apache Zookeeper.

Zookeeper

A Kafka cluster is made up of several brokers. They work in concert to form the cluster to achieve load balancing, reliable redundancy, and failover. To manage and coordinate stateless Brokers of a cluster, it uses Apache Zookeeper.

- Leadership election of Kafka Broker and Topic Partition pairs.
- Cluster configuration & management.
- Failure detection and Recovery.
- Storage of ACLs.

Producer

A Kafka client that publishes records (messages/stream) to Kafka cluster is called Kafka Producer. It is thread-safe and serves as a data source that optimizes, writes, and publishes messages to one or more Kafka topics. It also serializes, compresses, and load-balance data among brokers through partitioning.

Topic Logs

Kafka stores Topics in logs. A Topic log is broken up into partitions. Kafka spreads log partitions across multiple servers or disks or brokers.

Consumer

A Kafka client that consumes records (messages/stream) from Kafka cluster is called Kafka Consumer. It connects to Kafka Brokers and consumes records from one or more Topic's Partitions. Each Consumer is associated with a Consumer Group. The Consumer sends an acknowledgment to Kafka on successful ingestion of the records.

Consumer Group

Kafka uses the concept of Consumer Groups to allow a pool of processes to divide the work of consuming and processing records. A Consumer Group has a unique id. Each Consumer Group is a subscriber to one or more Kafka topics and maintains its offset per topic partition. If you need multiple subscribers, then you have multiple Consumer Groups. A record gets delivered to only one Consumer in a Consumer Group. Each Consumer in a Consumer Group processes records and only one Consumer in that group will get the same record. Consumers in a Consumer Group load balance record processing.

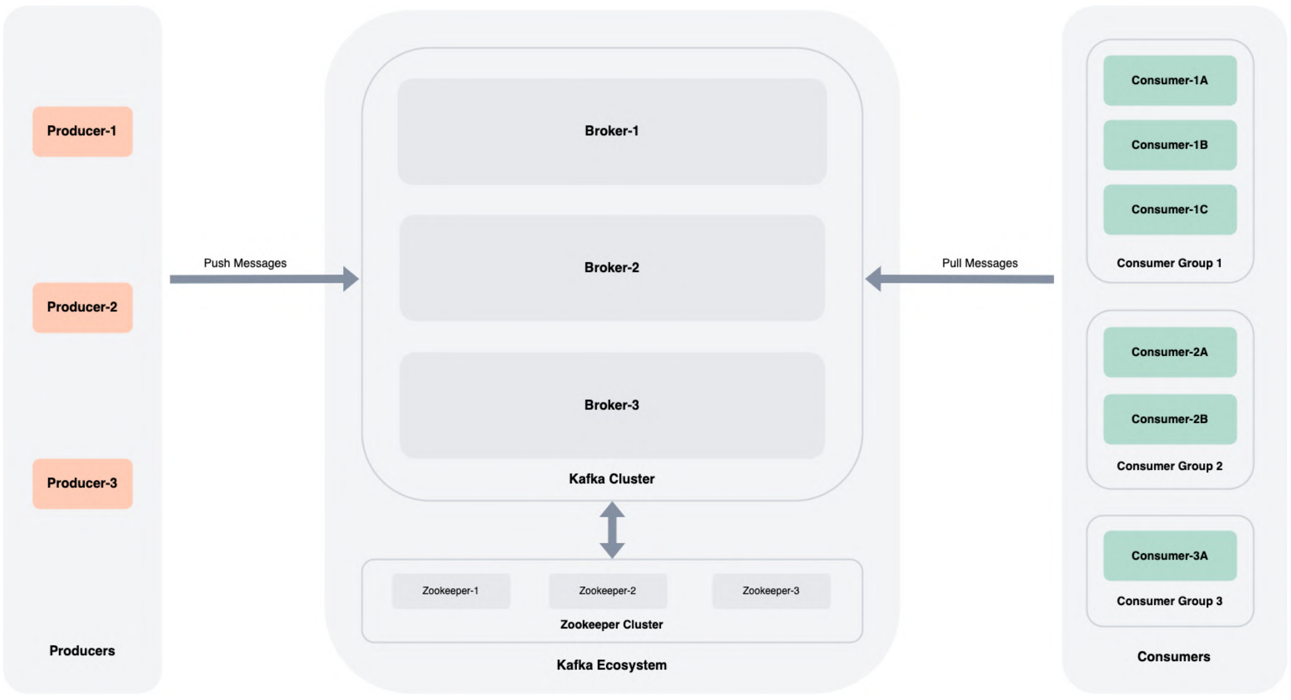
Topic

It is the name given to the channel through which records are streamed in Kafka. It is where Producers produce records and Consumers consume records. It has a unique name in the Kafka cluster. It can have zero or multiple subscribers (Consumers). Topics are broken up into partitions for speed, scalability, and size.

Given below is the visual of how the above components looks in the Kafka Ecosystem

Kafka Ecosystem

- Having 3 Brokers in Broker Cluster (Broker-1/2/3)
- 3 Zookeeper (Zookeeper-1/2/3)
- 3 Producers (Producer-1/2/3)
- 3 Consumer Groups (Consumer Group 1/2/3) having different set of Consumers.



Partition

Kafka Topic can be divided into Partitions. It is the physical space where data/message is published. These are distributed across Brokers in the cluster. Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log. A general formula to pick up the number of partitions is based on throughput needs. If partition keys are not used, then Kafka assigns the keys in a default manner making the ordering of messaging unguaranteed. Kafka uses partitions to scale a topic across many servers for producer writes. Also, Kafka uses partitions to facilitate parallel Consumers. Consumers consume records in parallel, up to the number of partitions. The order is guaranteed per partition. When partitioning by key, all records for the key will be on the same partition, which is useful if you need to replay the log. Kafka can replicate partitions to multiple brokers for failover.

Replica

Kafka can replicate partitions across a configurable number of Kafka servers which is used for fault tolerance. Each partition has a leader server and zero or more follower servers. Leaders handle all read and write requests for a partition. Followers replicate leaders and take over if the leader dies.

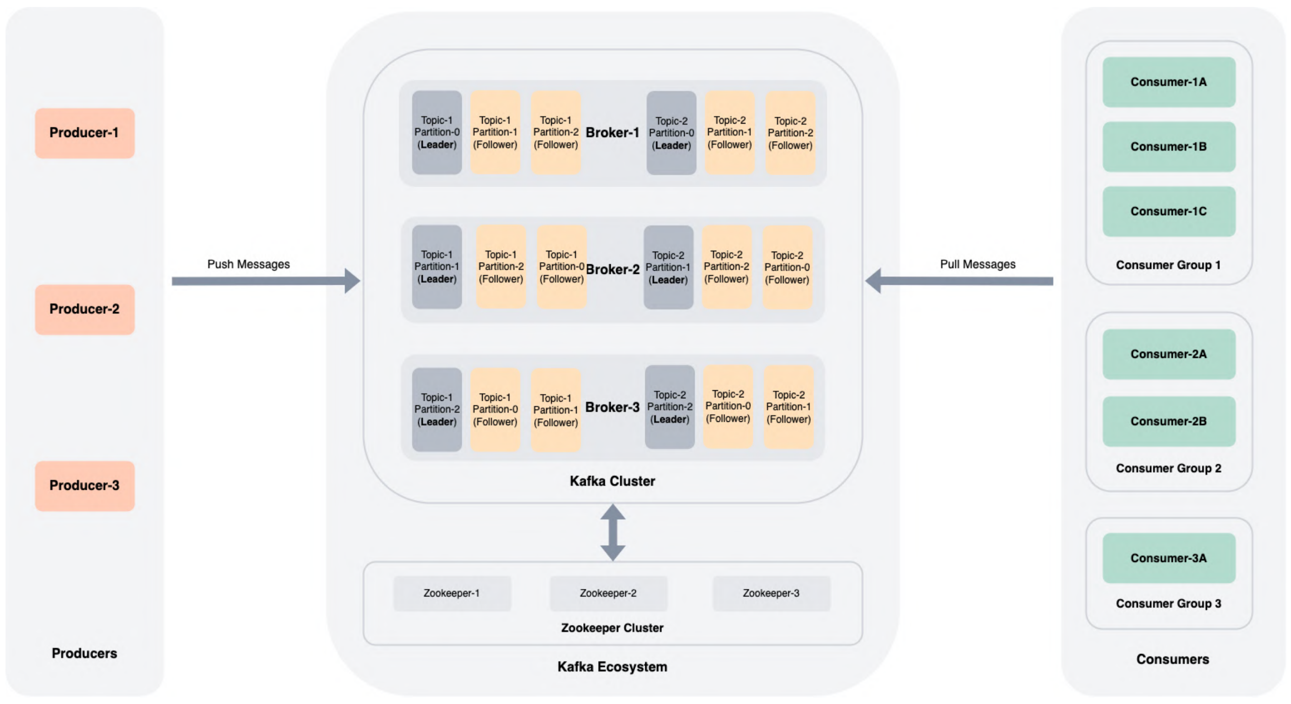
Leaders: Kafka chooses a replica of partition of a broker as Leader using Zookeeper. The broker that has the partition leader handles all reads and writes of records for the partition. Kafka replicates writes to the leader partition to followers (node/partition pair). A follower that is in sync is called an ISR (in-sync replica). If a partition leader fails, Kafka chooses a new ISR as the new leader.

Followers: All those replicas of the Broker's partition that is not a Leader will work as a Follower. Followers replicate leaders.

Given below is the visual of how the above components look in the updated Kafka Ecosystem

Kafka Ecosystem

- Having 3 Brokers in Broker Cluster (Broker-1/2/3)
- 3 Zookeeper (Zookeeper-1/2/3)
- 3 Producers (Producer-1/2/3)
- 3 Consumer Group having different set of Consumers
- 2 Topics (Topic-1/2) each having 3 partitions and 3 Replications


Producer-1
Producer-2
Producer-3
Producers
Push Messages
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pull Messages
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumers

Message

Message is the actual data/message that needs to be sent out to Kafka Producer. It is also named ProducerRecord. This consists of a topic name to which the record is being sent, an optional partition number, and an optional key and value. If a valid partition number is specified, then that partition will be used when sending the record. If no partition is specified but a key is present, then a partition will be chosen using the hash of the key. If neither key nor partition is present, then a partition will be assigned in a round-robin fashion.

Record Meta Data

It is Kafka server (Broker) response for the Producer sent message. It includes information about the Topic's partition, offset in partition, timestamp, etc.

Offset

Each record in the partitions is assigned a sequential id number called the offset that uniquely identifies each record within the partition.

Controller

The Controller is one of the brokers and is responsible for maintaining the leader/follower relationship for all the partitions. When a node shuts down, the Controller tells other replicas to become partition leaders to replace the partition leader on the node that is going away. Zookeeper is used to elect a controller, make sure there is only one, and elect a new one if it crashes. The Controller is responsible for admin operations such as partition assignment to broker, monitoring broker failures, etc.

Kafka Broker & Consumer Mapping Scenarios

Consumers of a Group are Equal to Partitions

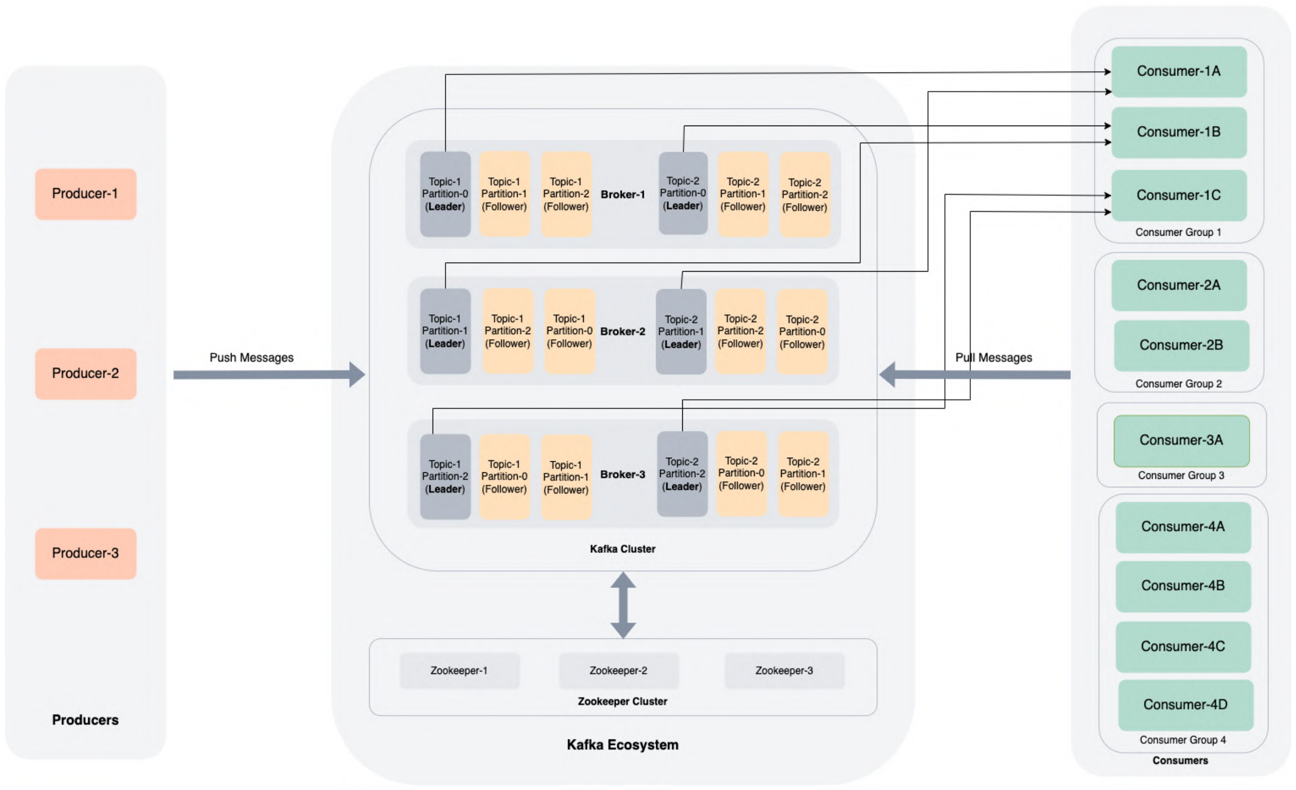
- One-to-one mapping of Consumer to Topic's partition
  - Each consumer subscribes to one partition and pulls messages from it in the order they are coming in that partition.

Kafka Ecosystem (Consumer Perpective)

- Having 3 Brokers in Broker Cluster (Broker-1/2/3)
- 3 Consumer Group having different set of Consumers
- 2 Topics (Topic-1/2) each having 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers
- Consumer Group 1 is active having 3 Consumers (Consumer-1A/B/C)

Scenario - Consumers are listening to both Topics and number of Consumers equal to number of Partitions.

- Consumer-1A is listening to Topic-1's Partitions-0 and Topic-2's Partition-1
- Consumer-1B is listening to Topic-1's Partitions-1 and Topic-2's Partition-0
- Consumer-1C is listening to Topic-1's Partitions-2 and Topic-2's Partition-2


Producer-1
Producer-2
Producer-3
Producers
Push Messages
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pull Messages
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumer-4A
Consumer-4B
Consumer-4C
Consumer-4D
Consumer Group 4
Consumers

Consumers of a Group are less than Topic's Partitions

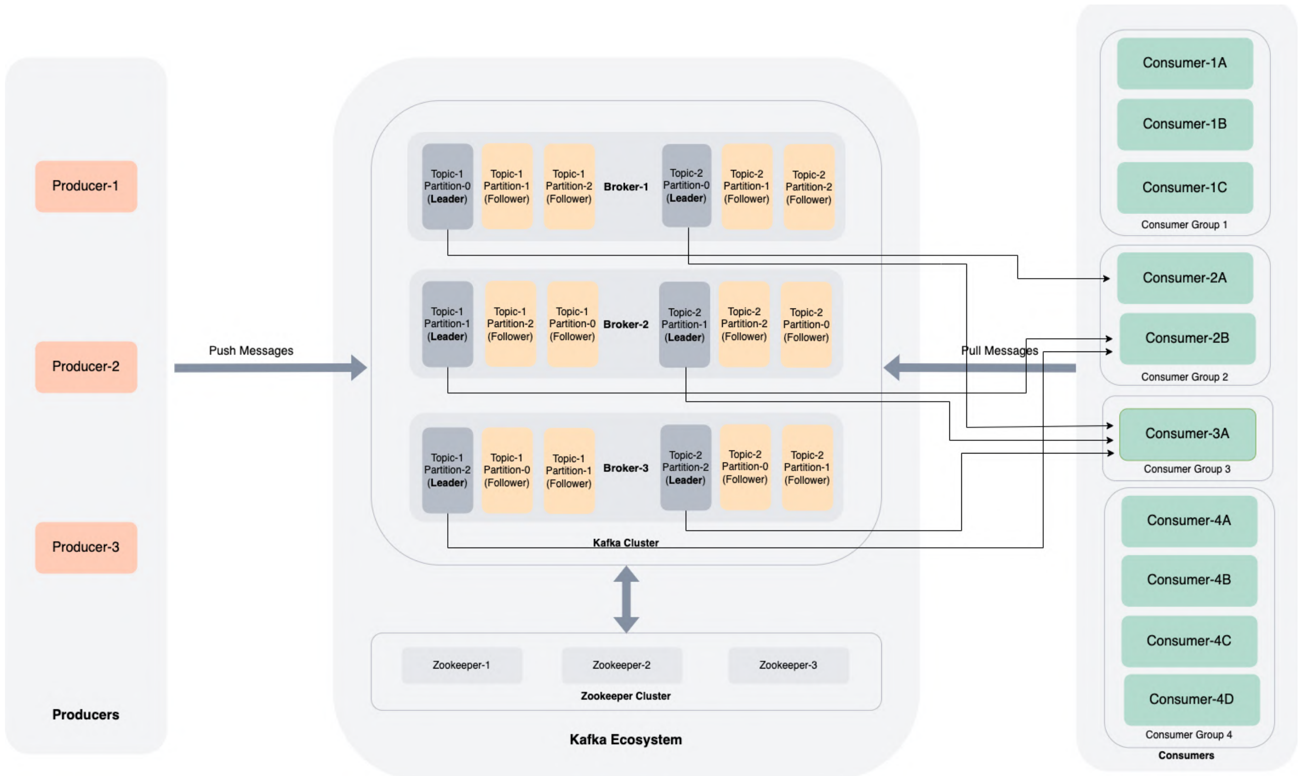
- One-to-Many mappings of Consumer to Topic's Partition
  - In this case, a single consumer may subscribe to multiple Partitions of a topic and pull data from them.

Kafka Ecosystem (Consumer Perpective)

- Having 3 Brokers in Broker Cluster (Broker-1/2/3)
- 3 Consumer Group having different set of Consumers
- 2 Topics (Topic-1/2) each having 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers
- Consumer Group 2/3 are active having different set of Consumers.

Scenario - Number of Consumers are less then number of Partitions.

- Consumer Group 2 is listening to Topic-1 (Consumer-2A is listening to Partition-0 and Consumer-2B is listening to Parition-1/2)
- Consumer Group 3 is listening to Topic-2 (Consumer-3A is listening to all Partitions (0/1/2)


Producer-1
Producer-2
Producer-3
Producers
Push Messages
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pull Messages
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumer-4A
Consumer-4B
Consumer-4C
Consumer-4D
Consumer Group 4
Consumers

Consumers of a Group are more than the Topic's Partitions

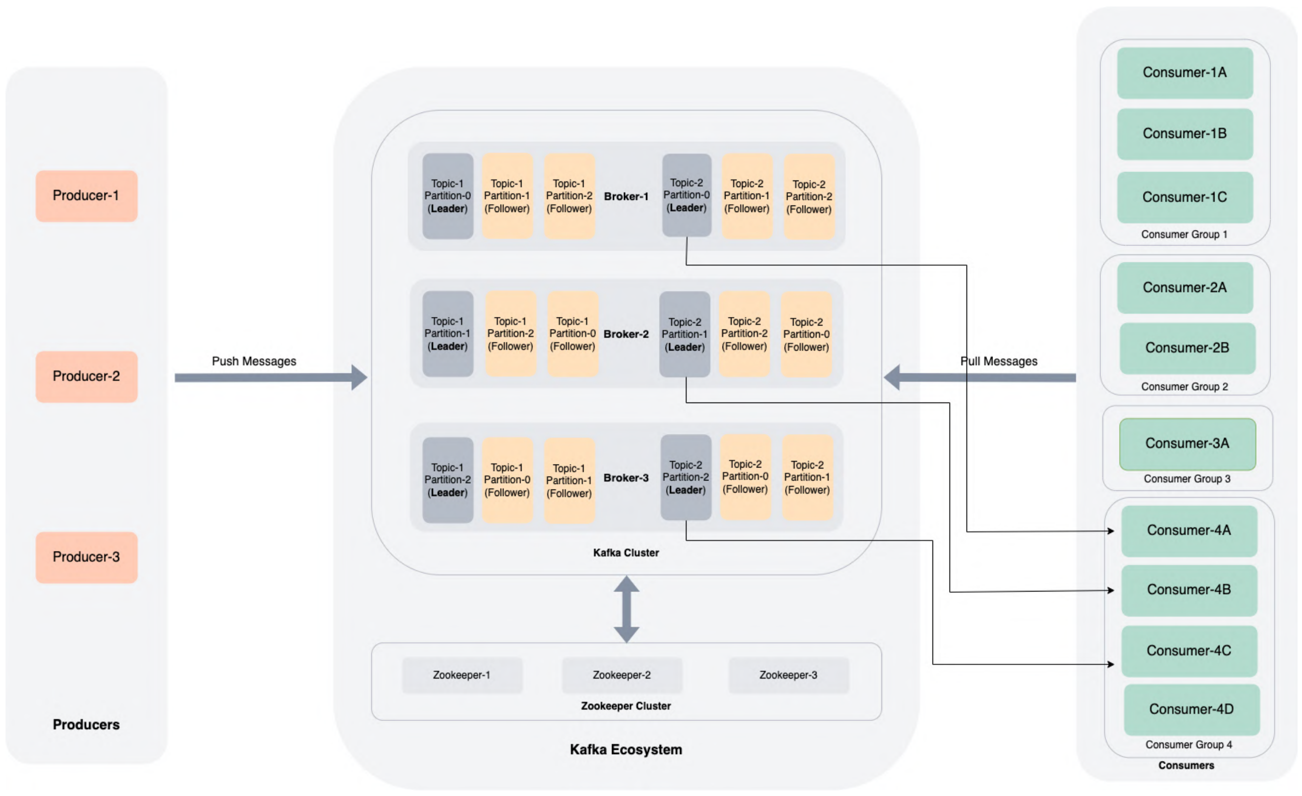
- Extra Consumer will be ideal in this case
  - Extra consumer will not add any value as it is not going to pull any data from Kafka.

Kafka Ecosystem (Consumer Perpective)

- Having 3 Brokers in Broker Cluster (Broker-1/2/3)
- 3 Consumer Group having different set of Consumers
- 2 Topics (Topic-1/2) each having 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers
- Consumer Group 4 is active and listening to Topic-2.

Scenario - Number of Consumers are more than number of Partitions.

- Consumer-4A is listening to Topic-2's Partition-0
- Consumer-4B is listening to Topic-2's Partition-1
- Consumer-4C is listening to Topic-2's Partition-2
- Consumer-4D is ideal



Producer-1
Producer-2
Producer-3
Producers
Push Messages
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pull Messages
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumer-4A
Consumer-4B
Consumer-4C
Consumer-4D
Consumer Group 4
Consumers

Kafka Producer & Broker Mapping Scenarios

Message (ProducerRecord) having Topic & Value

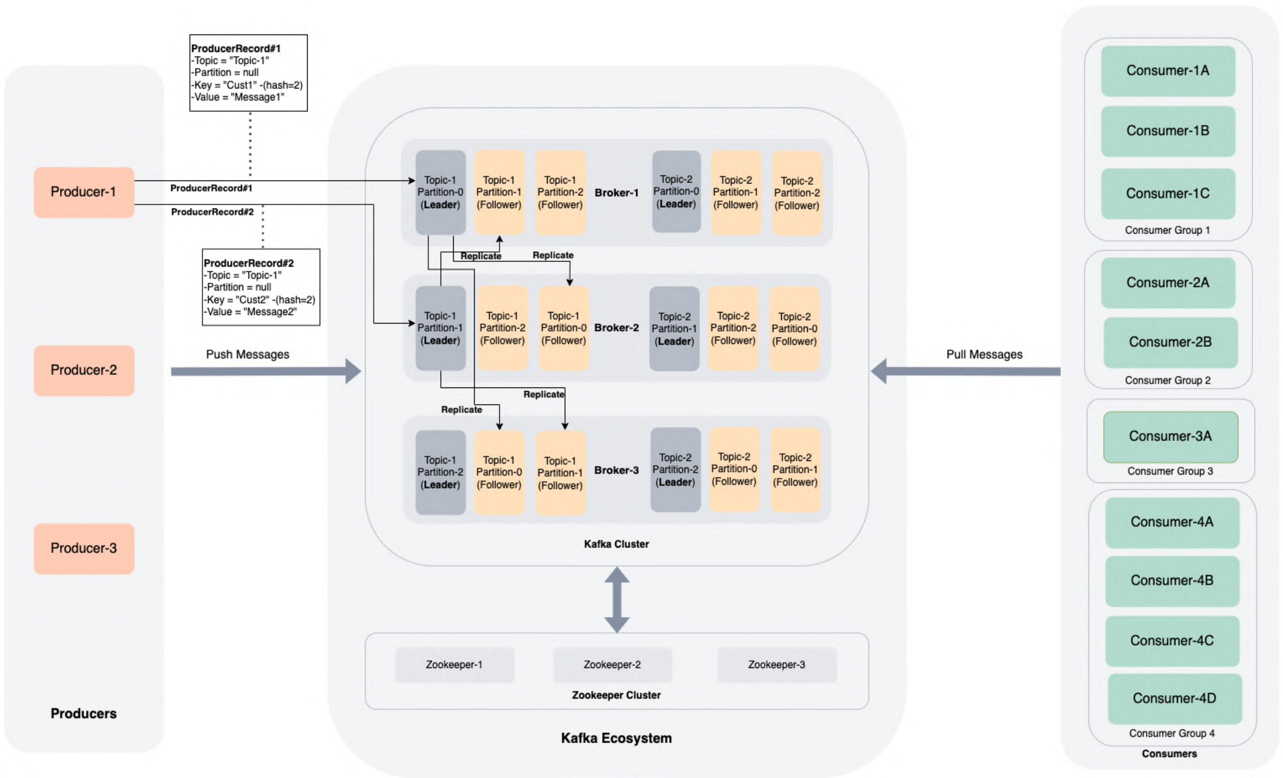
- Message pushed to Topic's Partition in round robin.

Kafka Ecosystem (Producer Perspective)

- Having 3 Brokers (Broker-1/2/3)
- 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing message to Topic-1

Scenario: - Producer Record (Message) have Topic and Value only.

- Example Topic = Topic-1 and Value = "Message1", "Message2"
- Message will be pushed to partition in round robin selection.



Message having Topic, Partition & Value

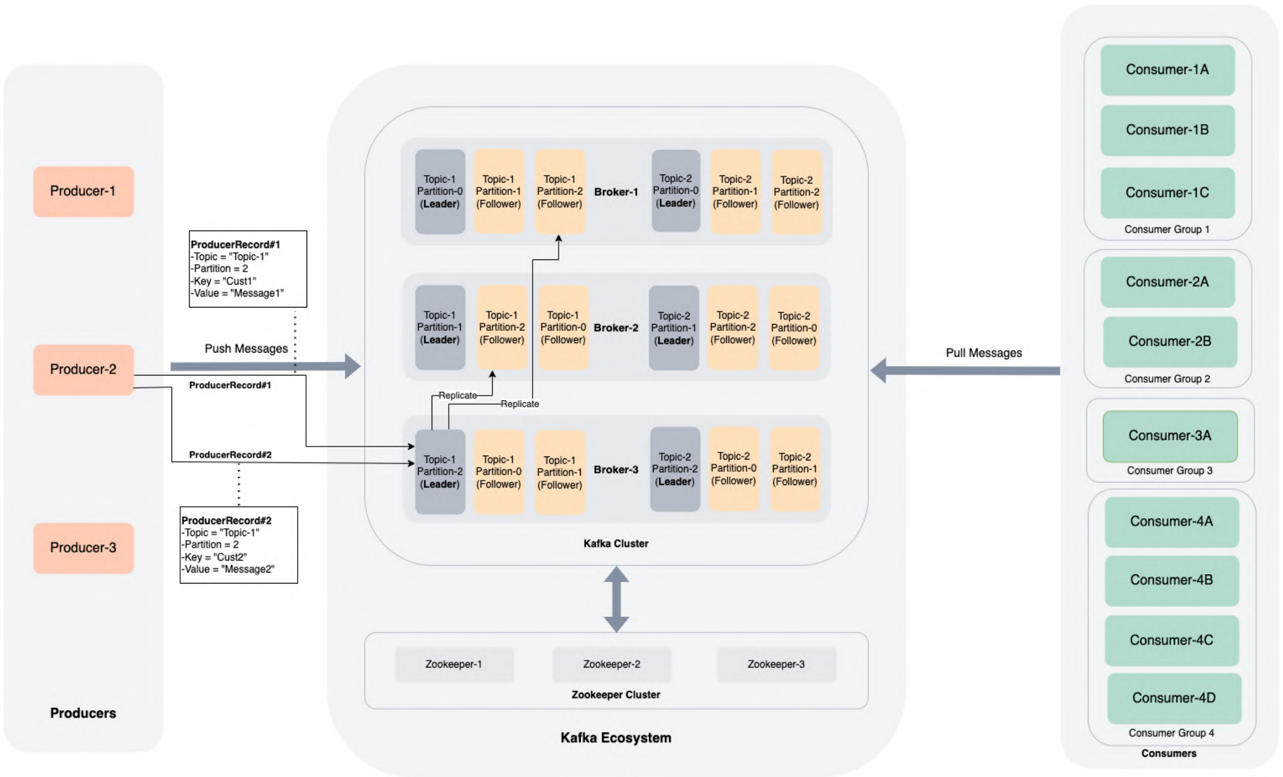
- All messages will be pushed to provided Topic's Partition. It will guarantee the ordering of messages in Kafka and to Consumers.

Kafka Ecosystem (Producer Perspective)

- Having 3 Brokers (Broker-1/2/3)
- 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing message to Topic-1

Scenario: - Producer Record (Message) have Topic and Value only.

- Example Topic = Topic-1, Partition-2 and Value = "Message1", "Message2"
- All message will be sent to same partition so it follow the ordering of message.



Message having Topic, Key & Value

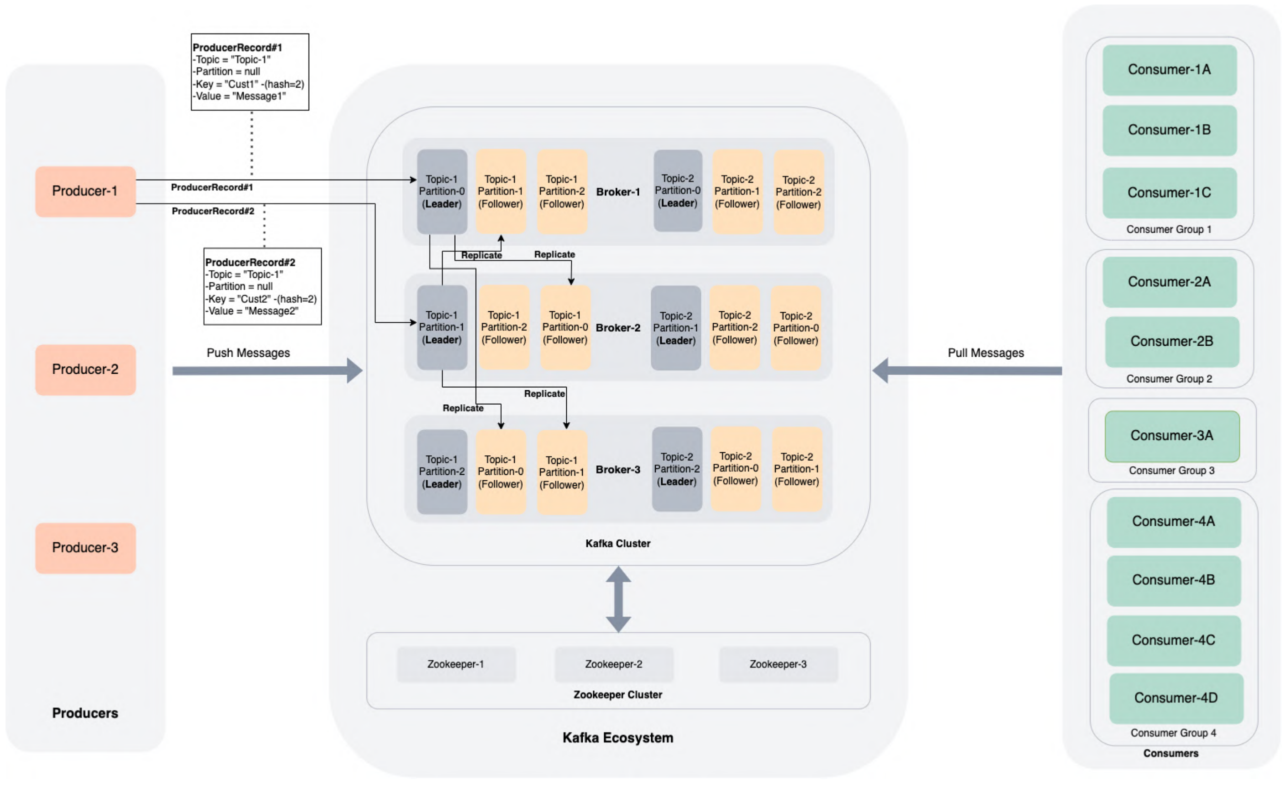
- Message pushed to Topic's Partition as per Key's hash. It will guarantee the ordering of messages for the same key.

Kafka Ecosystem (Producer Perspective)

- Having 3 Brokers (Broker-1/2/3)
- And 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing message to Topic-1

Scenario: - ProducerRecord (Message) have Topic, Key and Value only.

- ProducerRecord#1 having Topic = "Topic-1", Key = "Cust1" and Value = "Message1"
- ProducerRecord#2 having Topic = "Topic-1", Key = "Cust2" and Value = "Message2"
- Message will be pushed to partition as per the Hash value of Key.



Message having Topic, Partition, Key & Value

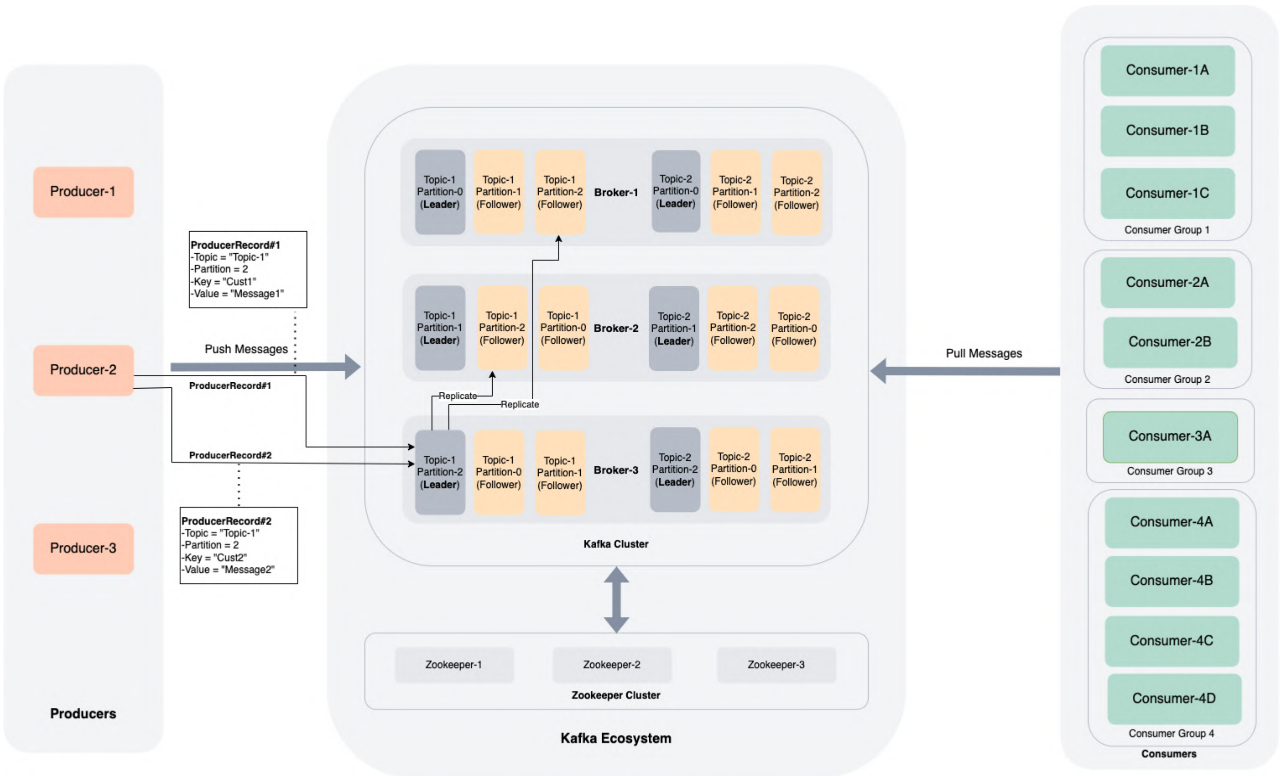
- Producer will ignore Key and consider Partition while pushing a message to Topic

Kafka Ecosystem (Producer Perspective)

- Having 3 Brokers (Broker-1/2/3)
- And 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-2 is active and publishing/pushing message to Topic-1's Partition-2

Scenario: - ProducerRecord (Message) have Topic, Partition, Key and Value only.

- ProducerRecord#1 is having Topic = Topic-1, Partition-2 , Key = "Cust1" and Value = "Message1"
- ProducerRecord#1 is having Topic = Topic-1, Partition-2 , Key = "Cust2" and Value = "Message2"
- It will ignore the Key and all message will be sent to same partition so it follow the ordering of message.



Journey of Message

From Producer to Partition

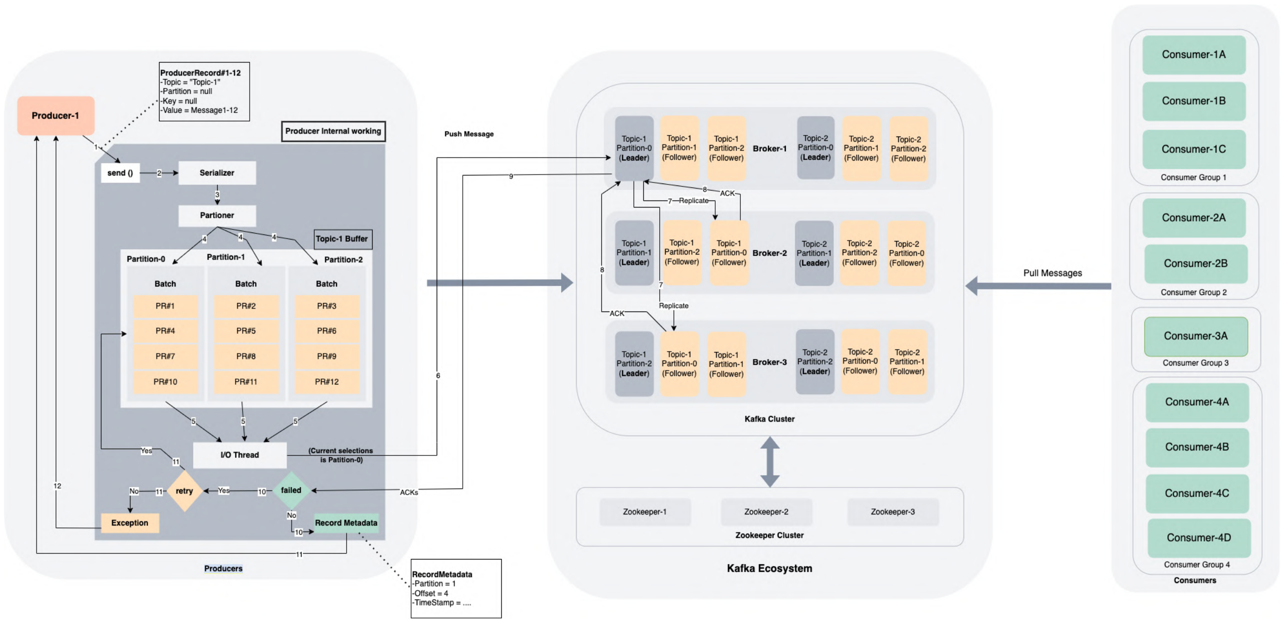
Producer's Send method performs several things before pushing a message to the Kafka cluster. Below are some of the primary tasks performed by the Producer

- **Serializing** – It serializes the Key and Value before pushing data.
- **Partitioning** – It takes care of assigning the Topic's partition to message.
- **Buffering/Batching** - To provide throughput, it maintains the buffer of a batch of non-pushed messages.
- **I/O** - It takes care of pushing message to Kafka from the buffer.
- **Retyring** – If configured, it pushes the message again for retrying. It happens in case of failure or NAK (not acknowledged by Partition or its replicas).

Kafka Ecosystem (Journey of Message from Producer to Broker)

- Having 3 Brokers (Broker-1/2/3)
- And 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing messages to Topic-1 and produce's enable.idempotent= true

Scenario: - ProducerRecord (Message) have Topic, Partition, Key and Value only.



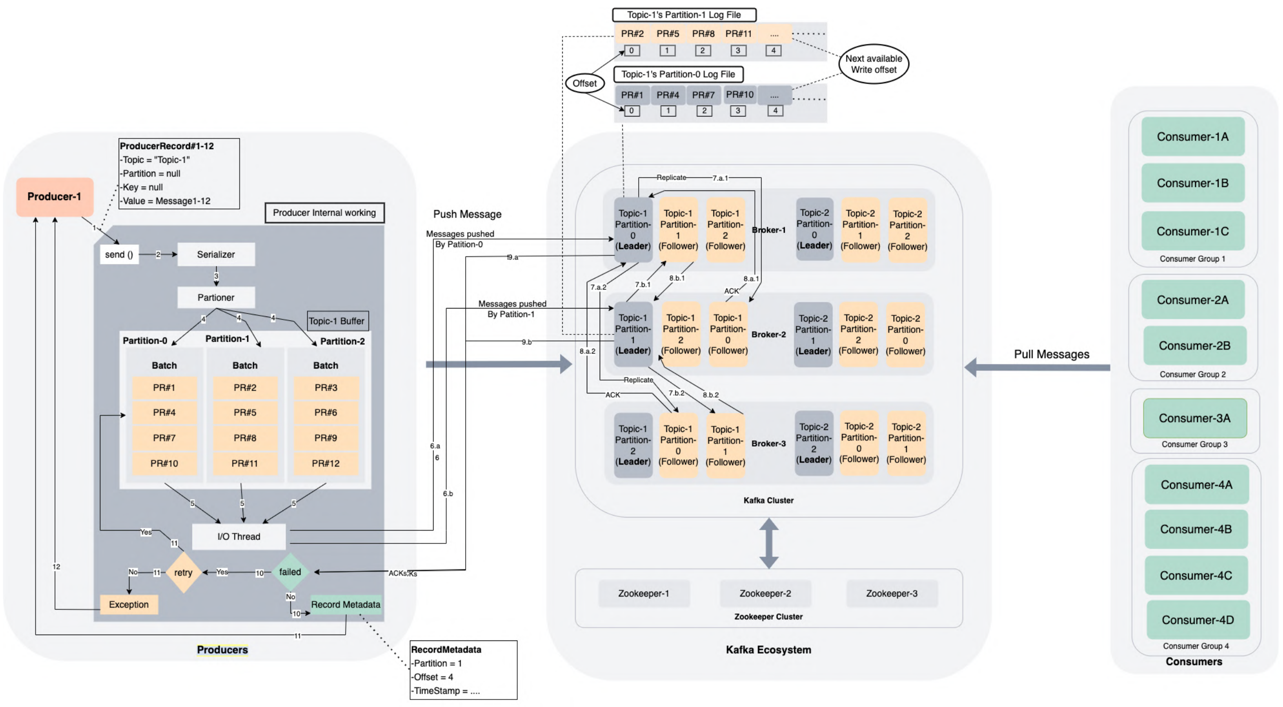
At Rest in Topic's Partition

Message is at rest in Topic's partitions. Each message will get a unique sequential offset number in the partition. It also confirms the replications and acks.

Kafka Cluster (Message at Rest in Topic's Partition)

- Having 3 Brokers (Broker-1/2/3)
- 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing messages to Topic-1 and produce's enable.idempotent= true
- Messages (PR#1/4/7/10 from Partition-0 Buffer and PR#2/5/8/11 from Partition-1) pushed to Brokers

Scenario: - Kafka Messages in Topic's Partition's Log file.


Topic-1's Partition-1 Log File
PR#2 PR#5 PR#8 PR#11 ....
0 1 2 3 4
Offset
Next available Write offset
Topic-1's Partition-0 Log File
PR#1 PR#4 PR#7 PR#10 ....
0 1 2 3 4
Producer-1
ProducerRecord#1-12
-Topic = "Topic-1"
-Partition = null
-Key = null
-Value = Message1-12
Producer Internal working
send ()
Serializer
Partioner
Topic-1 Buffer
Partition-0
Partition-1
Partition-2
Batch
PR#1 PR#4 PR#7 PR#10
PR#2 PR#5 PR#8 PR#11
PR#3 PR#6 PR#9 PR#12
I/O Thread
retry
failed
Yes
No
Exception
Record Metadata
ACKs
Producers
RecordMetadata
-Partition = 1
-Offset = 4
-TimeStamp = ...
Push Message
Messages pushed By Patition-0
Messages pushed By Patition-1
Replicate
ACK
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pull Messages
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumer-4A
Consumer-4B
Consumer-4C
Consumer-4D
Consumer Group 4
Consumers

From Partition to Consumers

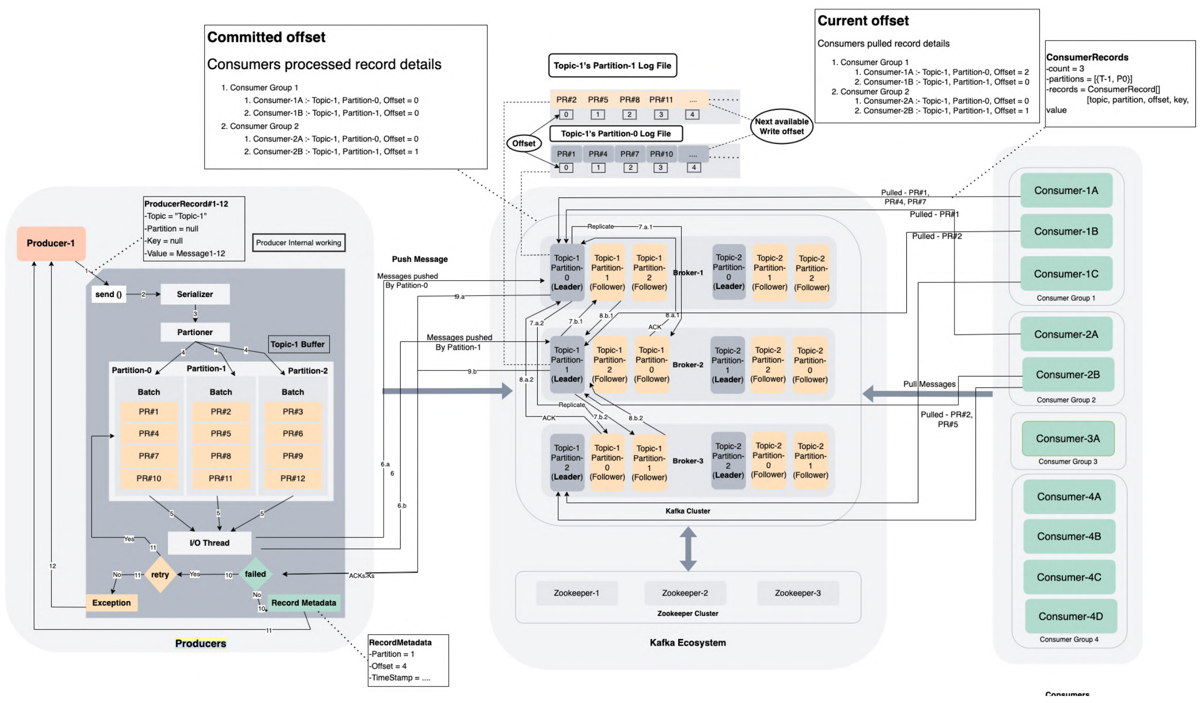
Consumers pull the records from subscribed Topic's partition(s). Kafka maintains the information about the Consumers' current and committed offset information.

Kafka Ecosystem (Consumers Reading Message from Topic's Partition)

- Having 3 Brokers (Broker-1/2/3)
- And 2 Topics (Topic-1/2), Each Topic has 3 Partitions (Partition-0/1/2) and 3 Replicas (1 Leader and 2 Followers)
- Producer-1 is active and publishing/pushing messages to Topic-1 and produce's enable.idempotent= true
- Messages (PR#1/4/7/10 from Partition-0 Buffer and PR#2/5/8/11 from Partition-1) pushed to Brokers.
    - Consumer Group 1 and 2 are active and pulling messages from Topic-1
    - Consumer Group 1 :- Consumer-1A pulling from Partition-0, Consumer-1B from Partition-1 and Consumer-1C from Partition-2
- Consumer Group 2 :- Consumer-2A pulling from Partition-0, Consumer-2B from Partition-1 and Partition-3

Scenario: - Kafka Messages reads by Consumer from Topic's Partition's Log file.

- Consumer-1A pulled PR#1, PR#4, PR#7 --- Committed PR#1 | Consumer-1B pulled PR#2 --- Committed PR#2
- Consumer-2A pulled PR#1 --- Committed PR#1 | Consumer-2B pulled PR#2, PR#5 --- Committed PR#2, PR#5


Committed offset
Consumers processed record details
1. Consumer Group 1
1. Consumer-1A :- Topic-1, Partition-0, Offset = 0
2. Consumer-1B :- Topic-1, Partition-1, Offset = 0
2. Consumer Group 2
1. Consumer-2A :- Topic-1, Partition-0, Offset = 0
2. Consumer-2B :- Topic-1, Partition-1, Offset = 1
Topic-1's Partition-1 Log File
PR#2
PR#5
PR#8
PR#11
....
0
1
2
3
4
Offset
Topic-1's Partition-0 Log File
PR#1
PR#4
PR#7
PR#10
....
Next available Write offset
Current offset
Consumers pulled record details
1. Consumer Group 1
1. Consumer-1A :- Topic-1, Partition-0, Offset = 2
2. Consumer-1B :- Topic-1, Partition-1, Offset = 0
2. Consumer Group 2
1. Consumer-2A :- Topic-1, Partition-0, Offset = 0
2. Consumer-2B :- Topic-1, Partition-1, Offset = 1
ConsumerRecords
-count = 3
-partitions = [{T-1, P0}]
-records = ConsumerRecord[]
[topic, partition, offset, key, value
ProducerRecord#1-12
-Topic = "Topic-1"
-Partition = null
-Key = null
-Value = Message1-12
Producer Internal working
Producer-1
send ()
Serializer
Partioner
Topic-1 Buffer
Partition-0
Partition-1
Partition-2
Batch
PR#1
PR#4
PR#7
PR#10
PR#2
PR#5
PR#8
PR#11
PR#3
PR#6
PR#9
PR#12
I/O Thread
Yes
No
retry
failed
Exception
Record Metadata
ACKs.Ks
Producers
RecordMetadata
-Partition = 1
-Offset = 4
-TimeStamp = ...
Push Message
Messages pushed By Patition-0
Messages pushed By Patition-1
9.a
9.b
6.a
6
6.b
7.a.1
7.a.2
7.b.1
7.b.2
8.a.1
8.a.2
8.b.1
8.b.2
Replicate
ACK
Topic-1 Partition-0 (Leader)
Topic-1 Partition-1 (Follower)
Topic-1 Partition-2 (Follower)
Broker-1
Topic-2 Partition-0 (Leader)
Topic-2 Partition-1 (Follower)
Topic-2 Partition-2 (Follower)
Topic-1 Partition-1 (Leader)
Topic-1 Partition-2 (Follower)
Topic-1 Partition-0 (Follower)
Broker-2
Topic-2 Partition-1 (Leader)
Topic-2 Partition-2 (Follower)
Topic-2 Partition-0 (Follower)
Topic-1 Partition-2 (Leader)
Topic-1 Partition-0 (Follower)
Topic-1 Partition-1 (Follower)
Broker-3
Topic-2 Partition-2 (Leader)
Topic-2 Partition-0 (Follower)
Topic-2 Partition-1 (Follower)
Kafka Cluster
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper Cluster
Kafka Ecosystem
Pulled - PR#1, PR#4, PR#7
Pulled - PR#1
Pulled - PR#2
Pull Messages
Pulled - PR#2, PR#5
Consumer-1A
Consumer-1B
Consumer-1C
Consumer Group 1
Consumer-2A
Consumer-2B
Consumer Group 2
Consumer-3A
Consumer Group 3
Consumer-4A
Consumer-4B
Consumer-4C
Consumer-4D
Consumer Group 4
Consumers

Comparison

ActiveMQ

It is the most popular open-source, multi-protocol, Java-based message broker. It supports multiple messaging Protocol.

- ActiveMQ supports both Queues (for P2P) and Topics (for Pub/Sub). On the other hand, Kafka supports Pub/Sub.
- Message delivery is the responsibility of Producer to ensure its delivery in the case of ActiveMQ. On the other hand, it is Kafka's Consumer responsibility to consume the message.
- ActiveMQ works on Push-based messages processing. On the other hand, Kafka works on Pull-based messages processing.
- ActiveMQ broker has to maintain the delivery state of every Message, so its throughput is slow. On the other hand, Kafka Broker only maintains the offset of Message, so it is lightweight.
- ActiveMQ is useful in Traditional messaging, on the other hand, Kafka is useful in Distributed Event Streaming.
- ActiveMQ does not guarantee the Ordering of messages. On the other hand, Kafka guarantees the Ordering of messages at the Partition level.

RabbitMQ

It is an open-source message broker. RabbitMQ is lightweight and easy to deploy on-premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.

- RabbitMQ uses Req/Res, P2P and Pub/Sub communication patterns. On the other hand, Kafka supports Pub/Sub.
- RabbitMQ broker works on Push-based mechanisms. Kafka Broker works on Pull-based mechanisms.
- RabbitMQ works in a model in which Broker is smart and Consumer is dump. Here Broker delivers the message, and the consumer is quite simple. On the other hand, in Kafka, Broker is dump and Consumer is Smart which keeps pulling messages from Broker.
- RabbitMQ does not support Message ordering.
- RabbitMQ works on Queues - Messages are done away with after they are pushed to Consumer and acknowledged. On the other hand, Kafka maintains logs of messages.
- RabbitMQ provides features to prioritize the messages. Kafka does not support this feature.

References

- Kafka Consumer Architecture - Consumer Groups and subscriptions
- Introduction to Topic Log Compaction in Apache Kafka
- Role of Apache ZooKeeper in Kafka – Monitoring & Configuration
- 10 Configs to Make Your Kafka Producer More Resilient
- Preferred Replica Leader Election Tool
- Kafka vs. RabbitMQ: Architecture, Performance & Use Cases

About Wissen

Established in the year 2000 in the US, Wissen is an Information Technology company headquartered in Bangalore, India. With global offices in US, India, UK, Australia, Mexico and Canada, Wissen is an end-to-end solution provider for companies in sectors such as Banking and Financial Services, Telecom, Healthcare, Manufacturing and Energy verticals. With best in class infrastructure and development facilities spread across the globe, the company has successfully delivered $1 billion worth of projects for more than 25 of the Fortune 500 companies. Wissen's 4500+ highly skilled professionals, a strong leadership team, and technology expertise help clients build enterprise systems, implement a modern digital strategy, and gain a competitive advantage with business transformation.

Our service offerings

Application Development

Big Data and Analytics

Artificial Intelligence and Machine Learning

Robotic Process Automation

Agile and DevOps

Visualization and Business Intelligence

Cloud and Mobility

Quality Assurance and Test Automation

Infrastructure Management

hello@wissen.com

WISSEN

www.wissen.com

India | UK | USA | Australia | Canada | Mexico | Vietnam