**KAFKA**

## Data Processing type:

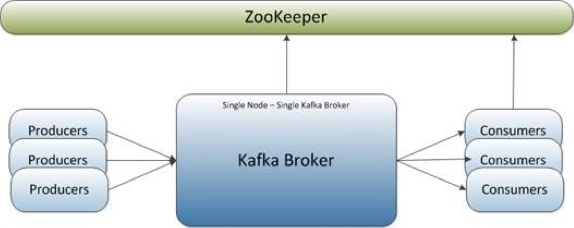
* + **Batch Processing:** processing of history data (ETL, SpringBatch)
  + **Stream Processing:** processing of real time data (Kafka)

## What is Kafka?

* Apachee kafka is fast, scalable (performance can be improved using many options like increasing consumers, broker, topic, partitions etc) durable, fault tolerance (because of replication factor) publish subscribe messaging system.
* It’s a distributed messaging system. (Data can be placed across multiple nodes, can be distributed across multiple topics and partitions and data can be inserted/fetched by multiple producers, consumer.
* It is designed to move data at high volume.
* Designed for big data.
* Originally developed by LinkedIn

## Kafka Terms:

* **Producer**: It writes message to topic.
* **Consumer**: It read data from topic.
* **Broker:** A broker is a process running on **single** machine. It is designated for Zookeeper. A zookeeper is like ATC in airport who manages all the incoming and outgoing flight landing and take-off.
* **Message:** Information that send from producer to consumer through kafka.
* **Topic:** A topic is feed name where messages are stored and publish. It contains **unistructural** data.
  + **Simit**
  + **Monika**
  + **Amit,10,2555,20000.00**
  + **Rahul 25 669 666999.00**
* **Topic Partitions:**  A topic can be divided into multiple partitions for parallel processing. Data is divided into each partition.
* **Replicas:** A replica of a partitions is backup of a partition; it allows to achieve fault tolerant.
* **Consumer Group:** A consumer groups is set of consumers that are subscribe to specific partitions.
* **Offset:** It is a unique identifier of a record within a partition. It denotes **the position of the consumer in the partition**.
* **Node:**  A node is single computer in the Apache kafka cluster.
* **Cluster**:A cluster is group of nodes.

****

## Kafka Broker and Zookeeper:

* Multiple Broker can be in single machine.
* Messages of topic spread across partitions and partitions spread across broker.
* Each broker handles many partitions.
* Multiple backups of the topics partitions are created in multiple broker based on the number of replication factors.
* Load balancing as data divide into partitions.
* If broker goes down, backup partition would be elected based on leader and follower.

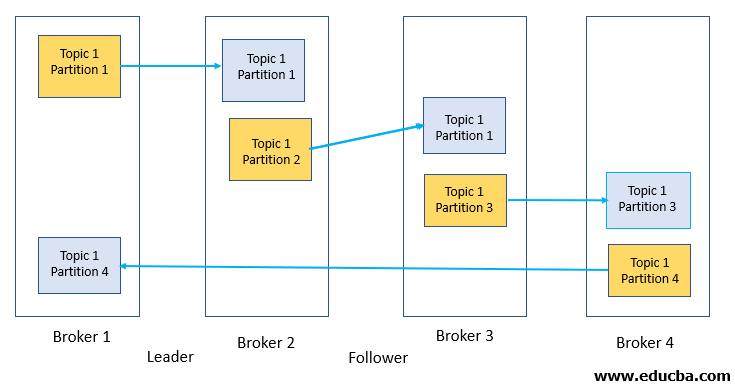
**NOTE: We have single node single broker, single node multiple brokers, multiple node multiple brokers.**

## Understanding Replication in Kafka

Each partition has one server which acts as the "**leader**" and zero or more servers which act as "**followers**". The leader handles all read and write requests for the partition while the followers passively replicate the leader. If the leader fails, one of the followers will **automatically** become the new leader.

* The kafka cluster has 4 brokers.
* The topic has 4 partitions
* The replication factor is 2.

**NOTE: Broker is associated with partitions not with topics.**



## Understanding Retention Policy

* By default, kafka has 7 days of retention policy, we can increase it inside zookeeper. Properties file

## Understanding Consumer Offset

* Consumer have three options to read

• from-beginning

• latest

• specific offset

## Understanding Consumers and Consumer Groups

**Consumer Groups:** Consumer groups is a place where we create multiple **instances** of **a** consumer. (Note here, we do not create multiple consumers, we create multiple instances of same consumer).

Whenever we create consumer in kafka, each consumer will be created in a separate group.

So, in order to create multiple instances for same consumer, we pass **group\_id** at the **end** while creating consumer to make sure current consumer will be part of **same group**.

* Who manages the consumer group?

• Kafka Broker manages the consumer-groups

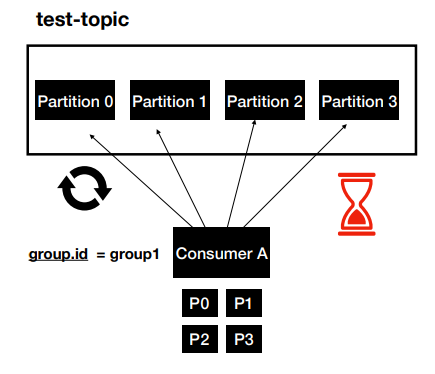
• Kafka Broker acts as a **Group Co-ordinator**

* Consumer Groups are used for scalable message consumption
* Each different application will have a unique consumer group

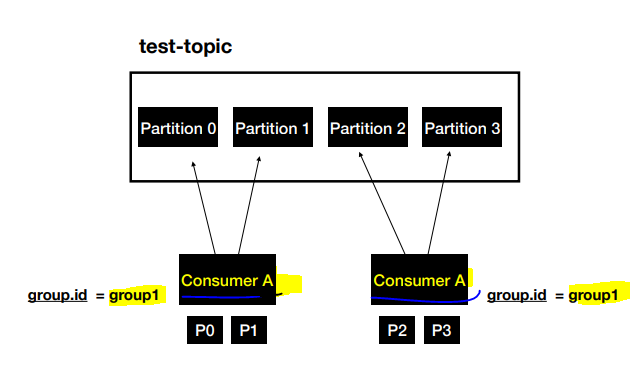
**Example 1:**

One consumer (**ConsumerA**) with group id **group1** consuming 4 partitions.

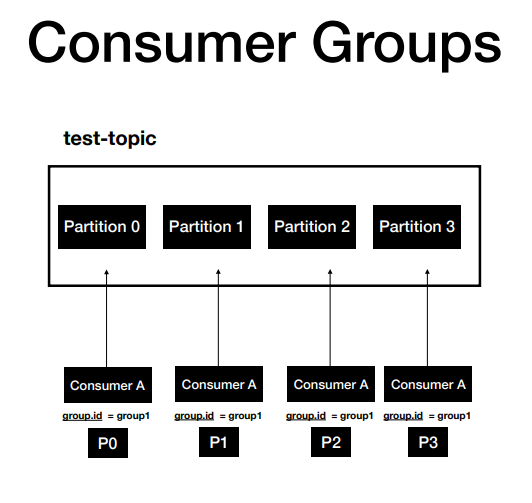
Performance will be poor.



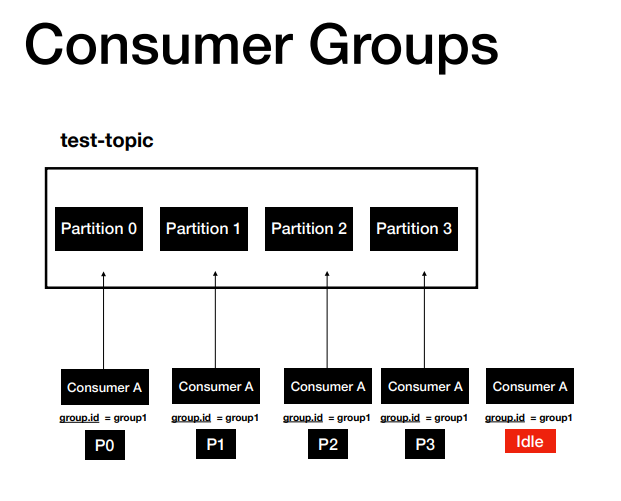
**Example 2:** One consumer (**ConsumerA**) with group id **group1** with 2 instances. Performance will be better than example-1.



Example -3: This will have best performance, **4 instances** of one consumer created for 4 partitions

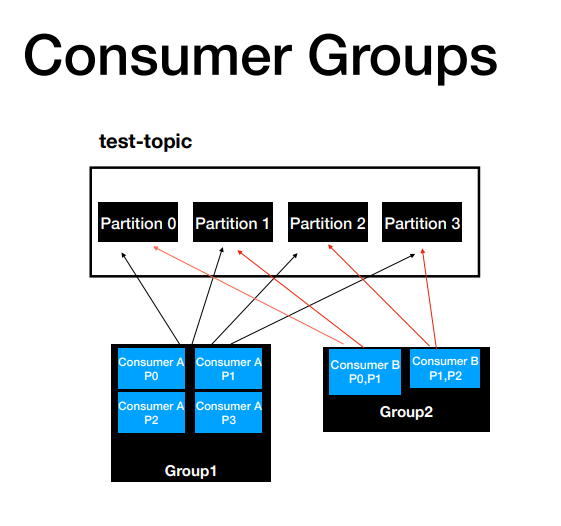


**Example-4:** If we have more instances of consumer then the partitions, other instances of the consumer will **idle.**



Example-5: Two consumer consuming same topic with their own instances.

Here, performance of ConsumerA will be better than ConsumerB.

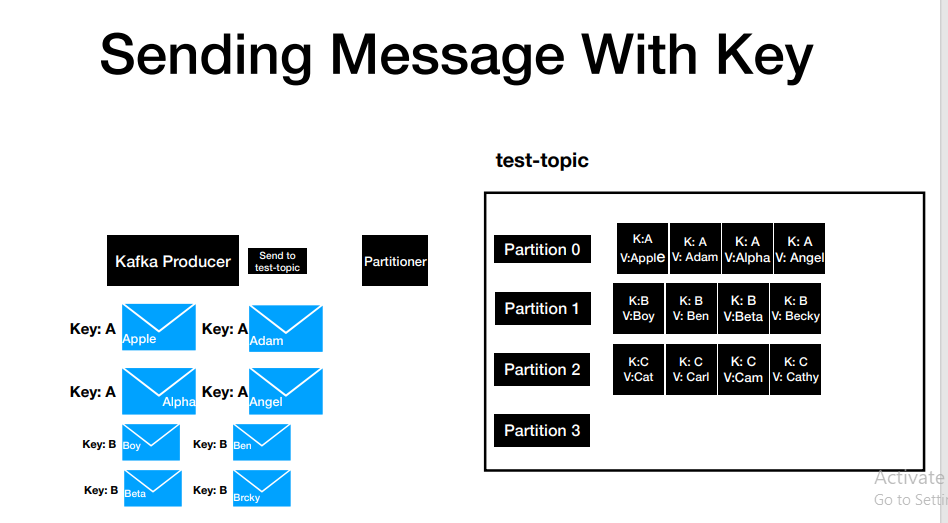


## \_\_Consumer\_Offsets:

It is a topic which store offset for each consumer.

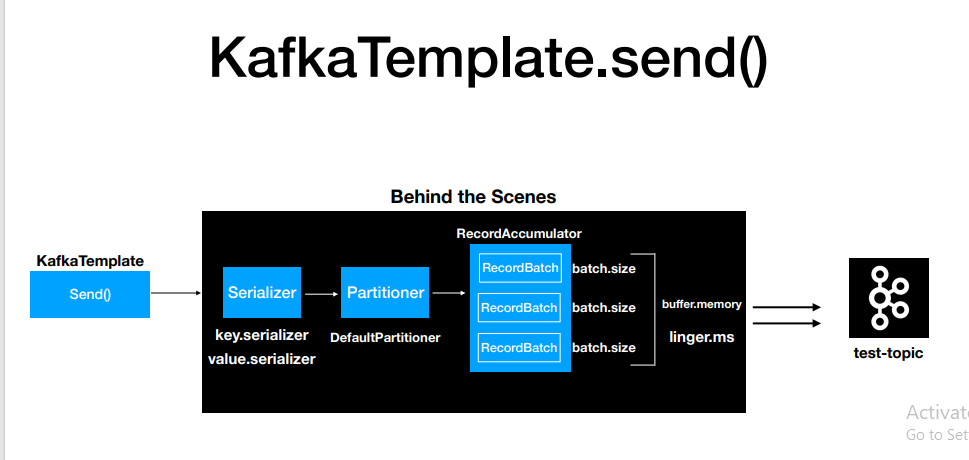
So, if the consumer crashed and after certain time, It return back, In order to know from where It should start retrieving the data It take offset value from \_\_ Consumer \_offset topic.

## Sending Message to a specific partition

We will send messages in key, value pair

## KafkaTemplate.send() Internal working principle:

* The messages pass through Serializer, Partitioner and Record Accumulator.
* **Serializer:**  This is use to encrypt key and value, Kafka does provide its own serializer but we can implement our own as well.
* **Partitioner:** This layer determines which partition the message is going to go in the partitions.
* **Record Accumulator:** The messages don’t send directly to partitions, It goes to Record Accumulator where we have record batch. Each record batch represent one partitions and the messages can be flow to topic partition one two different conditions:
  + **If the batch is full based on batch. Size**
  + **Even if the batch is not full but waiting time crosses linger.ms**



## Configuring Kafka Template

* Mandatory Values:
* bootstrap-servers: localhost:9092,localhost:9093,localhost:9094
* key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
* value-serializer: org.apache.kafka.common.serialization.StringSerializer