# APACHE KAFKA

## Introduction :

Apache Kafka is A msg queue for sending the data from

one application to other application.(from producer to Consumer).

Apache Kafka is going to act as a msg Queue to exchange the data.

Why Msg queue System:

Problem: Collecting all the data is not easy as data is generated from the

different sources and in various formats(json, img, txt, vdos etc.).

Creating the different pipelining system, it is very difficult to exchange the data.

Solution : By Using the messaging system, we can store and exchange the data

from different sources to the different applications.

So the msg queue will act as a mediator to exchange the data.

## Terminologies definition :

Publishers : The application which are generating the data are called publishers.

Subscribers : The applications which are taking the data from the msg queue are called the Subscribers.

Apache Kafka WORKS ON publishers subscribers model.

Apache Kafka is a distributed streaming platform.

# APIs available in Kafka :

1. Producer API -> it is used to store the data to the Kafka

2. Consumer API -> it is used to take the data from the Kafka

3. Streams API -> it is used to create the pipeline to take the data from the streams of data

4. Connector API-> it is used to establish the connection witht the Kafka.

Serializer : Converting the Java object into the Json format.

De-Serializer : Converting the Json format into the Java object.

## KAFKA Practical :

1. Install kafka and Zookeeper

2. run zookeeper : zookeeper-server-start.bat C:\Users\Rohit\_G\Downloads\Kafka\config\zookeeper.properties

3. run kafka : kafka-server-start.bat C:\Users\Rohit\_G\Downloads\Kafka\config\server.properties

4. create a kafka topic : kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic my-demo-topic

\*\* --replication-factor 1\*\* - How many copies of the topic we want to create is defined by the replication factor(Once the consumer comsumes the topic then it will be deleted from the msg queue).

5. List Topics created : kafka-topics.bat --list --bootstrap-server localhost:9092

Produce msg on Topic :

C:\Users\Rohit\_G\Downloads\Kafka\bin\windows>kafka-console-producer.bat --bootstrap-server localhost:9092 --topic customer

Message : > Welcome to my first topic from Kafka Producer

Check message on consumer : C:\Users\Rohit\_G\Downloads\Kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic customer --from-beginning

Response message :

Welcome to my first topic from Kafka Producer

## BROKER Configuration:

1. zookeeper.connect -> this take zookeeper connection string(HostName with portNumber)

- necessary to form a cluster

2. delete.topic.enable -> by default deleting a topic is not allowed, so to enable it we will use this property(true/false).

3. auto.create.topics.enable -> If a producer starts send messages to a non-existing topic, then Kafka will create the topic automatically and accept the data.

this behaviour is suitable in dev environment, we can set it to false if we don't want to accept data in non-existing topic.

4. default.replication.factor (by default 1)->

bydefault partition of a topic is 1, if we want to modify it then we can change this property.

5. num.partitions(default value) ->

6. log.retention.ms -> if we sent any data to Kafka, then it is not retained by kafka forever.

Kafka gives us two options to configure the retentionperiod -

1. retantion by time(default 7 days) {In this case Kafka will clear all the messages older than 7 days.}.

If we want to extend this time,then we can do by this property.

log.retention.bytes ->

1. Retention by Size : If we specify this propetry as 1 gb then, once the partition size approaches the size 1 gb, then KAFKA will trigger the clean up activity.

## IMPLEMENTATION USING JAVA :

1. Create a simple Java class for Producer, like “SimpleProducer”.
2. Create a **Property** object and specify these three mandatory properties and pass **Properties** class object in it’s constructor:
3. bootstrap.server
4. key.serializer
5. value.serializer
6. Create a **ProducerRecord** object and pass data like: topicName, key, value in it’s constructor.
7. Call the send method and pass **Producer** object in it, it will send the record data to the **topicName** passed in the **ProducerRecord** object.
8. Close the **Producer** object.

Example :

*package com.ApaceKafkaTut.producer;*

*import java.util.Properties;*

*import org.apache.kafka.clients.producer.KafkaProducer;*

*import org.apache.kafka.clients.producer.Producer;*

*import org.apache.kafka.clients.producer.ProducerRecord;*

*public class SimpleProducer {*

*public static void main(String[] args) {*

*String topicName = "TestTopicXYZ" ;*

*String key = "key-1";*

*String value = "value-1";*

*Properties props = new Properties();*

*props.put("bootstrap.servers", "localhost:9092, localhost:9093");*

*props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");*

*props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");*

*Producer<String, String> producer = new KafkaProducer<>(props);*

*ProducerRecord<String, String> record = new ProducerRecord<String, String>(topicName, key, value);*

*producer.send(record);*

*producer.close();*

*System.out.println("Completed Simple Producer");*

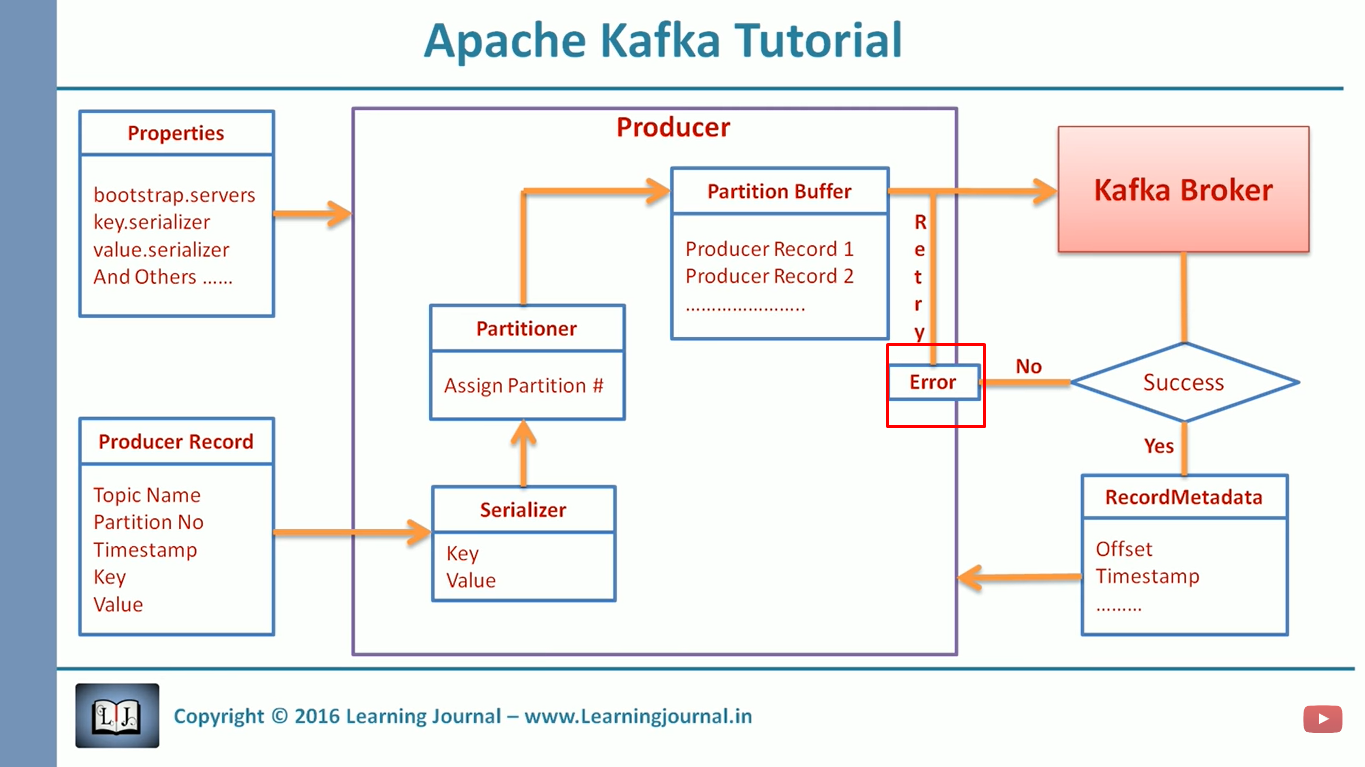
*}*

*}*

**NOTE IMPORTANT POINTS :**

1. ***The activity of converting Java object into an array of bytes is called Serialization.***
2. ***If the message key for different messages is same, then it will go to the same partition by hasing the message key.***
3. ***If the message key is not specified then the default partitioner will try to distribute the messages evenly to all the available partitions for the topic.***
4. ***Instead of sending the messages immediately, the Producer maintain the in-memory buffers for each partitions and sends the messages in batches.(we can configure this).***
5. ***If there is any error in sending messages to the broker, then the broker will retry sending it(configurable) in case if it is an recoverable error.***

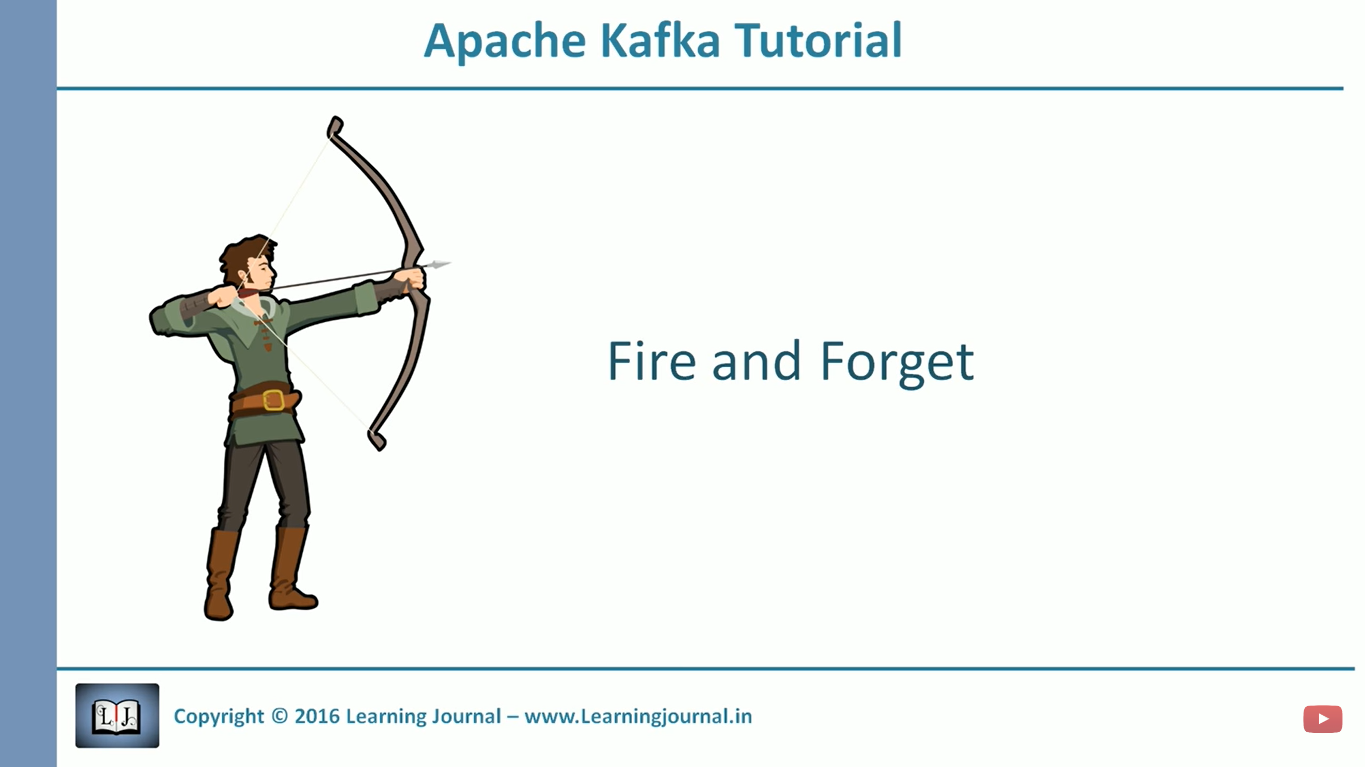
**KAFKA PRODUCER WORKFLOW :**



**Different approaches to implement a Kafka Producer :**

1. **Fire and forget approach :**

As we know that the Kafka is fault tolerant message processing system, that means the possibility of message loss is very loss in it. So. If our requirement can bear a small amount of data loss then we can use this approach.



1. **Synchronous approach :**

In this approach we send a message and wait until we get an acknowledgement.and in event of failure, we get an exception.

If we can not bear a small amount of data loss from our system, then we can follow this approach.

This approach may be slow than other approaches.