

Kafka: a Distributed Messaging System

to the next level

A Technical White Paper by DCP Team



**Abstract**

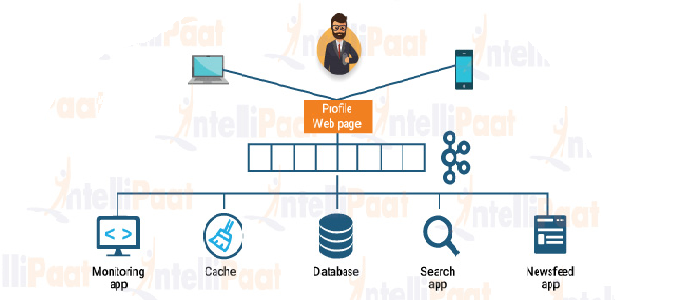
One of the biggest challenges that are associated with big data is, analyzing the data. But before we get to that part, the data has to be first collected and also for a system to process impeccably it should be able to grasp and make the data available to the users. This is where Apache Kafka comes in handy.

Apache Kafka has the ability to handle trillions of events occurring in a day. Kafka was initially developed for a message queue. A message queuing system helps in transferring the data rather than being focused on how data is being transferred and shared.

**Introduction**

Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.

Apache Kafka is a software where topics can be defined to where applications can add, Process and reprocess data(messages). Applications may connect to this system and transfer a message onto the topic. A message can include any kind of information. It could, for example, have information about an event that has happened on your website, or it could just be a simple text message that is supposed to trigger an even. Another application may connect to the system and process or re-process message from a topic. The data you send is stored in RAM or on the hard drive until by you specified retention period has passed by.



**What is Messaging System?**

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data and no need to worry about how it shares the data. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system.

Two types of messaging patterns are available :

1. point to point messaging system.
2. publish-subscribe (pub-sub) messaging system.

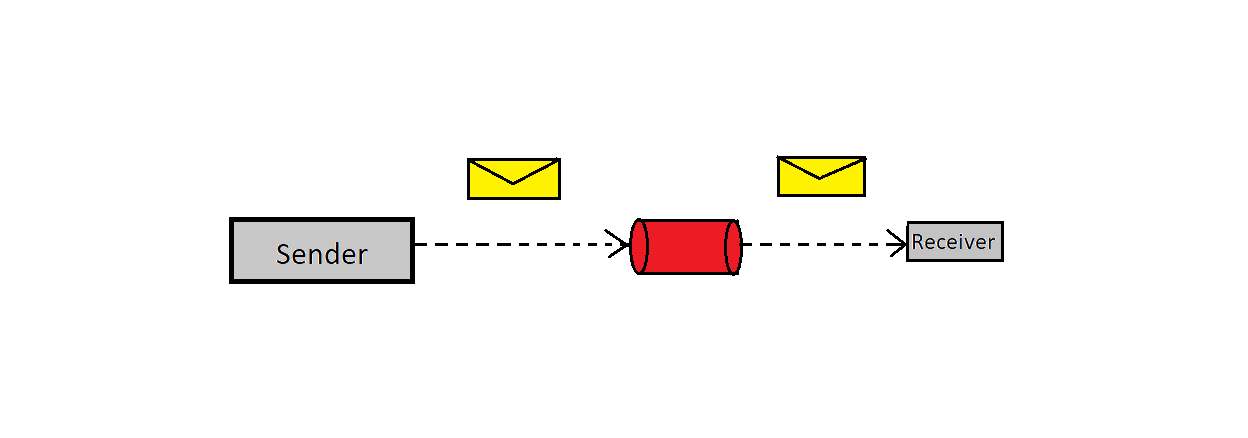
Most of the messaging patterns follow pub-sub.

### Point to Point Messaging System

In a point-point system, message are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue.

The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well processed by one Order processor, but Multiple Order Processors can work as well at the same time.

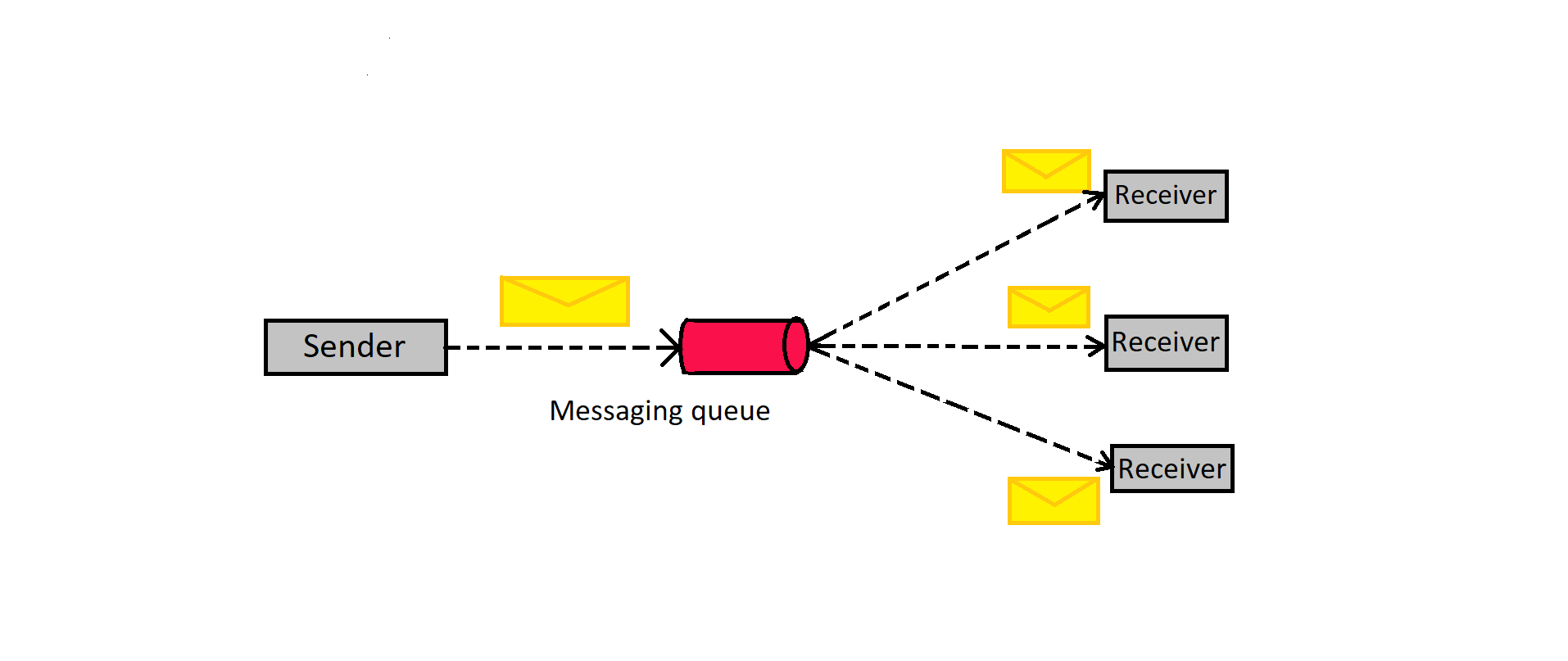
The following diagram depicts the structure.

****

**Publish-Subscribe Messaging System**

In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers.

A real-life example is Dish TV,which publishes different channels like sports , movies, musics, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.



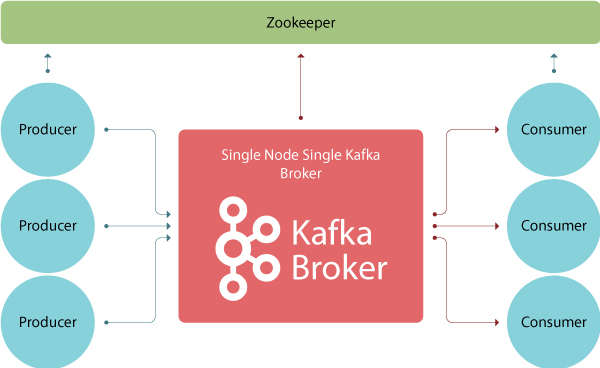
**What is Zookeeper?**

Zookeeper is used for managing and coordinating Kafka broker. Zookeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts co-ordinating their task with some other broker.



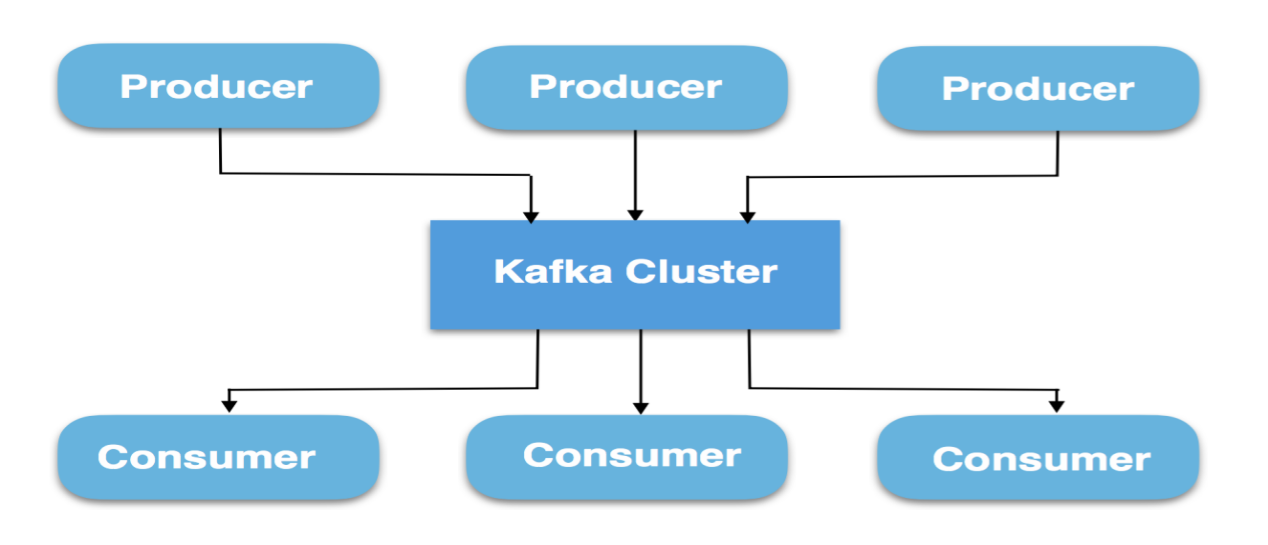
**What is Broker?**

Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use Zookeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact. Kafka broker leader election can be done by Zookeeper.



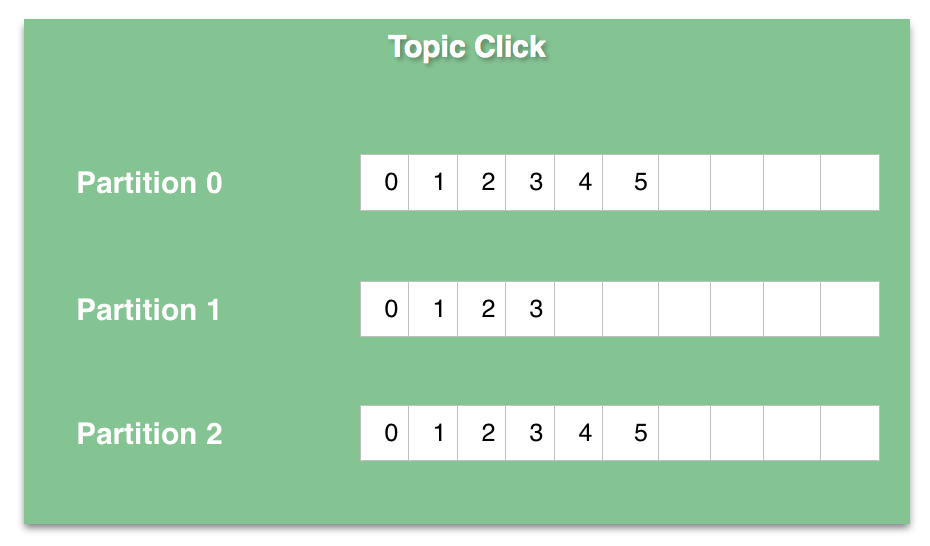
**What is Cluster?**

Kafka’s having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.



**What is Topic?**

A stream of messages belonging to a particular category is called a topic. Data is stored in topics.Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.



**What is Partition?**

Topics may have many partitions, so it can handle an arbitrary amount of data.

**What is Partition offset?**

Each partitioned message has a unique sequence id called as offset.

**What is Replica of Partition?**

Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss.

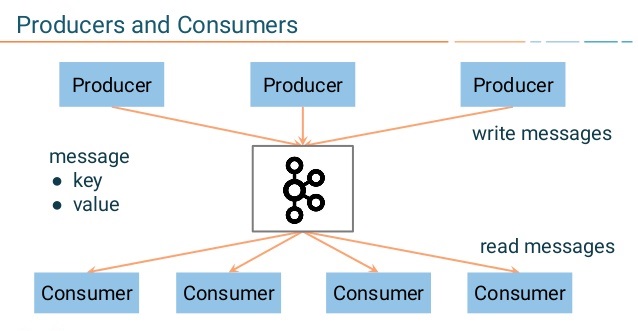
**What is Producer?**

Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.

Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn’t wait for acknowledgments from the broker and sends messages as fast as the broker can handle.

**What is Consumer?**

Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.



**Kafka Appender**

The Kafka log appender is responsible for transferring logs from the Operations server to the Apche kafka service. The logs are stored in the specified topic. Appenders are usually responsible for writing the event data to the target destination. Some appenders wrap other appenders so that they can modify the LogEvent, handle a failure in an appender, route the event to a subordinate appender based on advanced filter criteria or provide similar functionality that does not directly format the event for viewing.

<Appenders>

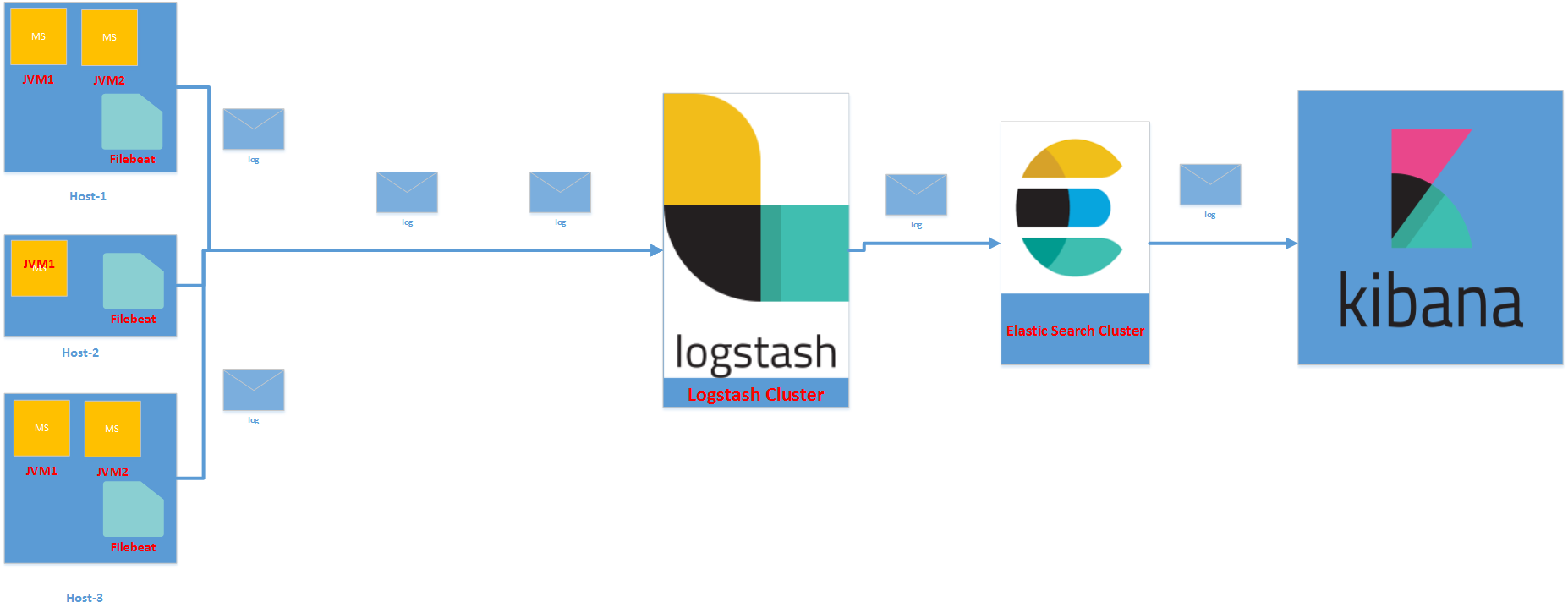
<Kafka name="Kafka" topic="log-test">

<PatternLayout pattern="%date %message"/>

<Property name="bootstrap.servers">localhost:9092</Property>

</Kafka>

</Appenders>

****

**Step to run Kafka in local system:**

**Step:1 Install Java SE Server JRE**

1. Download Java SE Server JRE from

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

1. Unpack it to a folder, for example C:\JREServer
2. Update the system enviroment variable PATH to include C:\JREServer\jre\bin

**Step:2 Download Kafka**

1. Download the binaries from <https://kafka.apache.org/downloads.html>
2. Unpack it to a folder, for example C:\kafka

**Step:3 Update file and configuration**

Update kafka configuration files, for windows update bat files

1. Open config/server.properties, change

Log.dirs=/tmp/kafka

to

Log.dirs=C:/kafka/kafka-logs

2. Open config/zookeeper.properties, change

dataDir=/tmp/zookeeper

to  
 dataDir=C:/kafka/zookeeper-data

**Step:4** **Start the Zookeeper Server in windows CMD**

In Windows Command Prompt, switch the current working directory to C:\kafka:

Kafka uses ZooKeeper so you need to first start a ZooKeeper server if you don’t already have one. You can use the convenience script packaged with Kafka to get a quick-and-dirty single-node ZooKeeper instance.

Command to run zookeeper server:.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

**Step:5 Start Kafka in another Command Prompt or Powershell Console**

Command to run kafka:.\bin\windows\kafka-server-start.bat .\config\server.properties

**Step:6 Create a topic**

1. Create a topic

Command to create topic:.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

1. List topics

Command to show topics:.\bin\windows\kafka-topics.bat --list --zookeeper localhost:2181

**Step:7 Start Console Producer**

Command to start producer:.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test

Send some messages like,

This is a first kafka topic.

It conatins kafka appender logstash.

**Step:8 Start Console Consumer**

Command to run Consumer:.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning

**Step to run kafka spring boot application in AWS**

**Step-1 Create AWS instance**

To create an AWS instance, please refer official AWS documentation

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2\_GetStarted.html#ec2-launch-instance\_linux.

To run kafka application create three EC2 instance, 1st is for kafkabroker, 2nd is for kafkaconsumer and 3rd is for microservices. Use name kafkabroker for 1st instance, kafkaconsumer for 2nd instance and application for 3rd instance, to avoid confusion, then download key and convert that key into .ppm format using puttyGen.

**Step-2 Install java in EC2 instance**

Since we will be working with the kafka binary for scala 2.12, Our instance must have java 8. To install java 8 in both instance please follow below steps:-

Step-(i) Open puttyGen and load .pem file key, then save that private key with some name in .ppk format.

Step-(ii) Open putty copy public IP address in hostname, from AWS EC2 instance dashboard and browse that .ppk key and click on open.

step-(iii) In AWS cli give user as ec2-user. Check pwd in linux virtual operating system then download java 8 using below cli command:

wget --no-cookies--no-check-certificate--header"cookie:gpw\_e24=http%3a%2f%2f www.oracle.com%2f;oraclelicense=accept-securbackup-cookie""https://download.oracle.com/otn-pub/java/jdk/8u201-b09/42970487e3af4f5aa5bca3f542482c60/jdk-8u201-linux-x64.tar.g"

step-(iv) Unzip this folder using command: tar zxvf jdk filename.

step-(v) Use ls -ai command to see all type of .bashrc file, then open .bashrc file using vi .bashrc command and insert(to set java enviroment variable) below :-

export JAVA\_HOME="/home/ec2-user/jdk1.8.0\_201"

export PATH=$PATH:$JAVA\_HOME/bin

then save this .bashrc file and exit.

step-(vi) Check java version using command: java -version,

if you get java version it means you have installed java successfully in linux machine.

But if it gives error then check with the above steps.

In order to install java in other instance repeat the above steps for the same.

**Step-3 Downloading and Extracting kafka**

Use one instance for kafkbroker and other for kafkaconsumer, note down public and private IP address of both instance and paste in notepad. Now open AWS Cli using kafkabroker public ip address through putty then copy the below command into cli:-

Download kafka in both instances using command

wget http://mirrors.estointernet.in/apache/kafka/2.2.0/kafka\_2.12-2.2.0.tgz

Command for extracting the .tgz file

tar zxvf kafka filename

**Step-4 Starting Zookeeper**

Since Kafka uses zookeeper, we need to first start a zookeeper server. Since we have 1GB RAM we would be setting KAFKA\_HEAP\_OPTS enviroment variable in our .bashrc to 50% of total RAM i.e 512 MB. To run zookeeper server open kafkabroker instance in cli using putty and follow step:-

start zookeeper by following command:-

cd kafka filename

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

**Step-5 Starting Kafka Instance**

To run kafka open kafkabroker instance in AWS cli using putty then copy below command:

cd kafka filename

export KAFKA\_HEAP\_OPTS="-Xmx512M -Xmx512M"

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

before running kafka command first edit server.properties by using below command:-

goto kafka directory then type ls -ai, and use command vi server.properties

edit your host name into your current private kafkabroker ip address and then save it and exit. After completion of this step run kafka command.

through steps-4 & 5 zookeeper and kafka will start running in background.

**Step-6 Create topic**

Open putty and use kafkabroker instance properties to open AWS Cli, then use users as ec2-user and open it.

Use below command to create topic:-

bin/kafka-topics.sh --create --zookeeper your kafkabroker private ip:2181 --replication-factor 1 --partitions 1 --topic topic1

command to list the topic:-

bin/kafka-topics.sh --list --zookeeper :2181

**Step-7 Deploy Spring-boot application**

Download kafka spring-boot aplication from github link:<https://github.com/Ashu827/kafka-spring-boot>

To deploy kafka application install winSCP and use previous .ppm format key. ec2-user as user name and public IP address of application instance then connect winSCP to EC2 instance. Copy .jar file spring-boot application into winSCP. After copying of .jar file we can see our file in application instance AWS Cli. Now, to run this application run below command:-

Java -jar .jar filename

after this command application will start running.

**Step-8 Start kafka consumer console**

To start consumer cli, use kafkaconsumer instance in putty and use previous key in .ppk format and run below command:-

bin/kafka-console-consumer.sh --bootstrap-server kafkabroker private ip:9092 --topic test1 --from-beginning