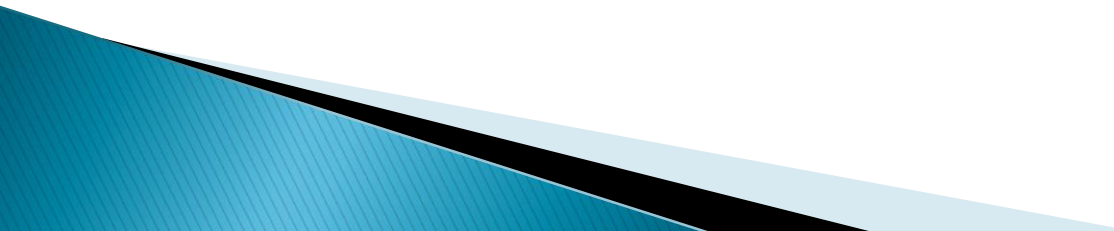
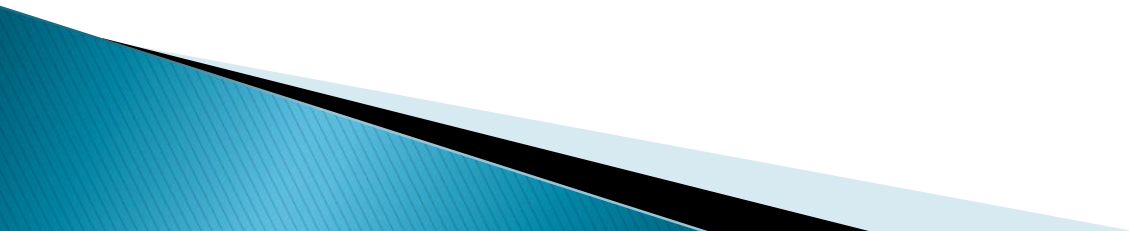


APACHE KAFKA ESSENTIALS

Rajesh Pasham

- ▶ Apache Kafka was originated at LinkedIn and later became an open sourced Apache project in 2011, then First-class Apache project in 2012.
 - ▶ Kafka is written in Scala and Java.
 - ▶ Apache Kafka is publish-subscribe based fault tolerant messaging system.
 - ▶ It is fast, scalable and distributed by design.
- 

- ▶ In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.



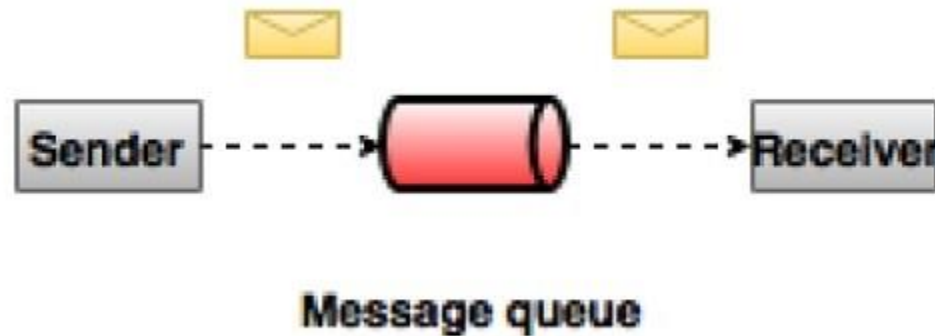
What is a Messaging System?

- ▶ A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it.
- ▶ 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 – one is point to point and the other is publish-subscribe (pub-sub) messaging system.


Point to Point Messaging System

- ▶ In a point-to-point system, messages 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 at the same time.

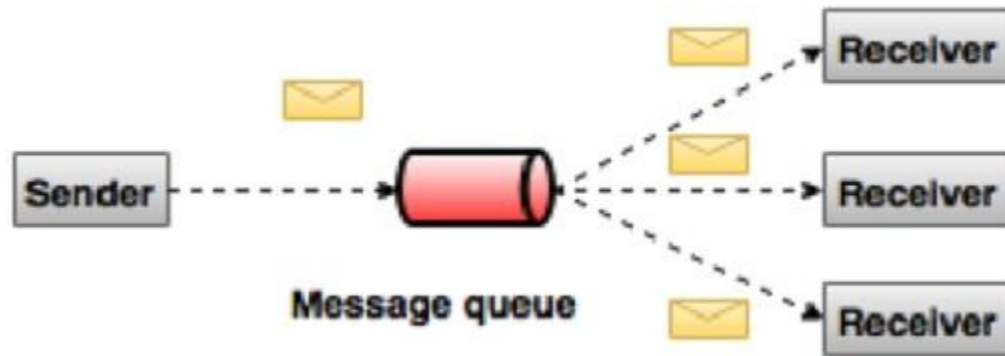
Point to Point Messaging System



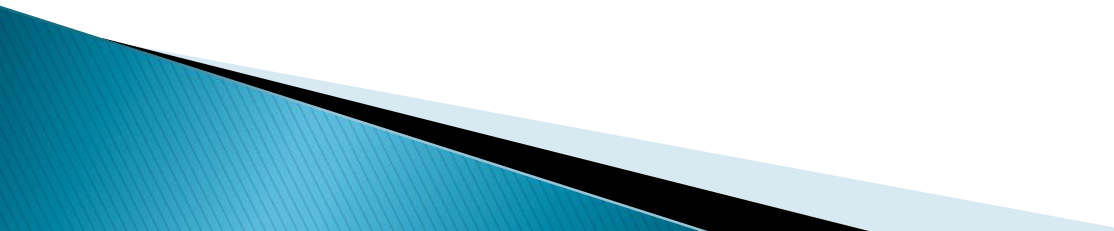
Publish-Subscribe Messaging System

- ▶ 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, music, etc.
- 

Publish-Subscribe Messaging System

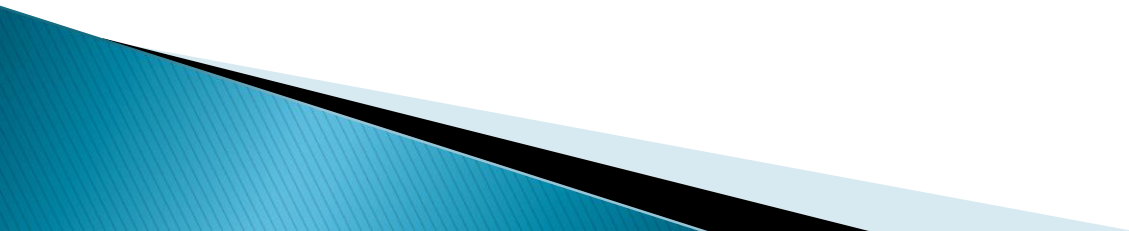


What is Kafka?

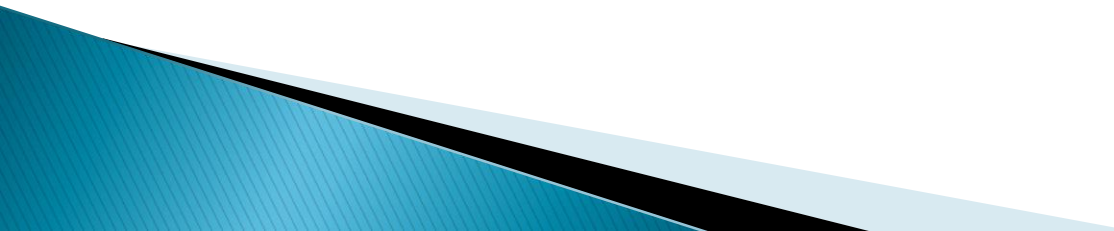
- ▶ Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another.
 - ▶ Kafka is suitable for both offline and online message consumption.
 - ▶ Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.
- 

What is Kafka?

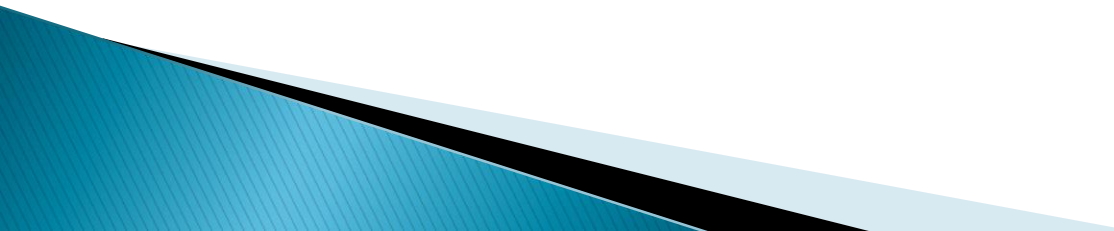
- ▶ Kafka is built on top of the ZooKeeper synchronization service.
- ▶ It integrates very well with Apache Storm and Spark for real-time streaming data analysis.



Benefits


- ▶ **Reliability** – Kafka is distributed, partitioned, replicated and fault tolerance.
 - ▶ **Scalability** – Kafka messaging system scales easily without down time.
 - ▶ **Durability** – Kafka uses Distributed commit log which means messages persists on disk as fast as possible, hence it is durable.
 - ▶ **Performance** – Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.
- 

Use Cases

- ▶ **Metrics** – Kafka is often used for operational monitoring data.
 - ▶ **Log Aggregation Solution** – Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.
 - ▶ **Stream Processing** – Popular frameworks such as Storm and Spark Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications.
- 

Use Cases

Walmart's Real-Time Inventory System Powered by Apache Kafka


- ▶ Consumer shopping patterns have changed drastically in the last few years.
 - ▶ Shopping in a physical store is no longer the only way.
 - ▶ Retail shopping experiences have evolved to include multiple channels, both online and offline, and have added to a unique set of challenges in this digital era.
 - ▶ Having an up to date snapshot of inventory position on every item is a very important aspect to deal with these challenges.
 - ▶ Walmart have solved this at scale by designing an event-streaming-based, real-time inventory system leveraging Apache Kafka®.
- 

Use Cases

Walmart's Real-Time Inventory System Powered by Apache Kafka

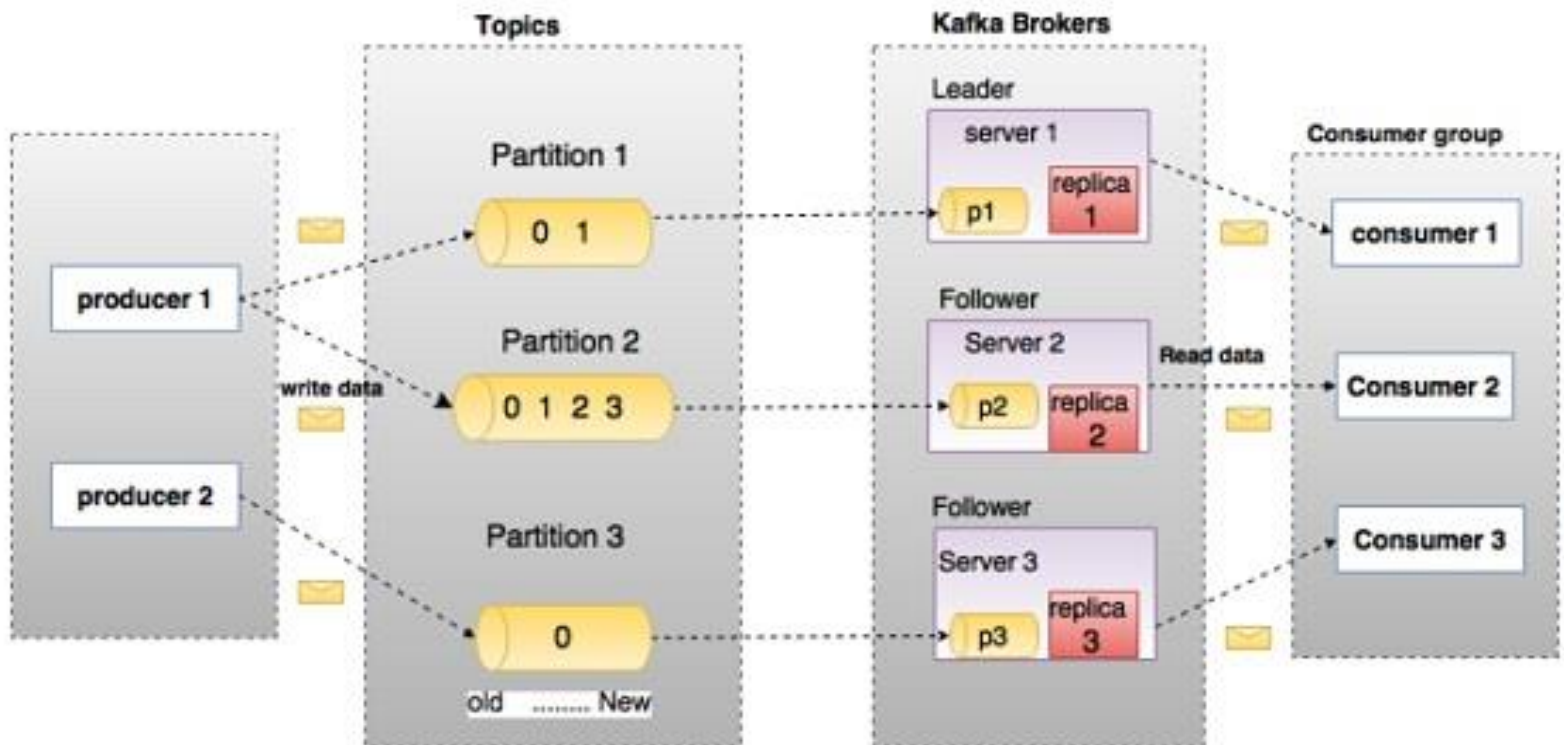
- ▶ Furthermore, like any supply chain network, Walmart's infrastructure involved a plethora of event sources with all different types of data contributing to net change to inventory positions, so they leveraged Kafka Streams to house the data and a Kafka connector to take the data and ingest it into Apache Cassandra and other data stores.

Need for Kafka

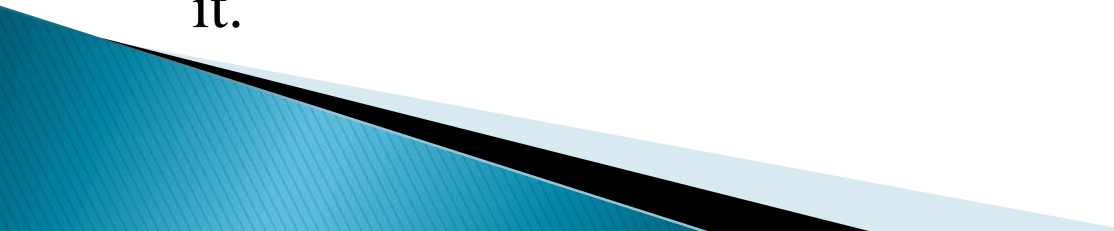
- ▶ Kafka is a unified platform for handling all the real-time data feeds.
 - ▶ Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures.
 - ▶ It has the ability to handle a large number of diverse consumers.
 - ▶ Kafka is very fast, performs 2 million writes/sec.
- 

Apache Kafka - Fundamentals

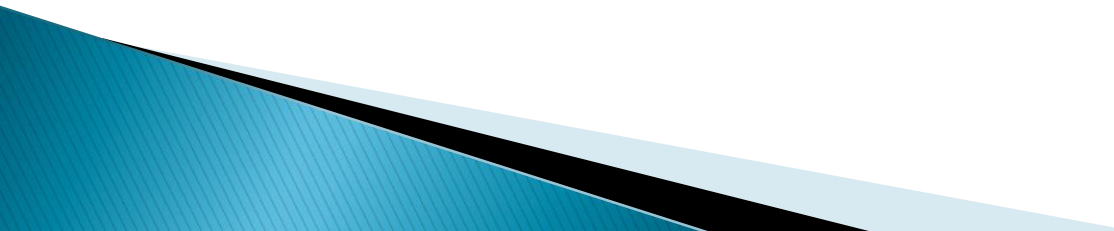
- ▶ The following diagram illustrates the main terminologies



Apache Kafka - Fundamentals


- ▶ In the above diagram, a topic is configured into three partitions.
 - ▶ Partition 1 has two offset factors 0 and 1.
 - ▶ Partition 2 has four offset factors 0, 1, 2, and 3.
 - ▶ Partition 3 has one offset factor 0.
 - ▶ The id of the replica is same as the id of the server that hosts it.
- 

Apache Kafka - Fundamentals

- ▶ Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations.
 - ▶ To balance a load in cluster, each broker stores one or more of those partitions.
 - ▶ Multiple producers and consumers can publish and retrieve messages at the same time.
- 

Apache Kafka - Fundamentals

Topics

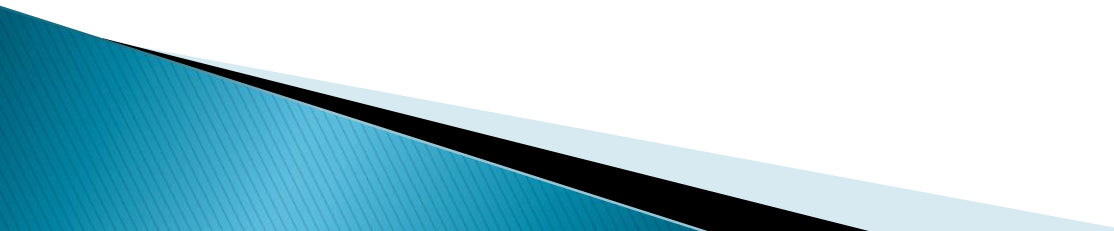
- ▶ A stream of messages belonging to a particular category is called a topic. Data is stored in topics. Topics are split into partitions.
 - ▶ For each topic, Kafka keeps a minimum of one partition. Each such partition contains messages in an immutable ordered sequence.
 - ▶ A partition is implemented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

Apache Kafka - Fundamentals

Partition

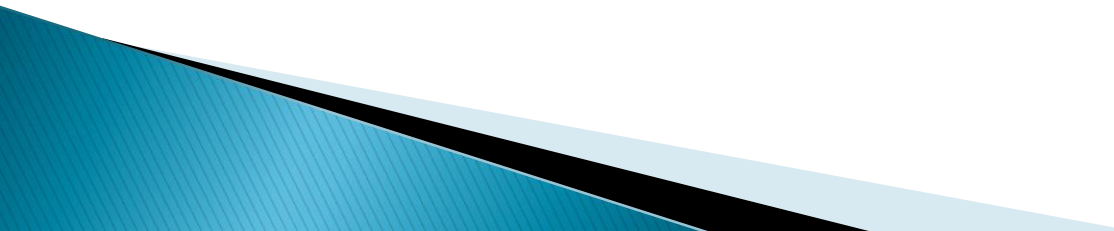
- ▶ Topics may have many partitions, so it can handle an arbitrary amount of data.

Partition offset

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

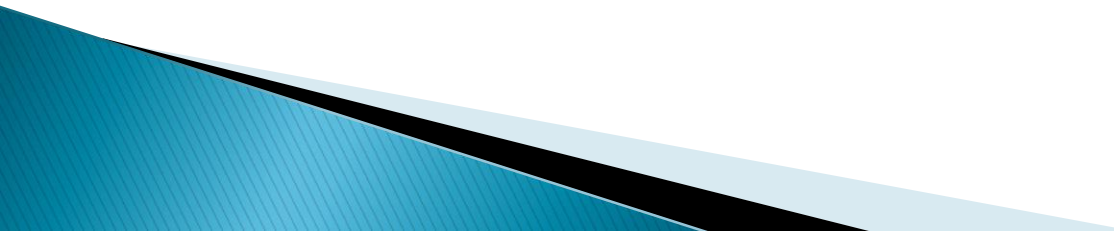
Apache Kafka - Fundamentals

Replicas of partition

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


Apache Kafka - Fundamentals

Brokers

- ▶ Brokers are simple system responsible for maintaining the published data.
 - ▶ Each broker may have zero or more partitions per topic.
 - ▶ If there are N partitions in a topic and N number of brokers, each broker will have one partition.
- 

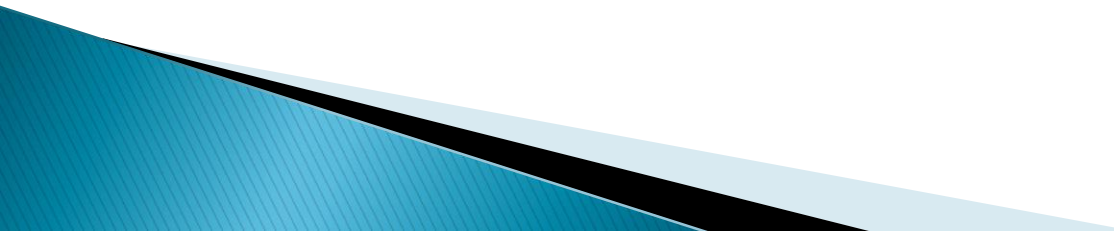
Apache Kafka - Fundamentals

Brokers

- ▶ If there are N partitions in a topic and more than N brokers ($n + m$), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.
 - ▶ If there are N partitions in a topic and less than N brokers ($n - m$), each broker will have one or more partition sharing among them.
 - ▶ This scenario is not recommended due to unequal load distribution among the broker.
- 


Apache Kafka - Fundamentals

Kafka 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.
- 

Apache Kafka - Fundamentals

Producers

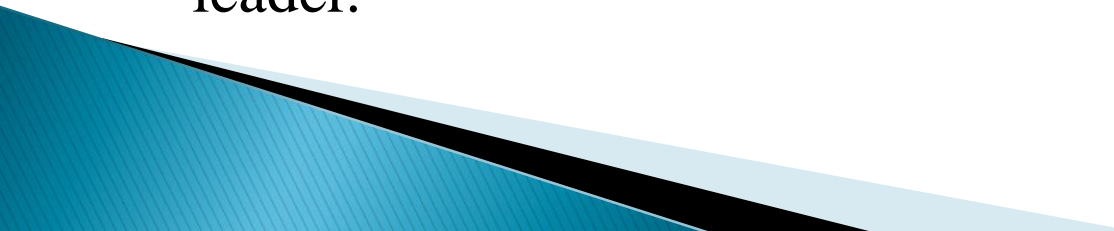
- ▶ Producers are the publisher of messages to one or more Kafka topics.
 - ▶ Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition.
 - ▶ Producer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

Apache Kafka - Fundamentals

Consumers

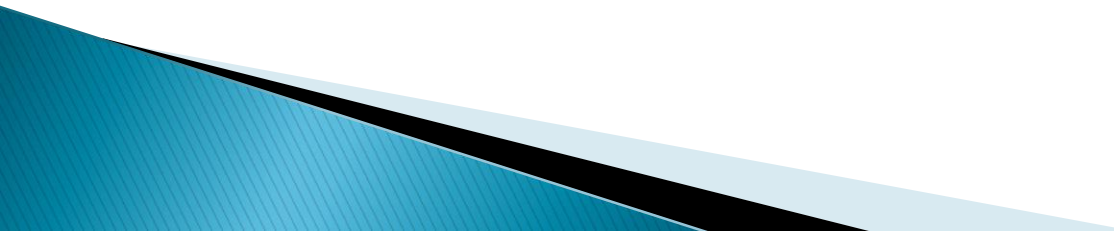
- ▶ Consumers read data from brokers.
- ▶ Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.

Leader

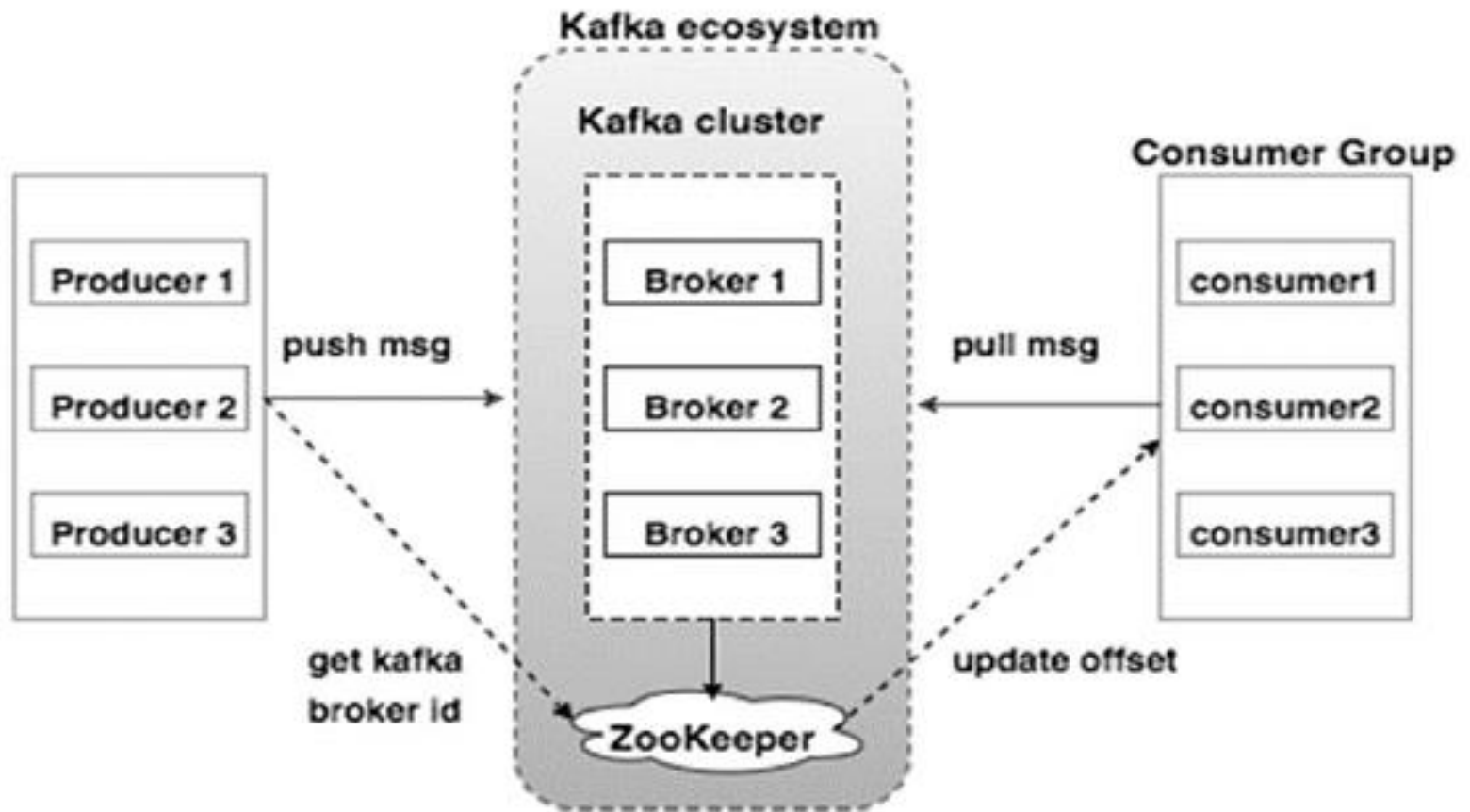
- ▶ Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.
- 

Apache Kafka - Fundamentals

Follower


- ▶ Node which follows leader instructions are called as follower.
 - ▶ If the leader fails, one of the follower will automatically become the new leader.
 - ▶ A follower acts as normal consumer, pulls messages and updates its own data store.
- 

Apache Kafka - Cluster Architecture



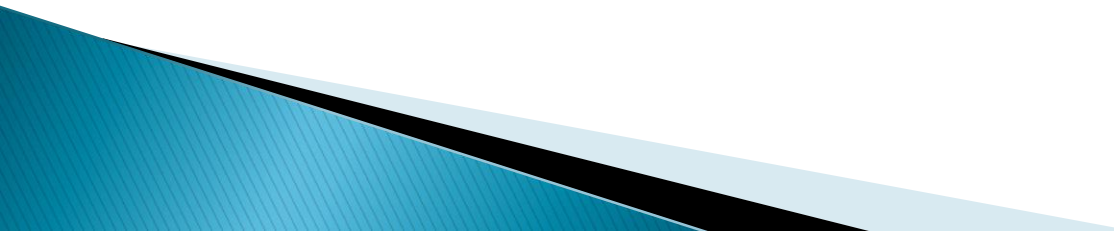
Apache Kafka - Cluster Architecture

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.
- 

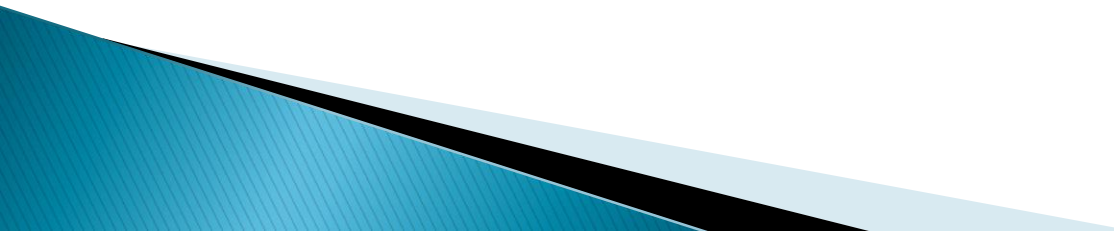
Apache Kafka - Cluster Architecture

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 coordinating their task with some other broker.
- 


Apache Kafka - Cluster Architecture

Producers

- ▶ 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 acknowledgements from the broker and sends messages as fast as the broker can handle.
- 

Apache Kafka - Cluster Architecture

Consumers

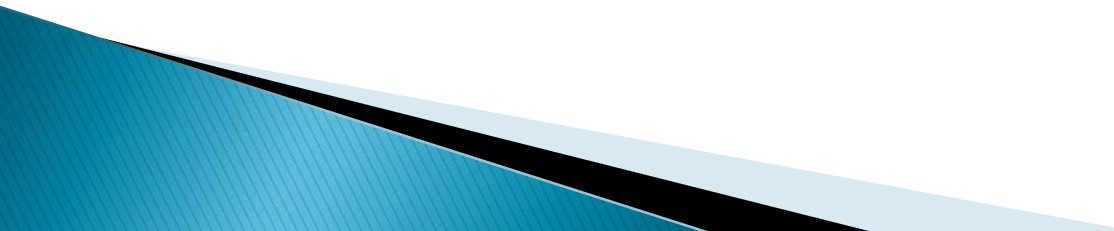
- ▶ Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset.
 - ▶ If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages.
 - ▶ The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume.
- 

Apache Kafka - Cluster Architecture

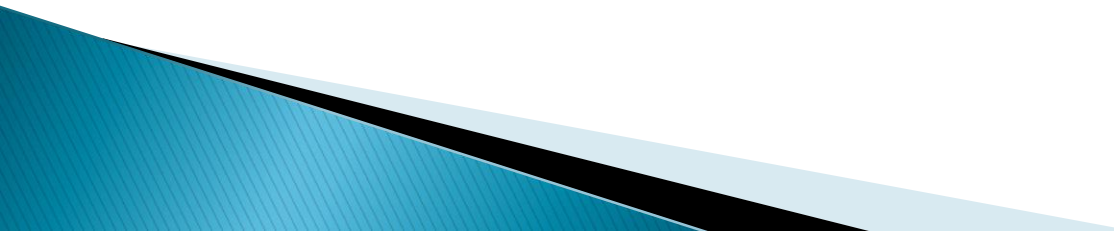
Consumers

- ▶ 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.

Apache Kafka - Workflow

- ▶ Kafka is simply a collection of topics split into one or more partitions.
 - ▶ A Kafka partition is a linearly ordered sequence of messages, where each message is identified by their index (called as offset).
 - ▶ All the data in a Kafka cluster is the disjointed union of partitions.
- 

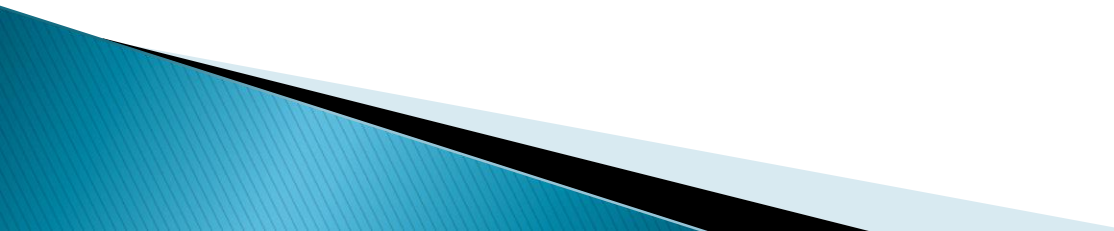
Apache Kafka - Workflow

- ▶ Incoming messages are written at the end of a partition and messages are sequentially read by consumers.
 - ▶ Durability is provided by replicating messages to different brokers.
 - ▶ Kafka provides both pub-sub and queue based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner.
- 


Apache Kafka - Workflow

- ▶ producers simply send the message to a topic and consumer can choose any one type of messaging system depending on their need.

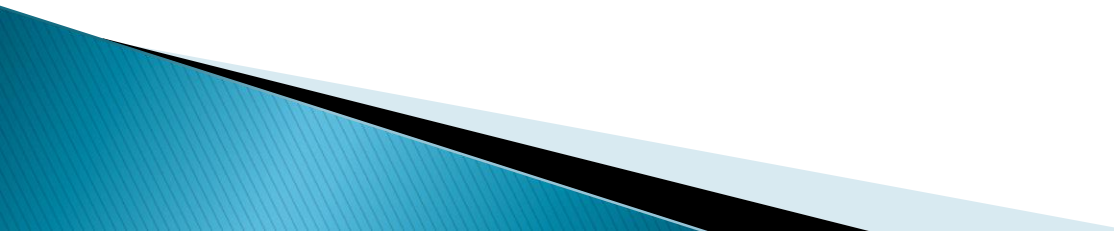
Workflow of Pub-Sub Messaging

- ▶ Producers send message to a topic at regular intervals.
 - ▶ Kafka broker stores all messages in the partitions configured for that particular topic.
 - ▶ It ensures the messages are equally shared between partitions.
 - ▶ If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
- 

Workflow of Pub-Sub Messaging

- ▶ Consumer subscribes to a specific topic.
 - ▶ Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
 - ▶ Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.
 - ▶ Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- 

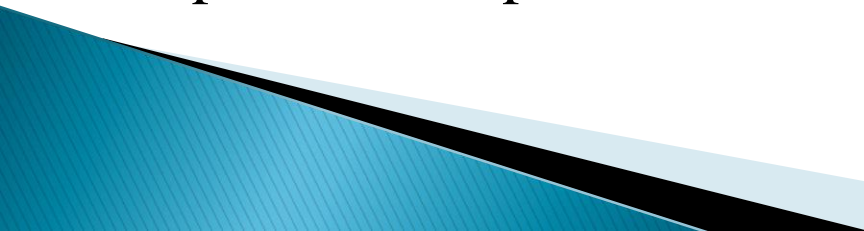
Workflow of Pub-Sub Messaging

- ▶ Consumer will receive the message and process it.
 - ▶ Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
 - ▶ Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper.
 - ▶ Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outages.
- 

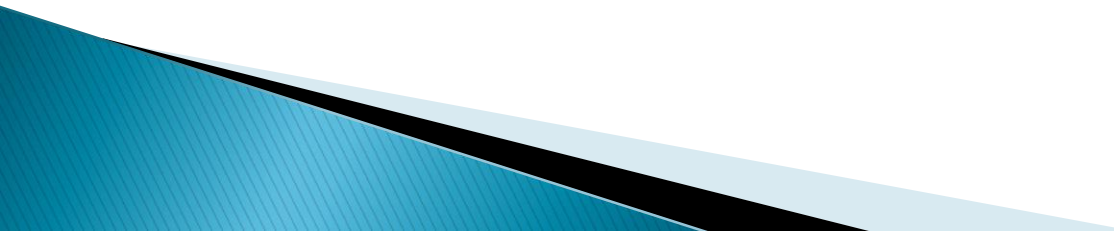
Workflow of Pub-Sub Messaging

- ▶ This above flow will repeat until the consumer stops the request.
- ▶ Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

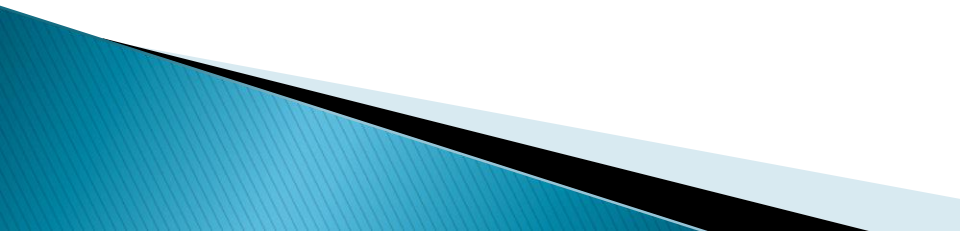
Workflow of Queue Messaging / Consumer Group

- ▶ In a queue messaging system instead of a single consumer, a group of consumers having the same Group ID will subscribe to a topic.
 - ▶ In simple terms, consumers subscribing to a topic with same Group ID are considered as a single group and the messages are shared among them.
 - ▶ Producers send message to a topic in a regular interval.
 - ▶ Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
- 

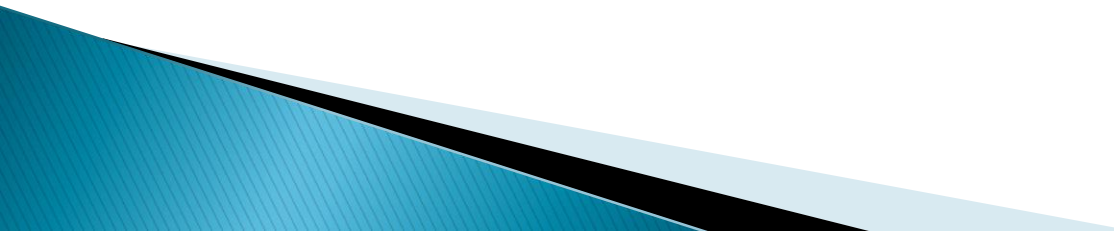
Workflow of Queue Messaging / Consumer Group

- ▶ A single consumer subscribes to a specific topic, assume Topic-01 with Group ID as Group-1.
 - ▶ Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, Topic-01 with the same Group ID as Group-1.
 - ▶ Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers.
- 


Workflow of Queue Messaging / Consumer Group

- ▶ This sharing will go on until the number of consumers reach the number of partition configured for that particular topic.
 - ▶ Once the number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribes.
 - ▶ This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.
- 

Role of ZooKeeper

- ▶ A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service.
 - ▶ Zookeeper serves as the coordination interface between the Kafka brokers and consumers.
 - ▶ The Kafka servers share information via a Zookeeper cluster.
 - ▶ Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.
- 

Role of ZooKeeper

- ▶ Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster.
 - ▶ Kafka will restore the state, once the Zookeeper restarts.
 - ▶ This gives zero downtime for Kafka.
 - ▶ The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.
- 

Apache Kafka - Installation Steps

▶ Step 1 - Verifying Java Installation

```
$ java -version
```

- If java is successfully installed on your machine, you could see the version of the installed Java.

▶ Step 2 - ZooKeeper Framework Installation

- Step 2.1 - Download ZooKeeper
 - As of now, latest version of ZooKeeper is 3.4.6 (ZooKeeper-3.4.6.tar.gz).

Apache Kafka - Installation Steps

- Step 2.2 - Extract tar file

```
$tar -zxf zookeeper-3.4.6.tar.gz
```

- Step 2.3 - Create Configuration File

- Open Configuration File named conf/zoo.cfg using the command vi “conf/zoo.cfg” and all the following parameters to set as starting point.

```
$ vi conf/zoo.cfg
```

```
tickTime=2000
```

```
dataDir=/path/to/zookeeper/data
```

```
clientPort=2181
```

```
initLimit=5
```

```
syncLimit=2
```

Apache Kafka - Installation Steps

- Step 2.4 - Start ZooKeeper Server

```
$ bin/zkServer.sh start
```

After executing this command, you will get a response as shown below –

```
$ JMX enabled by default
```

```
$ Using config: /Users/..../zookeeper-3.4.6/bin/..../conf/zoo.cfg
```

```
$ Starting zookeeper ... STARTED
```



Apache Kafka - Installation Steps

- Step 2.5 - Start CLI

```
$ bin/zkCli.sh
```

After typing the above command, you will be connected to the zookeeper server and will get the below response.

```
Connecting to localhost:2181
```

```
.....
```

```
.....
```

```
.....
```

```
Welcome to ZooKeeper!
```

```
.....
```

```
.....
```

```
WATCHER::
```

```
WatchedEvent state:SyncConnected type: None path:null
```

```
[zk: localhost:2181(CONNECTED) 0]
```

Apache Kafka - Installation Steps

- Step 2.6 - Stop Zookeeper Server
- After connecting the server and performing all the operations, you can stop the zookeeper server with the following command –

```
$ bin/zkServer.sh stop
```

- Now you have successfully installed Java and ZooKeeper on your machine. Let us see the steps to install Apache Kafka.

Apache Kafka - Installation Steps

- ▶ Step 3 - Apache Kafka Installation
 - Step 3.1 - Download Kafka
 - the latest version i.e., – **kafka_2.11_0.9.0.0.tgz**
 - Step 3.2 - Extract the tar file

```
$ tar -zxf kafka_2.11.0.9.0.0 tar.gz
```

```
$ cd kafka_2.11.0.9.0.0
```
 - Step 3.3 - Start Server

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

Apache Kafka - Basic Operations

- ▶ First let us start implementing single node-single broker configuration and we will then migrate our setup to single node-multiple brokers configuration.
- ▶ Before moving to the Kafka Cluster Setup, first you would need to start your ZooKeeper because Kafka Cluster uses ZooKeeper.
- ▶ Start ZooKeeper
`$bin/zookeeper-server-start.sh config/zookeeper.properties`

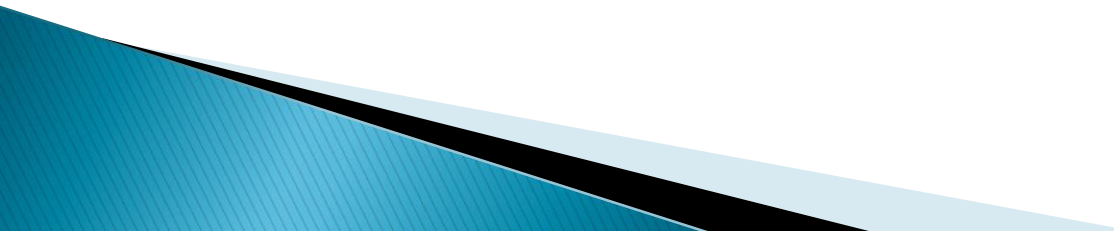
Apache Kafka - Basic Operations

- ▶ To start Kafka Broker, type the following command –
`$bin/kafka-server-start.sh config/server.properties`
- ▶ After starting Kafka Broker, type the command `jps` on ZooKeeper terminal and you would see the following response –

821 QuorumPeerMain
928 Kafka
931 Jps
- ▶ Now you could see two daemons running on the terminal where QuorumPeerMain is ZooKeeper daemon and another one is Kafka daemon.

Apache Kafka - Basic Operations

Single Node-Single Broker Configuration

- ▶ In this configuration you have a single ZooKeeper and broker id instance.
 - ▶ Following are the steps to configure it –
 - ▶ Creating a Kafka Topic – Kafka provides a command line utility named `kafka-topics.sh` to create topics on the server. Open new terminal and type the below example.
- 

Apache Kafka - Basic Operations

Single Node-Single Broker Configuration

- ▶ Syntax

```
$bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --  
replication-factor 1 --partitions 1 --topic topic-name
```

- ▶ Example

```
$bin/kafka-topics.sh --create --bootstrap-server localhost:9092  
--replication-factor 1 --partitions 1 --topic Hello-Kafka
```

- ▶ We just created a topic named Hello-Kafka with a single partition and one replica factor.

Apache Kafka - Basic Operations

Single Node-Single Broker Configuration

- ▶ The above created output will be similar to the following output –

Output – Created topic Hello-Kafka

- ▶ Once the topic has been created, you can get the notification in Kafka broker terminal window and the log for the created topic specified in “/tmp/kafka-logs/“ in the config/server.properties file.

Apache Kafka - Basic Operations

List of Topics

- ▶ To get a list of topics in Kafka server, you can use the following command –
Syntax : `$bin/kafka-topics.sh --list --bootstrap-server localhost:9092`
- ▶ Output
Hello-Kafka

Apache Kafka - Basic Operations

Start Producer to Send Messages

- ▶ **Syntax** : `$bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic-name`
- ▶ From the above syntax, two main parameters are required for the producer command line client –
- ▶ **Broker-list** – The list of brokers that we want to send the messages to. In this case we only have one broker.


Apache Kafka - Basic Operations

Start Producer to Send Messages

- ▶ The Config/server.properties file contains broker port id, since we know our broker is listening on port 9092, so you can specify it directly.
- ▶ Topic name – Here is an example for the topic name.
- ▶ Example
 - `$bin/kafka-console-producer.sh --broker-list localhost:9092 --topic Hello-Kafka`

Apache Kafka - Basic Operations

Start Producer to Send Messages

- ▶ The producer will wait on input from stdin and publishes to the Kafka cluster.
 - ▶ By default, every new line is published as a new message then the default producer properties are specified in `config/producer.properties` file.
 - ▶ Now you can type a few lines of messages in the terminal as shown below.
- 


Apache Kafka - Basic Operations

Start Producer to Send Messages

► Output

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic Hello-Kafka
```

```
[2016-01-16 13:50:45,931] WARN property topic is not valid  
(kafka.utils.VerifiableProperties)
```

- Hello
 - My first message
 - My second message
- 

Apache Kafka - Basic Operations

Start Consumer to Receive Messages

- ▶ Similar to producer, the default consumer properties are specified in `config/consumer.properties` file. Open a new terminal and type the below syntax for consuming messages.
- ▶ **Syntax:** `$bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic-name --from-beginning`
- ▶ **Example :** `$bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning`

Apache Kafka - Basic Operations


Start Consumer to Receive Messages

▶ **Output**

- Hello
 - My first message
 - My second message
-
- ▶ Finally, you are able to enter messages from the producer's terminal and see them appearing in the consumer's terminal.

Apache Kafka - Basic Operations

Single Node-Multiple Brokers Configuration

- ▶ Before moving on to the multiple brokers cluster setup, first start your ZooKeeper server.
 - ▶ Create Multiple Kafka Brokers – We have one Kafka broker instance already in `config/server.properties`.
 - ▶ Now we need multiple broker instances, so copy the existing `server.properties` file into two new config files and rename it as `server-one.properties` and `server-two.properties`.
- 

Apache Kafka - Basic Operations

Single Node-Multiple Brokers Configuration

- ▶ Then edit both new files and assign the following changes –
 - config/server-one.properties

The id of the broker. This must be set to a unique integer for each broker.
broker.id=1

The port the socket server listens on
port=9093

A comma separated list of directories under which to store log files
log.dirs=/tmp/kafka-logs-1

Apache Kafka - Basic Operations

Single Node-Multiple Brokers Configuration

- ▶ `config/server-two.properties`

The id of the broker. This must be set to a unique integer for each broker.
`broker.id=2`

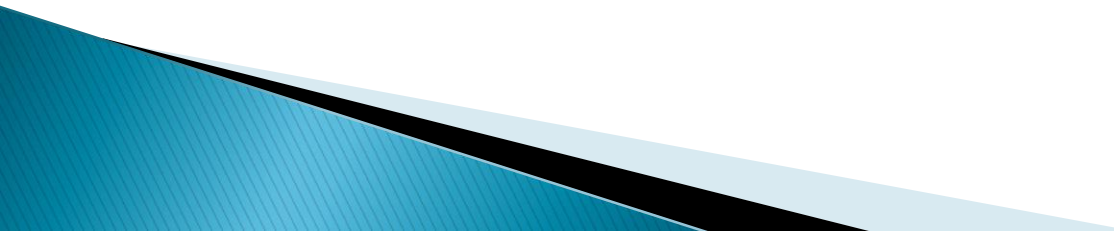
The port the socket server listens on
`port=9094`

A comma separated list of directories under which to store log files
`log.dirs=/tmp/kafka-logs-2`



Apache Kafka - Basic Operations

Single Node-Multiple Brokers Configuration

- ▶ Start Multiple Brokers— After all the changes have been made on three servers then open three new terminals to start each broker one by one.
 - ▶ Broker1 : `$bin/kafka-server-start.sh config/server.properties`
 - ▶ Broker2 : `$bin/kafka-server-start.sh config/server-one.properties`
- 

Apache Kafka - Basic Operations

Single Node-Multiple Brokers Configuration

- ▶ Broker3 : `$bin/kafka-server-start.sh config/server-two.properties`
- ▶ Now we have three different brokers running on the machine.
- ▶ Try it by yourself to check all the daemons by typing `jps` on the ZooKeeper terminal, then you would see the response.

Apache Kafka - Basic Operations

Creating a Topic

- ▶ Let us assign the replication factor value as three for this topic because we have three different brokers running.
- ▶ If you have two brokers, then the assigned replica value will be two.
- ▶ Syntax : `$bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 2 --topic topic-name`

Apache Kafka - Basic Operations

Creating a Topic

- ▶ Example: `$bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic Multibrokerapplication`
- ▶ Output
 - created topic “Multibrokerapplication”

Apache Kafka - Basic Operations

Creating a Topic

- ▶ The Describe command is used to check which broker is listening on the current created topic as shown below –

```
$bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic  
Multibrokerapplication
```

- ▶ Output

```
Topic:Multibrokerapplication PartitionCount:1 ReplicationFactor:3
```



Apache Kafka - Basic Operations

Creating a Topic

Configs:

Topic:Multibrokerapplication Partition:0 Leader:0 Replicas:0,2,1 Isr:0,2,1

- ▶ From the above output, we can conclude that first line gives a summary of all the partitions, showing topic name, partition count and the replication factor that we have chosen already.
- ▶ In the second line, each node will be the leader for a randomly selected portion of the partitions.

Apache Kafka - Basic Operations

Creating a Topic

- ▶ In our case, we see that our first broker (with broker.id 0) is the leader. Then Replicas:0,2,1 means that all the brokers replicate the topic finally Isr is the set of in-sync replicas.
- ▶ Well, this is the subset of replicas that are currently alive and caught up by the leader.

Apache Kafka - Basic Operations

Start Producer to Send Messages

- ▶ This procedure remains the same as in the single broker setup.
- ▶ Example : `$bin/kafka-console-producer.sh --broker-list localhost:9092 --topic Multibrokerapplication`
- ▶ Output
[2016-01-20 19:27:21,045] WARN Property topic is not valid
(kafka.utils.VerifiableProperties)
This is single node-multi broker demo
This is the second message

Apache Kafka - Basic Operations

Start Consumer to Receive Messages

- ▶ This procedure remains the same as shown in the single broker setup.
- ▶ Example: `$bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Multibrokerapplication --from-beginning`
- ▶ Output
 - This is single node-multi broker demo
 - This is the second message

Basic Topic Operations

Modifying a Topic

- ▶ Now let us modify a created topic using the following command
- ▶ Syntax: `$bin/kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic topic_name --partitions count`
- ▶ Example: `$bin/kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic Hello-kafka --partitions 2`

Basic Topic Operations

Modifying a Topic

- ▶ We have already created a topic “Hello-Kafka” with single partition count and one replica factor.
- ▶ Now using “alter” command we have changed the partition count.
- ▶ Output
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected Adding partitions succeeded!

Basic Topic Operations

Deleting a Topic

- ▶ To delete a topic, you can use the following syntax.
- ▶ Syntax: `$bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic topic_name`
- ▶ Example: `$bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic Hello-kafka`
- ▶ Output: `> Topic Hello-kafka marked for deletion`