Need of Kafka

IntelliPaat

Problems faced without Kafka

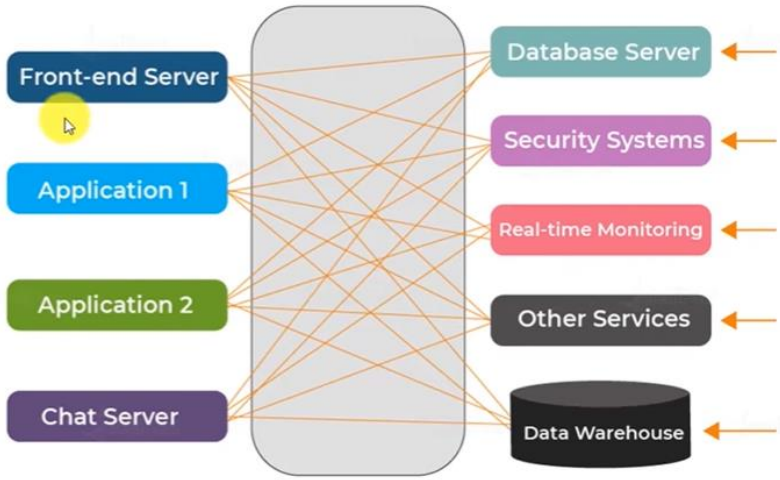
- The current-day industry is emerging with lots of real-time data that needs to be processed in real time. For example:
  - Sensor data that is used to predict the failure of a system ahead of time
  - Real-time economic data that is based on the preliminary estimates and is frequently adjusted for better estimates to be available
- Organizations can have multiple servers at front-end and back-end like the Web or Application Server for hosting websites or applications
- Now, all of these servers will need to communicate with the database server; thus, we will have multiple data pipelines connecting all of them to the database server

\

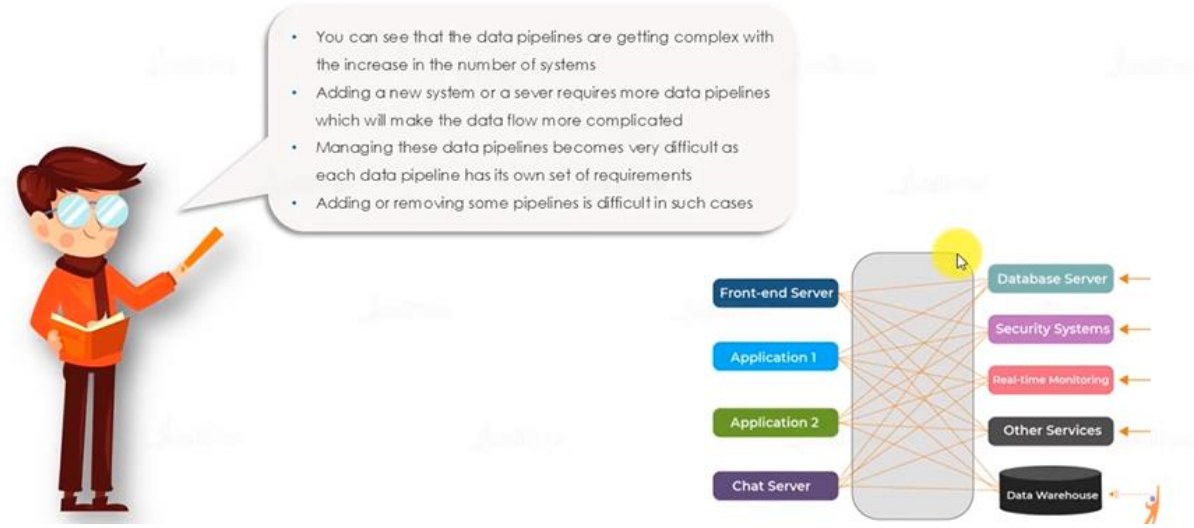
Need of Kafka

IntelliPaat

Problems faced without Kafka

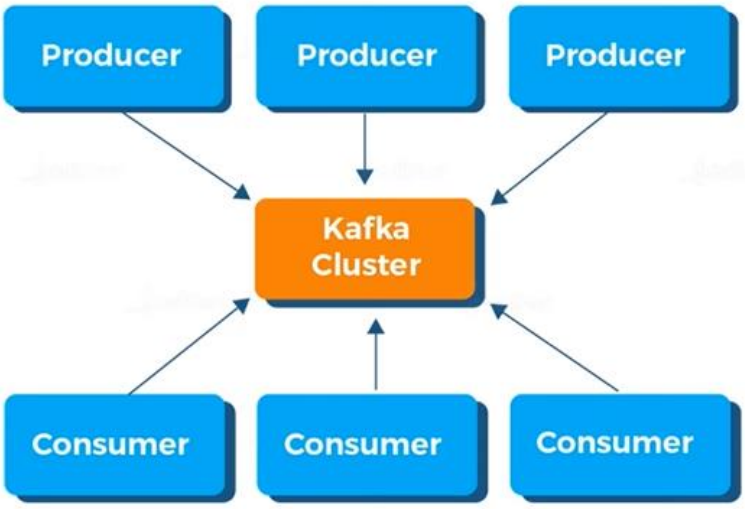


Need of Kafka



How does Kafka resolve this problem?

Kafka Decouples Data Pipelines



What is Kafka?

Apache Kafka: Introduction

Apache Kafka is an open-source, distributed, publish–subscribe messaging system that manages and maintains the real-time stream of data from different applications, websites, etc.

- Apache Kafka was originated at LinkedIn, later became an open-sourced Apache project in 2011, and then the first-class Apache project in 2012
- Kafka is written in Scala and Java
- Kafka is fast, scalable, durable, fault-tolerant, and distributed by design

Apache Kafka
A high-throughput distributed messaging system

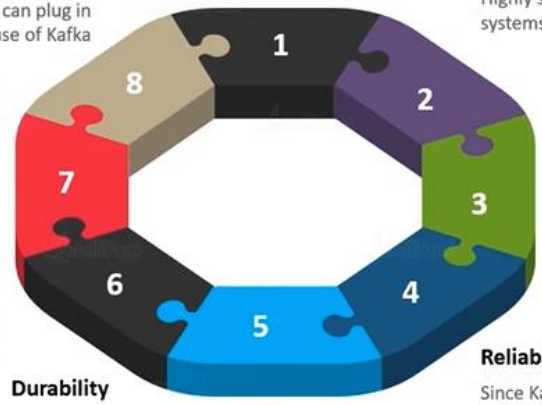
Let's Recall the Solution Provided by Kafka

Kafka Decouples Data Pipelines

- Apache Kafka reduces the complexity of data pipelines
- It makes communication between systems simpler and manageable
- With Kafka, it is easy to establish remote communication and send data across a network
- We can establish asynchronous communication and send messages with the help of Kafka
- It ensures reliable communication

Kafka Features

Features of Kafka



Extensibility

There are many ways by which applications can plug in and make use of Kafka

Scalability

Highly scalable distributed systems with no downtime

Zero Downtime

Kafka is very fast and guarantees zero downtime and zero data loss

High Volume

Works with the huge volume of data streams, easily

Performance

For both publishing and subscribing messages, Kafka has high throughput and maintains stable performance for TBs of messages

Fault Tolerance

The Kafka cluster can handle failures with masters and databases

Durability

Kafka is durable because it uses distributed commit log, i.e., messages

Reliability

Since Kafka is distributed, partitioned, replicated, and fault tolerant, it is very reliable

Components of Kafka

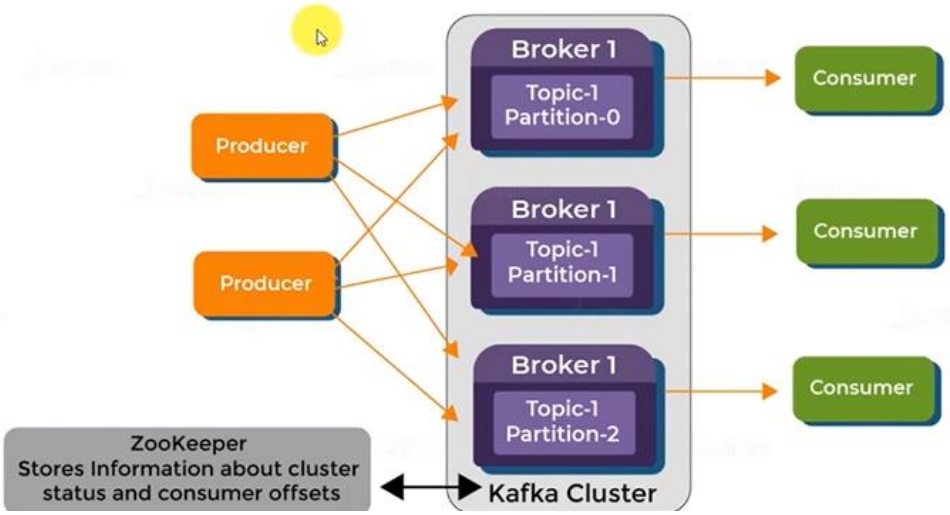
Broker	• Kafka brokers are the servers that manage and mediate the conversation between two different systems • Brokers are responsible for the delivery of messages to the right party
Message	• Messages are simply byte arrays, and any object can be stored in any format by developers • The format can be String, JSON, Avro, etc.
Topic	• In Apache Kafka, all messages are maintained in what we call topics • Messages are stored, published, and organized in Kafka topics
Cluster	• In Kafka, more than one broker, i.e., a set of servers is collectively known as a Kafka cluster • It is a group of computers, each having one instance of a Kafka broker

Components of Kafka

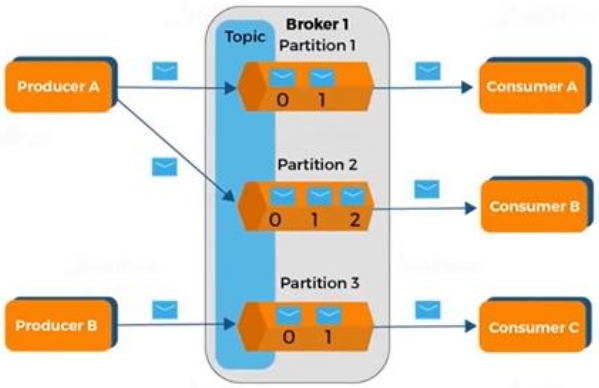
Producers	• Producers are the processes that publish data or messages to one or more topics • They are basically the source of data stream in Kafka
Consumers	• Consumers are the processes that read and process the data from topics by subscribing to one or more topics in the Kafka cluster
Partitions	• Every broker holds few partitions and each partition can be either a leader or a replica for a topic • All 'writes' and 'reads' to a topic go via the leader, which is responsible for updating replicas with new data • If the leader fails, the replica takes over as the new leader

Architecture of Kafka Cluster

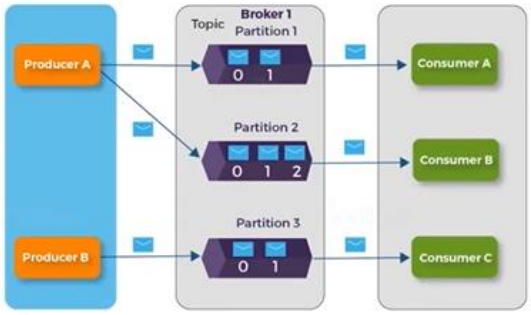
Architecture of Kafka Cluster



Architecture: The Inside View

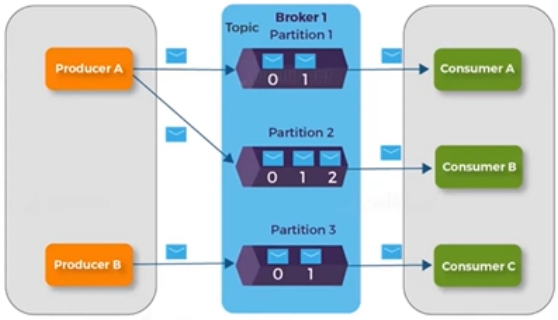


Kafka Producer



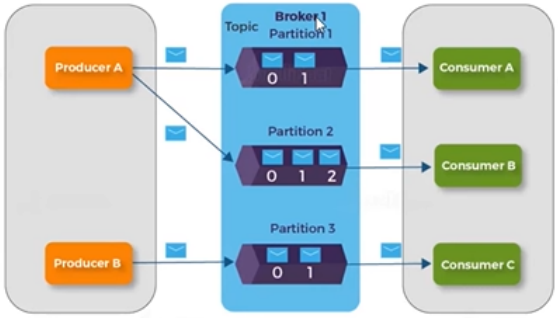
- Producers send records (also referred to as messages) to topics
- Producers select the partition to send the message per topic
- Producers can implement priority systems, which are based on sending records to certain partitions depending on the priority of the record
- Producers send records to a partition based on the record's key
- Producers don't wait for acknowledgements from a broker and send messages as fast as the broker can handle

Kafka Broker



- A cluster typically consists of multiple brokers to maintain the load balance
- A broker on receiving messages from the producer assigns offsets to them and commits the messages to the storage on the disk
- It serves consumers by responding to fetch requests for partitions
- One broker instance can handle thousands of reads–writes per second and TBs of messages
- Backups of topic partitions are present in multiple brokers
- If a broker goes down, one of the brokers containing the backup partitions would be elected as the leader for the respective partitions

Kafka Topics and Partitions



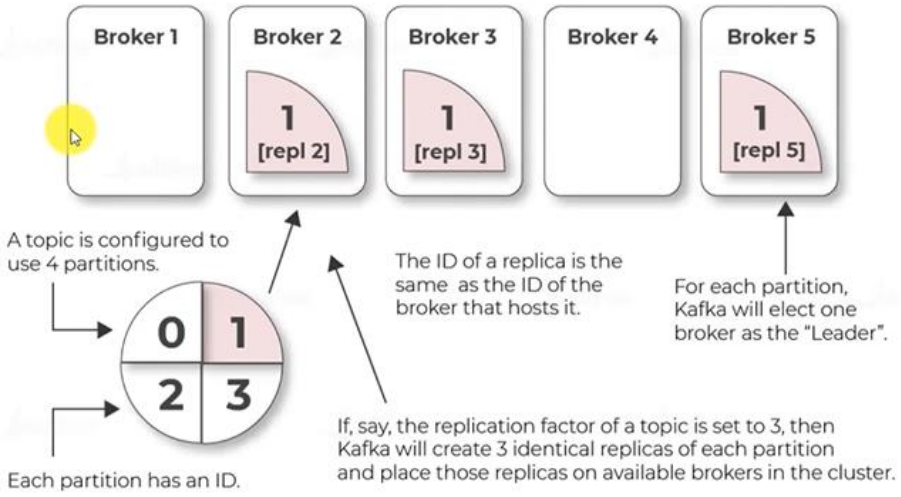
- Messages in Kafka are categorized into topics
- Topics are broken down into a number of partitions
- Messages are written to it in an append-only fashion
- Reading messages can either be done in the order from beginning to end or skip or rewind to any point in the partition by providing an offset value
- An offset value is the sequential ID provided to messages
- Partitions provide redundancy and scalability
- Partitions can be hosted on a different server, i.e., a single topic can be scaled horizontally across multiple servers, thus enhancing the performance

Kafka Topics and Partitions

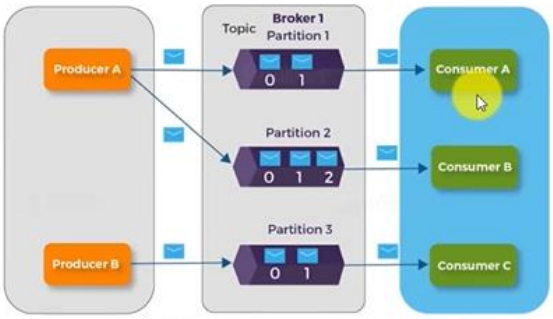
Representation of a topic with multiple partitions

- The figure shows a topic with four partitions, with writes being appended to the end of each
- A record is stored on a partition either by the record key if the key is present or by round-robin if the key is missing (the default behavior)

Kafka Topics and Partitions

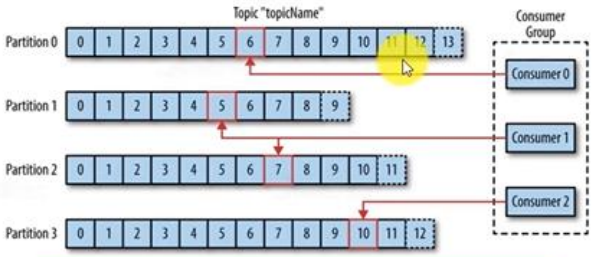


Kafka Consumer



- The consumer can subscribe to one or more topics and read messages in the order they were produced
- The consumer keeps track of the messages it has already consumed by keeping the track of the offset of messages
- Consumers work as part of a consumer group, i.e., one or more consumers that work together to consume a topic
- Messages with the same key arrive at the same consumer
- The group assures that each partition is consumed by only one member

Kafka Consumer



A consumer group reading from a topic

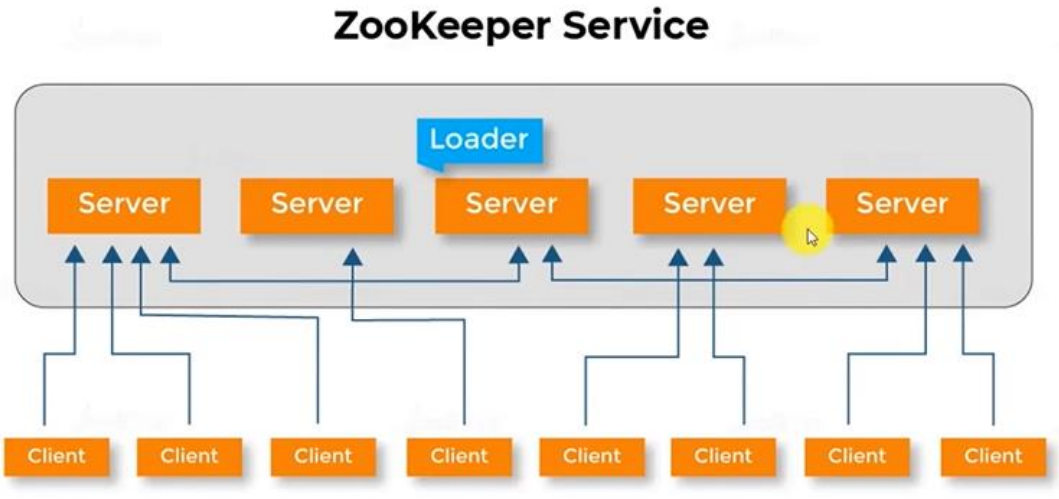
- Three consumers in a single group, consuming a topic
- Two consumers are working on one partition each, while the third consumer is working on two partitions

Apache ZooKeeper

ZooKeeper is an open-source Apache project that provides centralized infrastructure and services that enable synchronization across an Apache Hadoop cluster

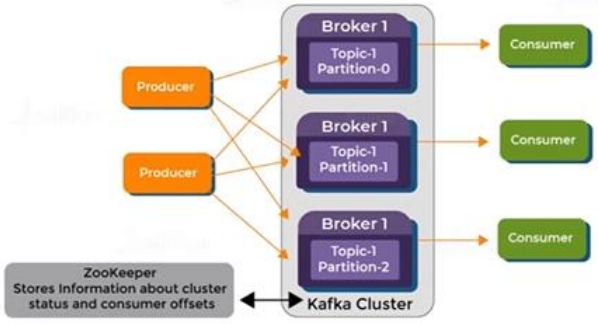
- Developed originally at Yahoo, ZooKeeper facilitates synchronization in the process by maintaining a status on ZooKeeper servers, which store information in local log files
- ZooKeeper servers are capable of supporting a large Hadoop cluster

ZooKeeper Architecture



ZooKeeper and Kafka

- Kafka brokers coordinate with each other using ZooKeeper
- Producers and consumers are notified by the ZooKeeper service about the presence of a new broker in the system or about the failure of a broker in the system
- If the leader node fails, then on the basis of the currently live nodes, Apache ZooKeeper will elect the new leader
- ZooKeeper in Kafka keeps a set of in-sync replicas



Kafka Workflow

Workflow of Pub/Sub Messaging

IntelliPaat

- Producers will send messages to a topic at regular intervals
- Brokers store the messages in partitions configured for that particular topic
- If a producer sends two messages and there exist two partitions, Kafka will store one message in the first partition and the second message in the second
- A consumer always subscribes to a specific topic
- When the consumer subscribes to a topic, Kafka provides the current offset of the topic to the consumer and the offset is saved in the ZooKeeper ensemble
- For new messages, the consumer will request Kafka in a regular interval (like in every100 minutes)

Publish/Subscribe Pattern

Workflow of Pub/Sub Messaging

IntelliPaat

- As soon as the message is received from the producer, it is forwarded to consumers
- On receiving the message, consumers will process it
- Once the message is processed, an acknowledgement is sent to the broker
- On receiving the acknowledgement, the offset is changed to the new value and is updated in ZooKeeper
- The consumers are able to read the new message correctly even during server outrages since the offsets are maintained in ZooKeeper
- The flow repeats until the consumers stop the request

Publish/Subscribe Pattern

Who uses Apache Kafka?

Companies Using Apache Kafka

IntelliPaat

List of the top companies using Apache Kafka :

Company	Website	Country	Revenue	Company Size
Samsung Electronics	samsung.com	Korea, Republic of	>1000M	>10000
HOMEAWAY INC	homeaway.com	United States	200M-1000M	1000-5000
Square Inc	squareup.com	United States	>1000M	1000-5000
Harman International Industries, Inc.	harman.com	United States	>1000M	>10000
Nanigans, Inc.	nanigans.com	United States	10M-50M	50-200

Kafka Setup

Answer 1

Each Kafka partition has one server that acts as the _____.

A	Leaders
B	Followers
C	Staters
D	All of the above

Answer 2

Kafka provides only a _____ order over messages within a partition.

A	Partial
B	Total
C	30%
D	None of the mentioned

Answer 3

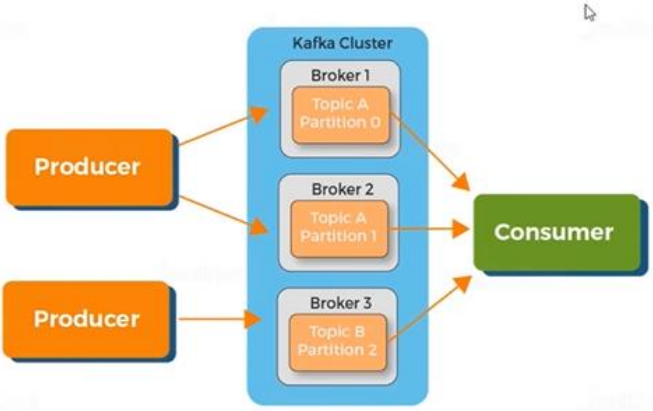
Kafka maintains feeds of messages in categories called _____.

A	Topics
B	Chunks
C	Domains
D	Partitions

Kafka Cluster

Kafka Cluster

- A single Kafka server works well for the local development work
- There are significant benefits of having multiple brokers configured as a cluster
- The biggest benefit is the ability to scale the load across multiple servers
- Replications help in performance maintenance of the Kafka cluster or the underlying systems
- A Kafka cluster is effective for applications that involve large-scale message processing



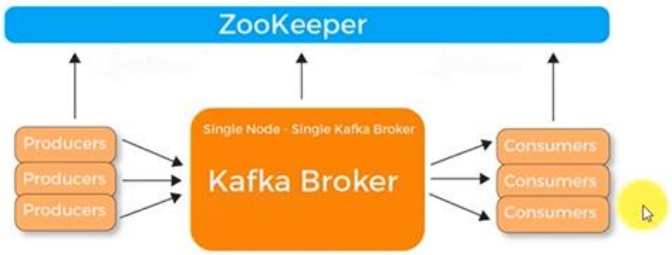
Kafka Command-line Tools

- The Kafka cluster can run against the following broker setup:
  - Single Broker Cluster
  - Multiple Broker Cluster
- Some of the commonly used commands are:

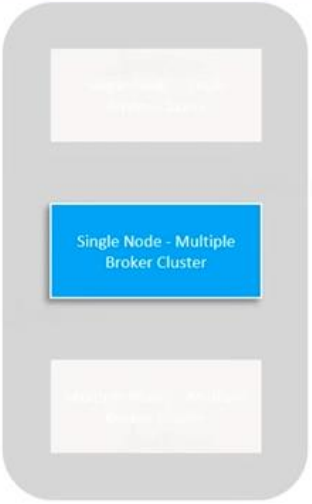
Kafka Shell Script	Description
`zookeeper-server-start.sh`	It starts ZooKeeper using the properties configured under config/zookeeper.properties
`kafka-server-start.sh`	It starts the Kafka server using the properties configured under config/server.properties
`kafka-topics.sh`	It is used to create and list topics
`kafka-console-producer.sh`	A command-line client to send messages to the Kafka cluster
`kafka-console-consumer.sh`	A command line client to consume messages from the Kafka cluster

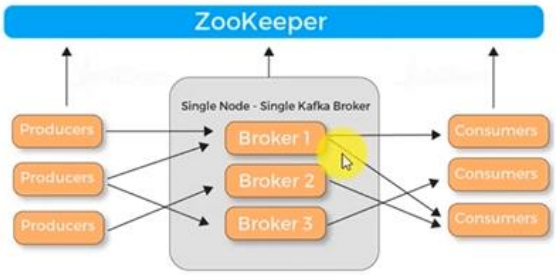
Types of Kafka Cluster

Kafka Cluster: Types

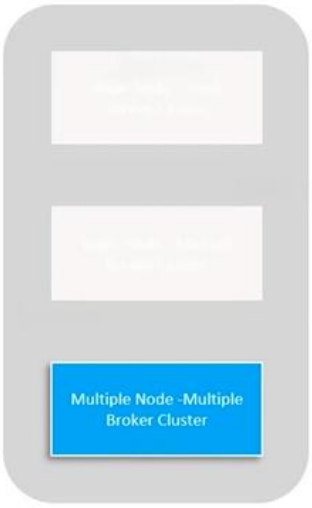


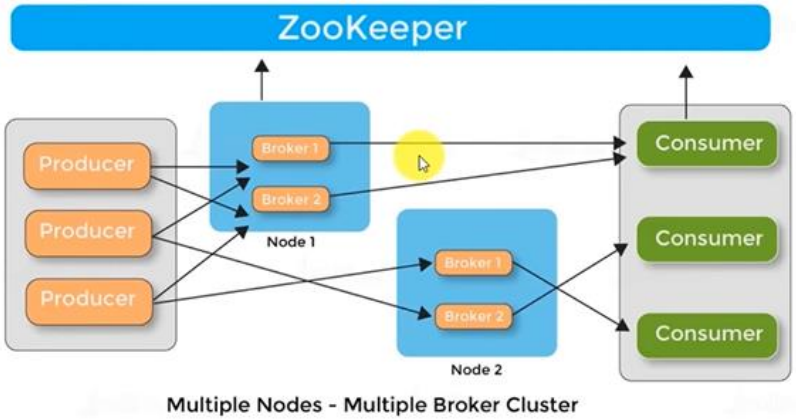
Kafka Cluster: Types





Kafka Cluster: Types





Multiple Nodes - Multiple Broker Cluster

Configuring Single Node - Single Broker Cluster

Prerequisites

- Your system should have:
  - Java
  - Kafka
  - ZooKeeper
- An installation document is provided in order to download and setup the Kafka cluster

Single Broker Setup

- Open your terminal and start ZooKeeper, after which start Kafka broker

```
zookeeper-server-start.sh kafka/config/zookeeper.properties
kafka-server-start.sh kafka/config/server.properties
```

```
[training@ip-172-31-27-203 ~]$ zookeeper-server-start.sh kafka/config/zookeeper.
properties
[2018-11-23 07:48:40,852] INFO Reading configuration from: kafka/config/zookeepe
r.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2018-11-23 07:48:40,855] INFO autopurge.snapRetainCount set to 3 (org.apache.zo
okeeper.server.DatadirCleanupManager)
```

```
[training@ip-172-31-27-203 ~]$ kafka-server-start.sh kafka/config/server.propert
ies
[2018-11-23 07:50:32,395] INFO Registered kafka:type=kafka.Log4jController MBean
 (kafka.utils.Log4jControllerRegistration$)
[2018-11-23 07:50:32,813] INFO starting (kafka.server.KafkaServer)
[2018-11-23 07:50:32,814] INFO Connecting to zookeeper on localhost:2181 (kafka.
server.KafkaServer)
```

Single Broker Setup

- You can use the `jps` command to check whether both services have started or not
- Now, you can see two daemons running on the terminal, where **`QuorumPeerMain`** is a **`ZooKeeper daemon`** and the other one is a **`Kafka daemon`**

```
[training@ip-172-31-27-203 ~]$ jps
2400 ResourceManager
1921 NameNode
2515 NodeManager
26051 Kafka
25605 QuorumPeerMain
26459 Jps
2060 DataNode
2238 SecondaryNameNode
```

Single Broker Setup

- **Creating a Kafka Topic**

```
kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --
partitions 1 --topic Example1
```

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --create --zookeeper localhost:21
81 --replication-factor 1 --partitions 1 --topic Example1
Created topic "Example1".
```

- **List of Topics**

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181
Example1
__consumer_offsets
my-kafka-topic
```

Single Broker Setup

- **Start the producer to send messages**

```
kafka-console-producer.sh --broker-list localhost:9092 --topic Example1
```

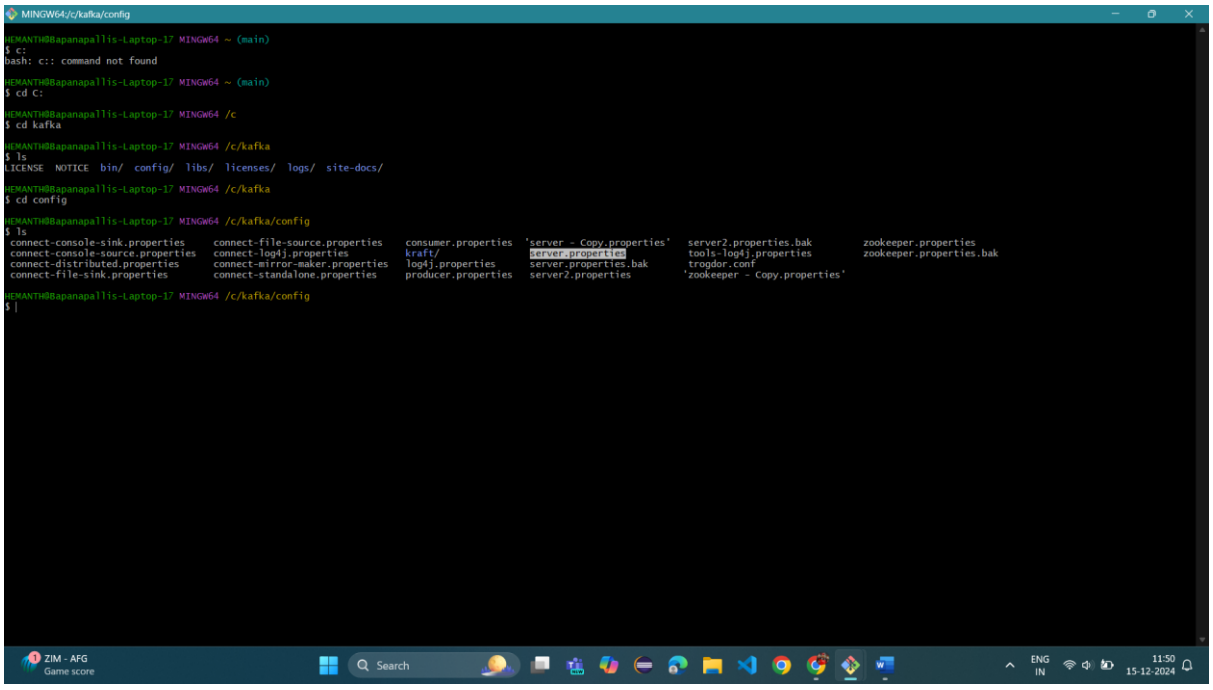
```
[training@ip-172-31-27-203 ~]$ kafka-console-producer.sh --broker-list localhost
:9092 --topic Example1
>
>hello
>this is my first ex
```

Single Broker Setup

- Start the consumer to receive messages

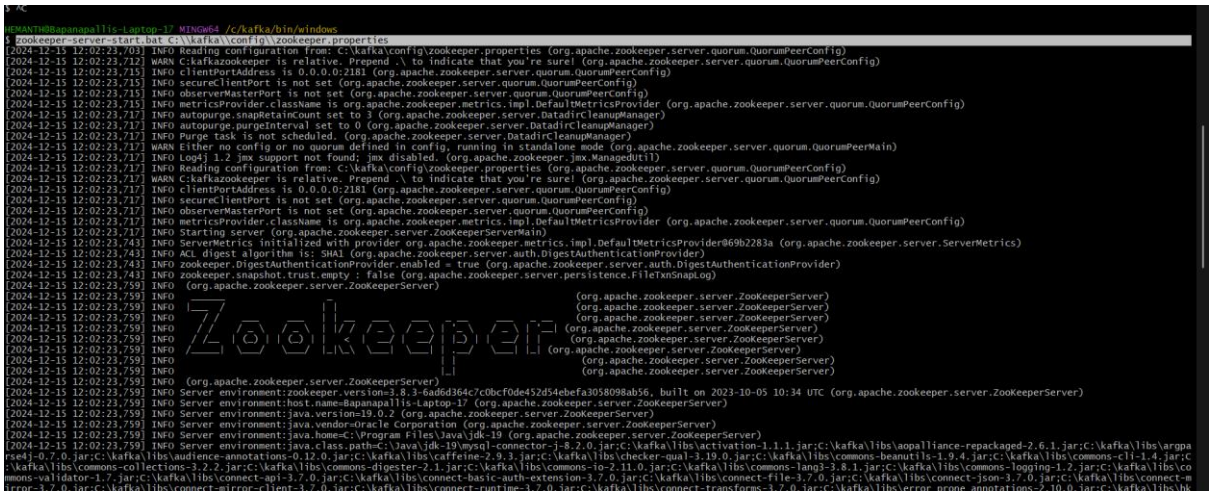
```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Example1 --from-beginning
```

```
[training@ip-172-31-27-203 ~]$ kafka-console-consumer.sh --bootstrap-server loca
lhost:9092 --topic Example1 --from-beginning

hello
this is my first example
```

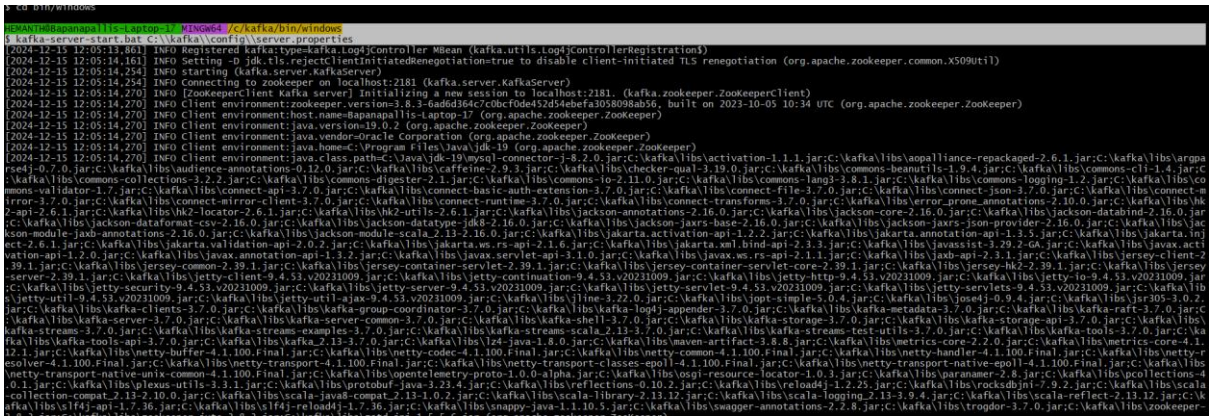


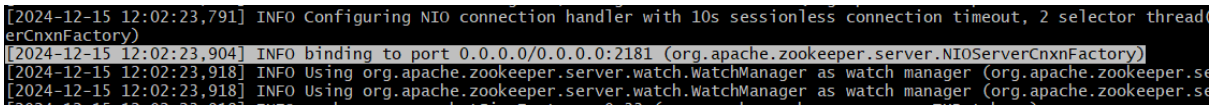
```
HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ cd bin

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin
$ ls
connect-distributed.sh*        kafka-configs.sh*             kafka-e2e-latency.sh*        kafka-mirror-maker.sh*          kafka-streams-application-reset.sh*  zookeeper-server-start.sh*
connect-mirror-maker.sh*       kafka-console-consumer.sh*    kafka-features.sh*           kafka-producer-perf-test.sh*    kafka-topics.sh*                     zookeeper-server-stop.sh*
connect-plugin-path.sh*        kafka-console-producer.sh*    kafka-get-offsets.sh*        kafka-reassign-partitions.sh*   kafka-transactions.sh*               zookeeper-shell.sh*
connect-standalone.sh*         kafka-consumer-groups.sh*     kafka-jmx.sh*                kafka-replica-verification.sh*  kafka-verifiable-consumer.sh*
kafka-acls.sh*                 kafka-consumer-perf-test.sh*  kafka-leader-election.sh*    kafka-run-class.sh*             kafka-verifiable-producer.sh*
kafka-broker-api-versions.sh*  kafka-delegation-tokens.sh*   kafka-log-dirs.sh*           kafka-server-start.sh*          trogdor.sh*
kafka-client-metrics.sh*       kafka-delete-records.sh*      kafka-metadata-quorum.sh*    kafka-server-stop.sh*           windows/
kafka-cluster.sh*              kafka-dump-log.sh*            kafka-metadata-shell.sh*     kafka-storage.sh*               zookeeper-security-migration.sh*

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin
$ cd windows

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$ ls
connect-distributed.bat        kafka-configs.bat             kafka-dump-log.bat           kafka-metadata-quorum.bat       kafka-server-stop.bat                kafkazookeeper/
connect-plugin-path.bat        kafka-console-consumer.bat    kafka-e2e-latency.bat        kafka-mirror-maker.bat          kafka-storage.bat                    zookeeper-server-start.bat
connect-standalone.bat         kafka-console-producer.bat    kafka-features.bat           kafka-producer-perf-test.bat    kafka-streams-application-reset.bat  zookeeper-server-stop.bat
kafka-acls.bat                 kafka-consumer-groups.bat     kafka-get-offsets.bat        kafka-reassign-partitions.bat   kafka-topics.bat                     zookeeper-shell.bat
kafka-broker-api-versions.bat  kafka-consumer-perf-test.bat  kafka-jmx.bat                kafka-replica-verification.bat  kafka-transactions.bat
kafka-client-metrics.bat       kafka-delegation-tokens.bat   kafka-leader-election.bat    kafka-run-class.bat             kafkakafka-logs/
kafka-cluster.bat              kafka-delete-records.bat      kafka-log-dirs.bat           kafka-server-start.bat          kafkalogs-2/

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$ ls
connect-distributed.bat        kafka-configs.bat             kafka-dump-log.bat           kafka-metadata-quorum.bat       kafka-server-stop.bat                kafkazookeeper/
connect-plugin-path.bat        kafka-console-consumer.bat    kafka-e2e-latency.bat        kafka-mirror-maker.bat          kafka-storage.bat                    zookeeper-server-start.bat
connect-standalone.bat         kafka-console-producer.bat    kafka-features.bat           kafka-producer-perf-test.bat    kafka-streams-application-reset.bat  zookeeper-server-stop.bat
kafka-acls.bat                 kafka-consumer-groups.bat     kafka-get-offsets.bat        kafka-reassign-partitions.bat   kafka-topics.bat                     zookeeper-shell.bat
kafka-broker-api-versions.bat  kafka-consumer-perf-test.bat  kafka-jmx.bat                kafka-replica-verification.bat  kafka-transactions.bat
kafka-client-metrics.bat       kafka-delegation-tokens.bat   kafka-leader-election.bat    kafka-run-class.bat             kafkakafka-logs/
kafka-cluster.bat              kafka-delete-records.bat      kafka-log-dirs.bat           kafka-server-start.bat          kafkalogs-2/

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$
```

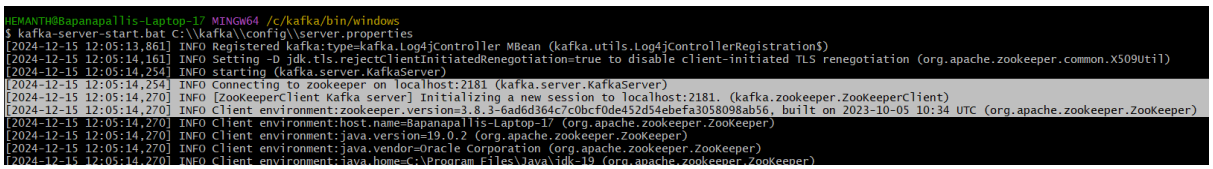
Zookeeper server starting command:



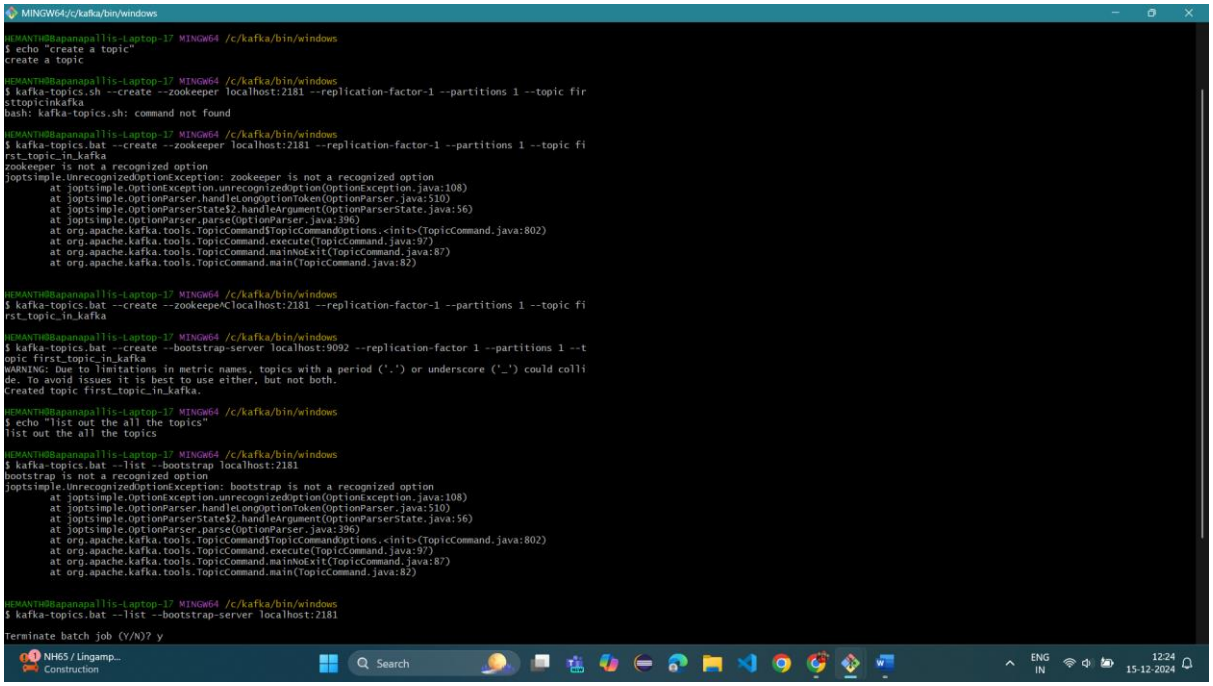
Kafka server starting command:





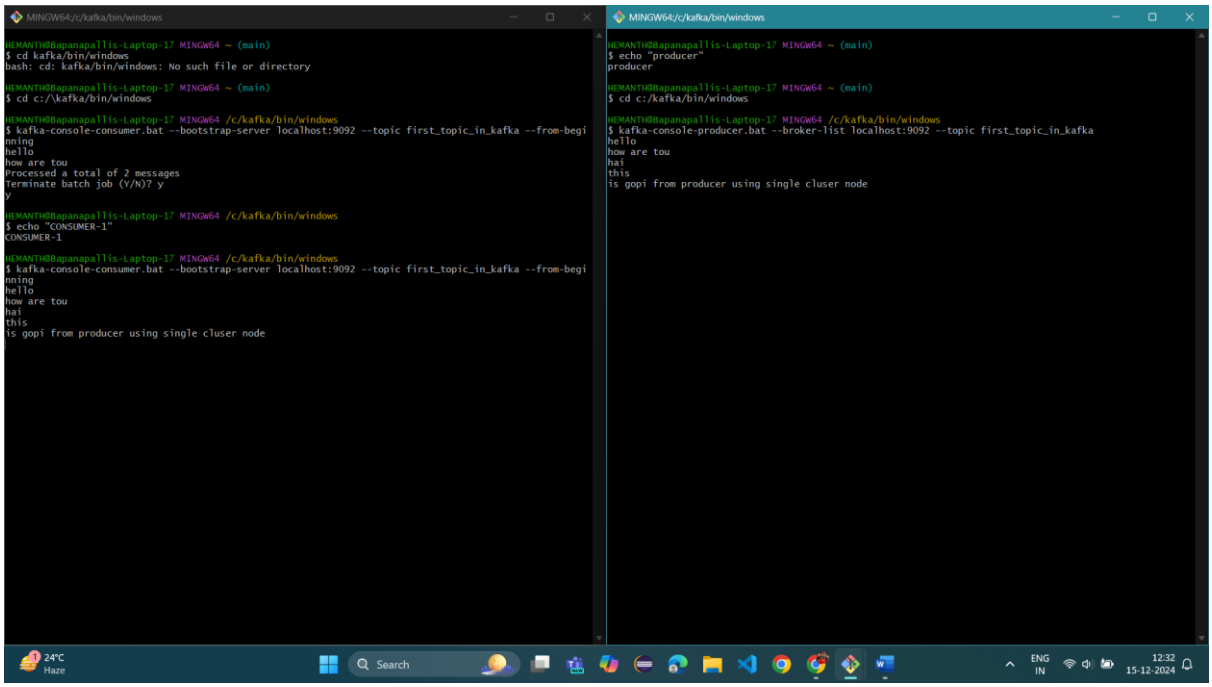


Creating a topic



```
HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$ kafka-topics.bat --list --bootstrap-server localhost:2181

Terminate batch job (Y/N)? y
y

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$ kafka-topics.bat --list --bootstrap-server localhost:9092
USER_CREATED
__consumer_offsets
abd
first_topic_in_kafka
virat

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/bin/windows
$
```

Create a consumer and producer for topic



Configuring Single Node - Multi Broker Cluster

Multi Broker Setup

- **Open your terminal and start ZooKeeper, after which start Kafka broker**

```
zookeeper-server-start.sh kafka/config/zookeeper.properties
kafka-server-start.sh kafka/config/server.properties
```

```
[training@ip-172-31-27-203 ~]$ zookeeper-server-start.sh kafka/config/zookeeper.
properties
[2018-11-23 07:48:40,852] INFO Reading configuration from: kafka/config/zookeepe
r.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2018-11-23 07:48:40,855] INFO autopurge.snapRetainCount set to 3 (org.apache.zo
okeeper.server.DatadirCleanupManager)
```

```
[training@ip-172-31-27-203 ~]$ kafka-server-start.sh kafka/config/server.propert
ies
[2018-11-23 07:50:32,395] INFO Registered kafka:type=kafka.Log4jController MBean
 (kafka.utils.Log4jControllerRegistration$)
[2018-11-23 07:50:32,813] INFO starting (kafka.server.KafkaServer)
[2018-11-23 07:50:32,814] INFO Connecting to zookeeper on localhost:2181 (kafka.
server.KafkaServer)
```

Multi Broker Setup

Create Multiple Kafka Brokers:

- We have one Kafka broker instance already in config/server.properties
- We need multiple broker instances, so we will copy the existing server.properties file into two new files and rename them as server1.properties and server2.properties
- Get inside the `cd kafka/config` path and create the files

```
[training@ip-172-31-27-203 config]$ cp server.properties server1.properties
```

```
[training@ip-172-31-27-203 config]$ cp server.properties server2.properties
```

Multi Broker Setup

Open the server1.properties file and make the following changes:

- broker.id=1
- listeners=PLAINTEXT://:9093
- log.dirs=/tmp/kafka-logs1

Open the server2.properties file and make the following changes:

- broker.id=2
- listeners=PLAINTEXT://:9094
- log.dirs=/tmp/kafka-logs2

These three properties have to be unique for each broker instance, while you don't have to worry about the others

Multi Broker Setup

Start Multiple Kafka Brokers

```
kafka-server-start.sh kafka/config/server1.properties
kafka-server-start.sh kafka/config/server2.properties
```

Multi Broker Setup

- **Create a Kafka Topic**

```
kafka-topics.sh --create --topic Example2 --zookeeper localhost:2181 --partitions 3 -
-replication-factor 2
```

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --create --topic Example2 --zookeeper localhost:2181 --partitions 3 --repli
cation-factor 2
Created topic "Example2".
```

- The **describe** command is used to check which broker is listening on the current created topic

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --describe --zookeeper localhost:2181 --topic Example2
Topic:Example2  PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: Example2 Partition: 0    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: Example2 Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: Example2 Partition: 2    Leader: 1       Replicas: 1,2   Isr: 1,2
```

Multi Broker Setup

- **Start the producer to send messages**

```
kafka-console-producer.sh --broker-list
localhost:9093,localhost:9094,localhost:9092 --topic Example2
```

```
[training@ip-172-31-27-203 ~]$ kafka-console-producer.sh --broker-list localhost:9093,localhost:9094,localhost:9092 --topi
c Example2
>Bonjour
>Do you understand kafka now?
>Hope you liked the session !!
```

Multi Broker Setup

- **Start the consumer to receive messages**

```
kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic Example2 --
from-beginning
```

```
[training@ip-172-31-27-203 ~]$ kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic Exa
mple2 --from-beginning

Bonjour
Do you understand kafka now?
Hope you liked the session !!
```

Example in practical

Creating logs for multiple brokers

```
HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ ls
LICENSE  NOTICE  bin/  config/  libs/  licenses/  logs/  site-docs/

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ mkdir logs-server-1

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ ls
LICENSE  NOTICE  bin/  config/  libs/  licenses/  logs/  logs-server-1/  site-docs/

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ mkdir logs-server-2

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka
$ cd config

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/config
$ ls
 connect-console-sink.properties     connect-file-source.properties    consumer.properties  'server - Copy.properties'   server.properties.bak    trogdor.conf
 connect-console-source.properties   connect-log4j.properties         kraft/               server-1.properties          server2.properties       'zookeeper - Copy.properties'
 connect-distributed.properties      connect-mirror-maker.properties  log4j.properties     server-2.properties          server2.properties.bak   zookeeper.properties
 connect-file-sink.properties        connect-standalone.properties    producer.properties  server.properties            tools-log4j.properties   zookeeper.properties.bak

HEMANTH@Bapanapallis-Laptop-17 MINGW64 /c/kafka/config
```

Primary server

```
############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

############################# Socket Server Settings #############################

# The address the socket server listens on. If not configured, the host name will be equal to the value of
# java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
#advertised.listeners=PLAINTEXT://your.host.name:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the network and sending responses to the network
num.network.threads=3

# The number of threads that the server uses for processing requests, which may include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=C:\kafka\logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1
```

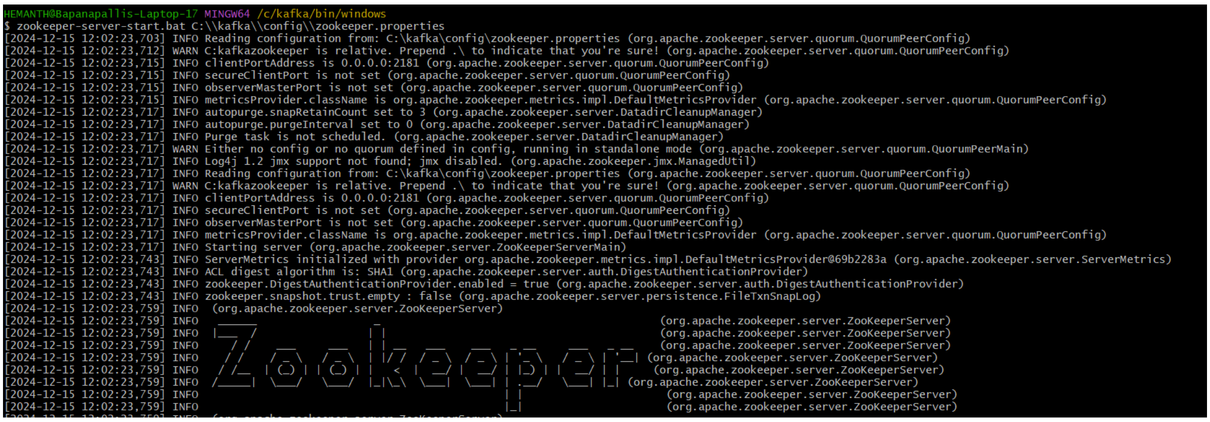
Server-1

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=3

############################# Socket Server Settings #############################

# The address the socket server listens on. If not configured, the host name will be equal to the value of
# java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9094

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
#advertised.listeners=PLAINTEXT://your.host.name:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the network and sending responses to the network
num.network.threads=3

# The number of threads that the server uses for processing requests, which may include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=C:\kafka\logs-server-1

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1
```

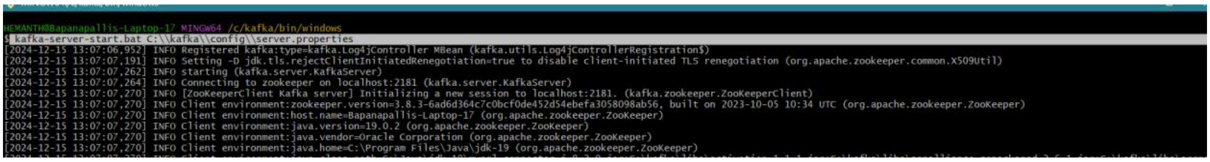
Server-2

```
#
############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=4

############################# Socket Server Settings #############################

# The address the socket server listens on. If not configured, the host name will be equal to the value of
# java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9095

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
#advertised.listeners=PLAINTEXT://your.host.name:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the network and sending responses to the network
num.network.threads=3

# The number of threads that the server uses for processing requests, which may include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=C:\kafka\logs-server-2
```

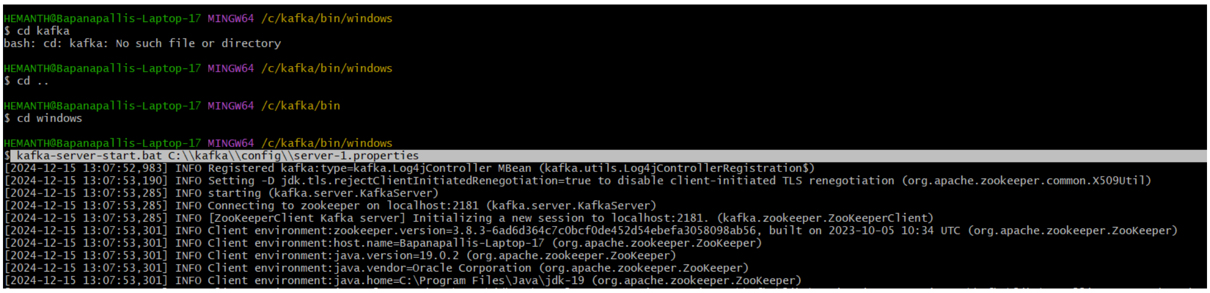
Zookeeper Running



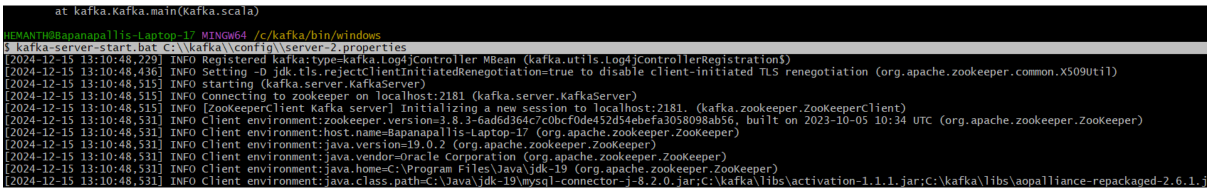
Start primary kafka server i.e broker-0 on localhost 9092



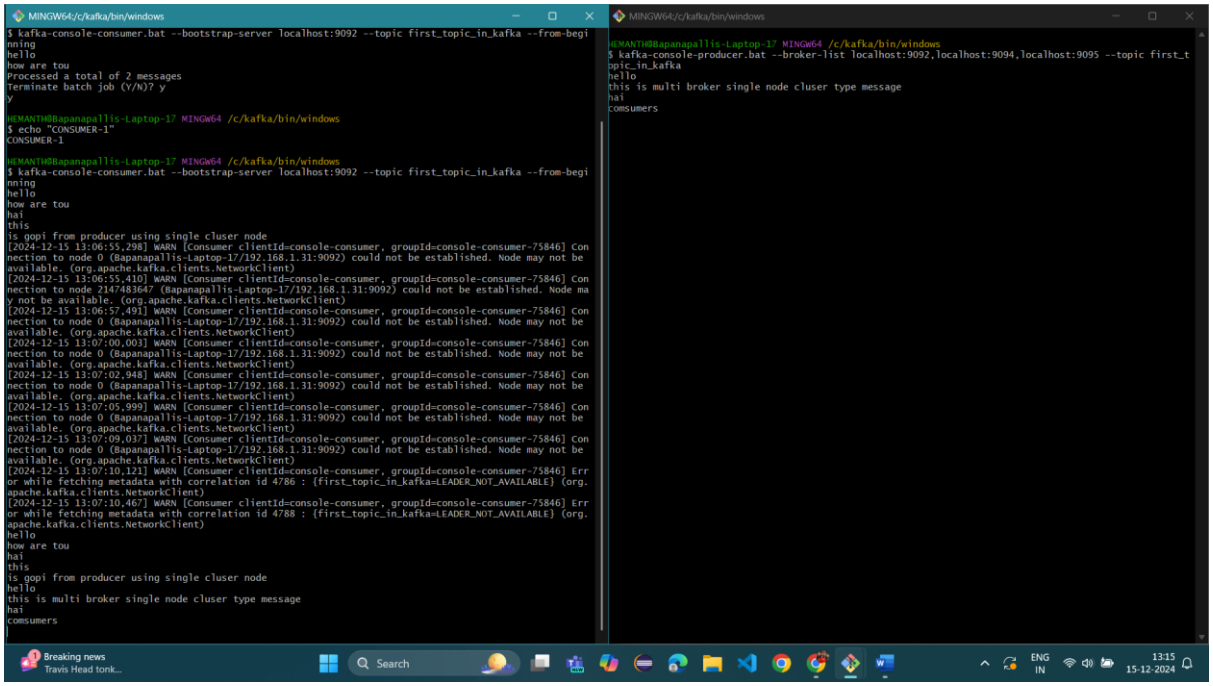
Server-1



Server-2



Producer and consumer setup



Basic Topic Operations

Basic Topic Operations

IntelliPaat

Modifying a Topic

```
kafka-topics.sh --zookeeper localhost:2181 --alter --topic Example1 --
partitions 2
```

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --zookeeper localhost:2181 --alter
r --topic Example1 --partitions 2
WARNING: If partitions are increased for a topic that has a key, the partition l
ogic or ordering of the messages will be affected
Adding partitions succeeded!
```

Deleting a Topic

Basic Topic Operations

Modifying a Topic

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181
Example1
Example2
Flume_topic1
__consumer_offsets
my-kafka-topic
```

Deleting a Topic

```
kafka-topics.sh --zookeeper localhost:2181 --delete --topic Flume_topic1
```

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --zookeeper localhost:2181 --dele
te --topic Flume_topic1
Topic Flume_topic1 is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Basic Topic Operations

Modifying a Topic

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181
Example1
Example2
__consumer_offsets
my-kafka-topic
```

Deleting a Topic

Answer 1

Kafka is run as a cluster comprised of one or more servers each of which is called:

A	cTakes
B	Broker
C	Test
D	None of the mentioned

Answer 2

Point out the wrong statement:

A	The Kafka cluster does not retain all published messages
B	A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients
C	Kafka is designed to allow a single cluster to serve as the central data backbone for a large organization
D	Messages are persisted on disk and replicated within the cluster to prevent data loss

- ✓ Multi Node Cluster Setup
- ✓ Administration Commands
- ✓ Graceful Shutdown
- ✓ Balancing Leadership
- ✓ Rebalancing Tools
- ✓ Expending Your cluster and Using partition Reassignment Tool
- ✓ Custom Partition Assignment
- ✓ Decommissioning Broker
- ✓ Increasing Replication Factor

Multi Node Kafka Cluster Implementation

Links

✓ Apache Mirror –

We suggest the following mirror site for your download:

http://mirror.fibergrid.in/apache/zookeeper/

Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures to v working.

HTTP

http://a.mbbsindia.com/zookeeper/

http://mirror.fibergrid.in/apache/zookeeper/

✓ Cloudera's Website

Apache ZooKeeper	zookeeper-3.4.5+28	Tarball	Release notes	Changes

Setup –

- ✓ **Perform on 1st Node and SCP to rest of the Nodes**
- ✓ **Untar –**
  - ✓ tar -xzf kafka_2.10-0.9.0.0.tgz
  - ✓ tar -xzf zookeeper-3.4.5-cdh4.4.0.tar.gz
- ✓ **Rename –**
  - ✓ mv kafka_2.10-0.9.0.0 kafka
  - ✓ mv zookeeper-3.4.5-cdh4.4.0 zookeeper
- ✓ **Setup the Kafka and zookeeper Home in .bashrc**
  - ✓ Add – export KAFKA_HOME=/home/kafka/kafka
    export ZOOKEEPER_HOME=/home/kafka/zookeeper
  - ✓ Run – source .bashrc
- ✓ **Make the Kafka & ZK log directories**
  - ✓ mkdir $KAFKA_HOME/kafka-logs
  - ✓ mkdir $ZOOKEEPER_HOME/zkdata - Where ZK services run