

PySpark

Apache Kafka and Flume



Agenda

- | | | | |
|----|--------------------|----|-------------------------|
| 01 | Need for Kafka | 02 | What is Kafka? |
| 03 | Kafka Architecture | 04 | Kafka Workflow |
| 05 | Kafka Cluster | 06 | Basic Topic Operations |
| 07 | Performance Tuning | 08 | Features of Flume |
| 09 | Kafka Applications | 10 | Kafka–Flume Integration |



What is the need for
Kafka?

Why do we need Kafka?

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 preliminary estimates and is frequently adjusted for better estimates to be available

Why do we need Kafka?

Problems faced without Kafka

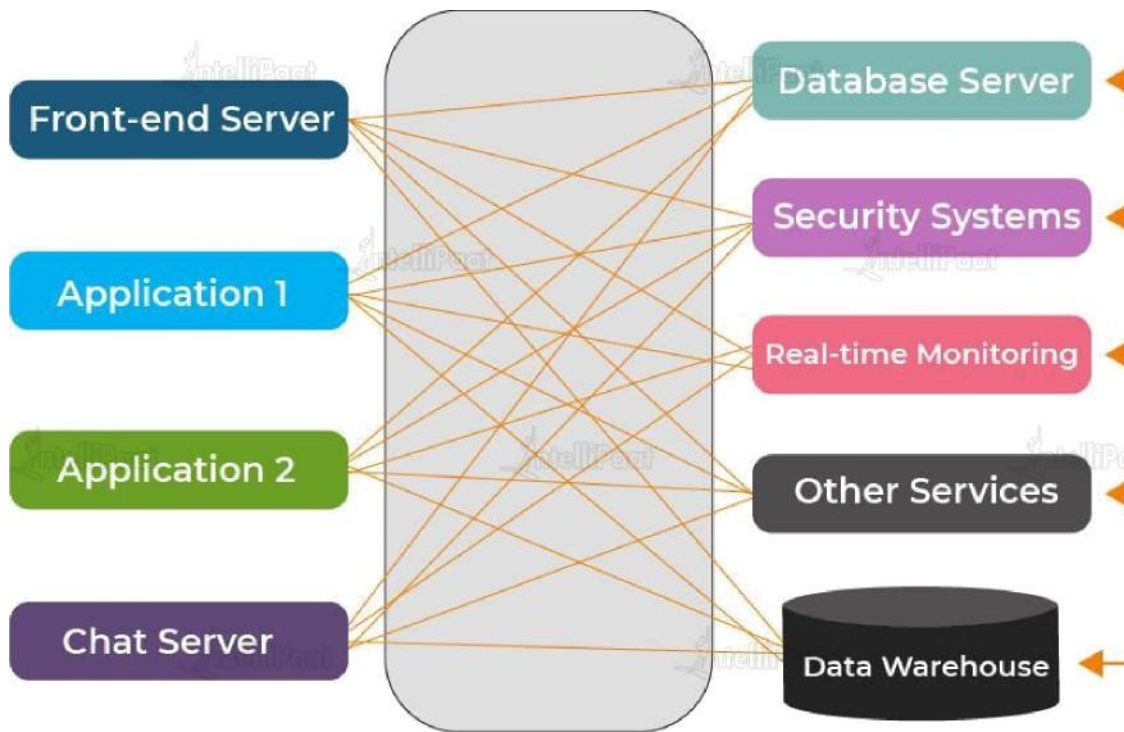
- The current-day industry is emerging with lots of real-time data that needs to be processed in real time. For example:
 - Sensor data that is used to predict the failure of a system ahead of time
 - Real-time economic data that is based on preliminary estimates and is frequently adjusted for better estimates to be available
- Organizations can have multiple servers at the frontend and the backend like a web server or an application server for hosting a website or an application, respectively

Why do we need Kafka?

Problems faced without Kafka

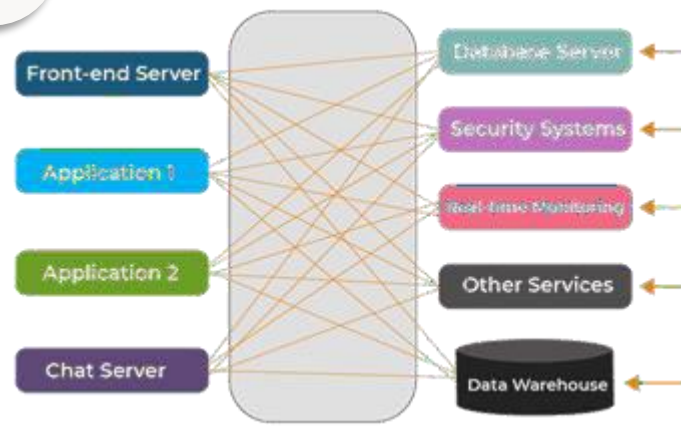
- The current-day industry is emerging with lots of real-time data that needs to be processed in real time. For example:
 - Sensor data that is used to predict the failure of a system ahead of time
 - Real-time economic data that is based on preliminary estimates and is frequently adjusted for better estimates to be available
- Organizations can have multiple servers at the frontend and the backend like a web server or an application server for hosting a website or an application, respectively
- Now, all of these servers need to communicate with database servers. Thus, we will have multiple data pipelines connecting all of them to the database servers

Why do we need Kafka?



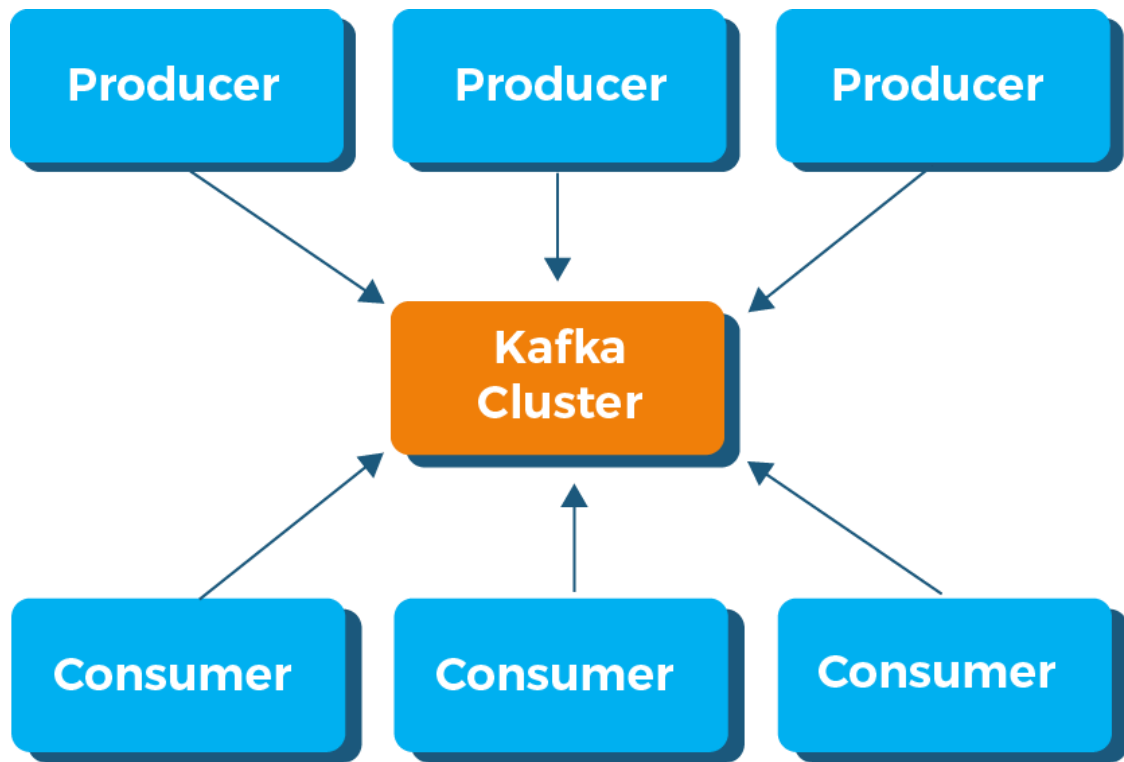
Why do we need Kafka?

- You can see that data pipelines are getting complex with the increase in the number of systems
- Adding a new system or server requires more data pipelines, which makes the data flow more complicated
- Managing these data pipelines becomes very difficult as each data pipeline has its own set of requirements
- Adding some pipelines or removing some is difficult in such cases



How does **Kafka**
solve the problem?

Kafka Decouples Data Pipelines



What is **Kafka**?

Apache Kafka: Introduction

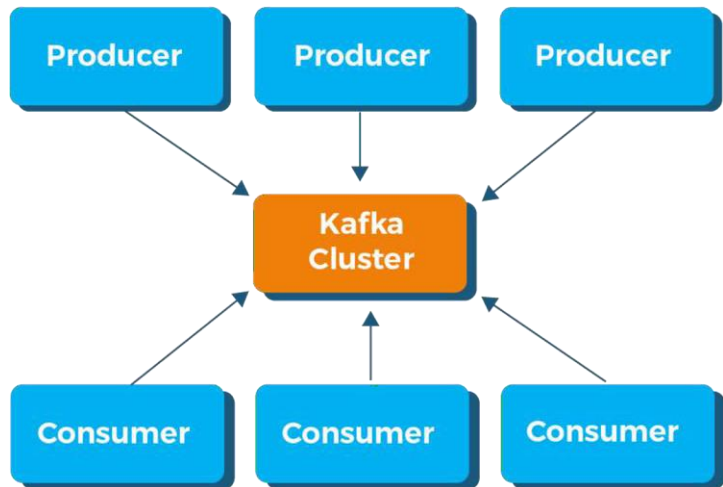
Apache Kafka is an open-source, distributed, publish—subscribe messaging system that manages and maintains the real-time stream of data from different applications, websites, etc.

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

Apache Kafka

A high-through distributed management system

Solution Provided by Kafka



- 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

Components of **Kafka**

Components of Kafka

Brokers

Kafka brokers are servers that manage and mediate conversation between two different systems. They are responsible for delivering messages to the right party

Messages

Messages are simply byte arrays; in them, any object can be stored in any format by developers. The format can be a String, JSON, Avro, and much more

Topics

In Kafka, all messages are maintained in what we call topics, i.e., messages are stored, published, and organized in Kafka topics

Clusters

In Apache Kafka, more than one broker, i.e., a set of servers, is collectively called 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 the data or messages to one or more topics. Producers are basically the source of data stream in Kafka

Consumers

Consumers are the processes that read and process data from topics by subscribing to one or more topics in the Kafka cluster

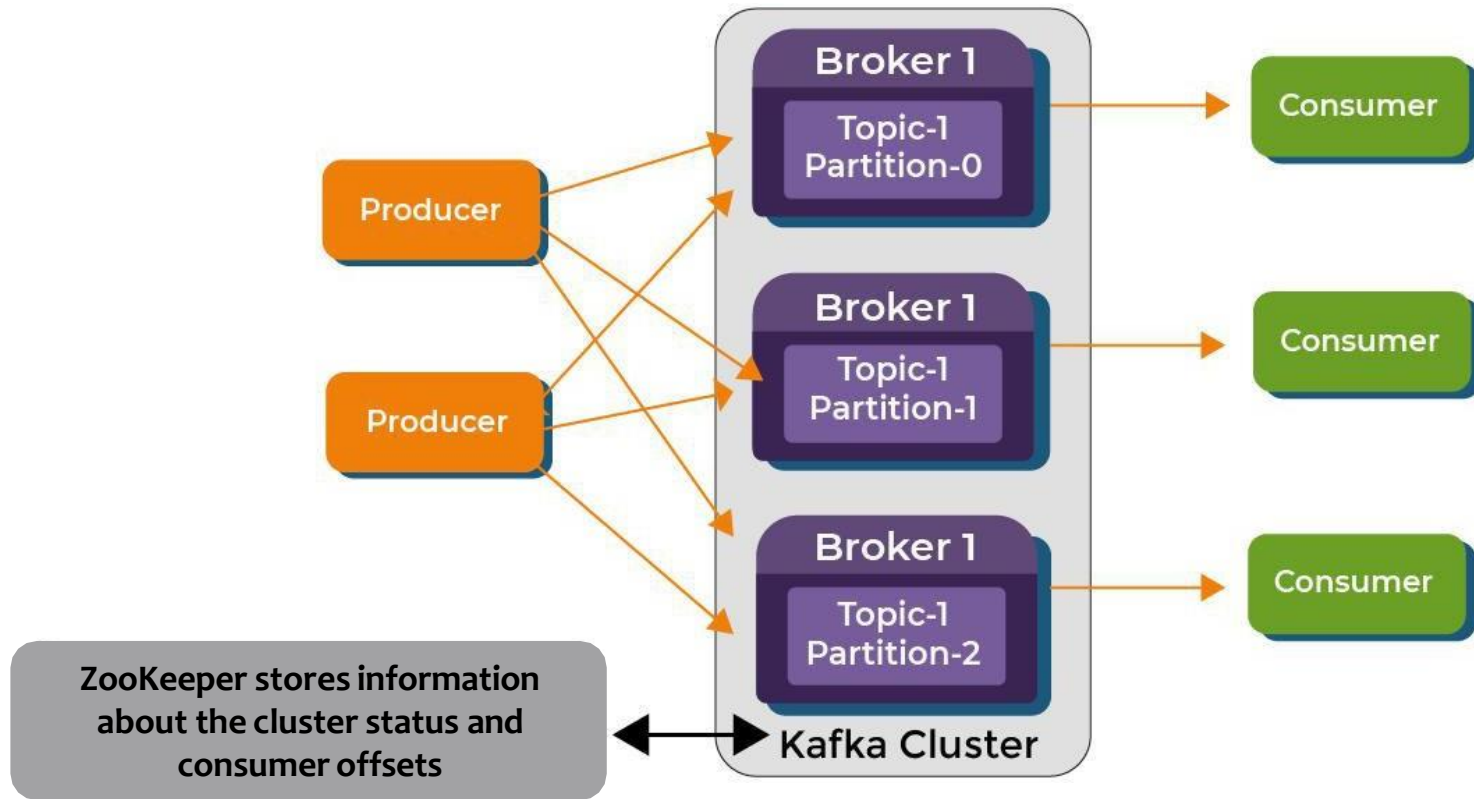
Partitions

Every broker holds a few partitions; each partition can be either a leader or a replica of a topic. All 'writes' and 'reads' to a topic go via the leader, who is responsible for updating replicas with the new data. If the leader fails, the replica takes over as the new leader

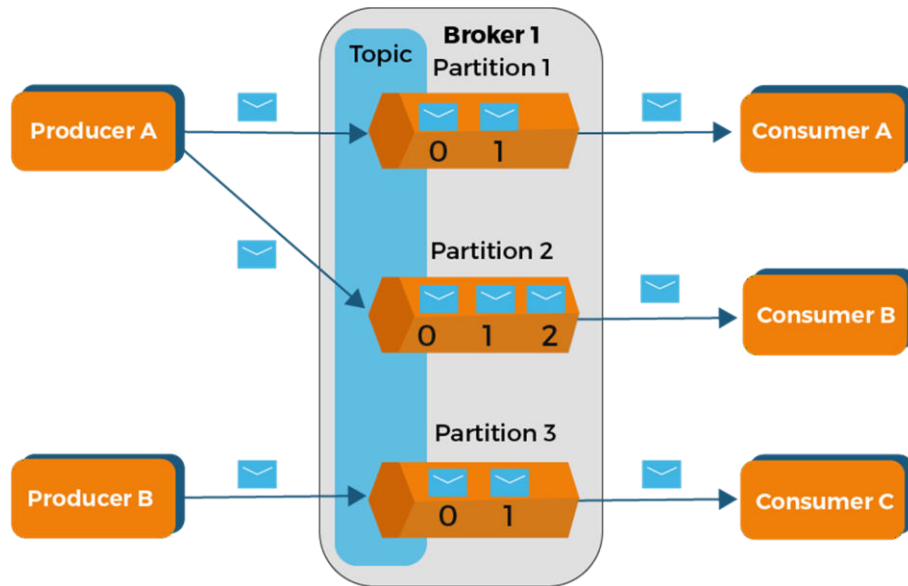


Architecture of **Kafka Cluster**

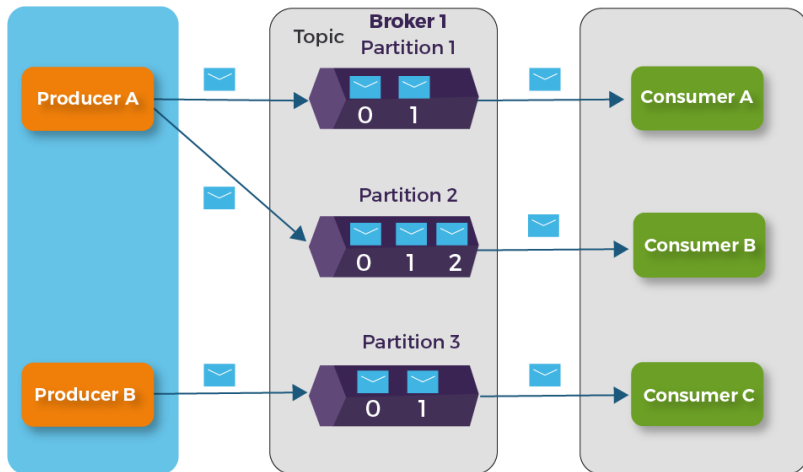
Architecture of Kafka Cluster



Architecture: The Inside View

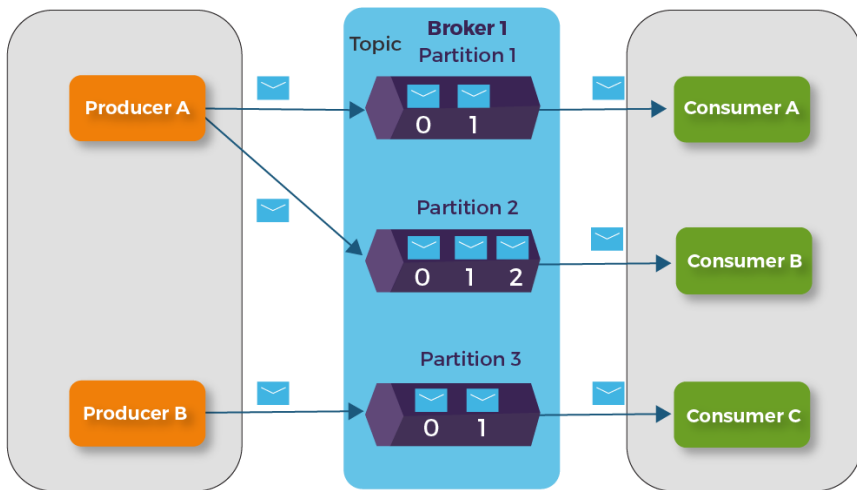


Kafka Producer



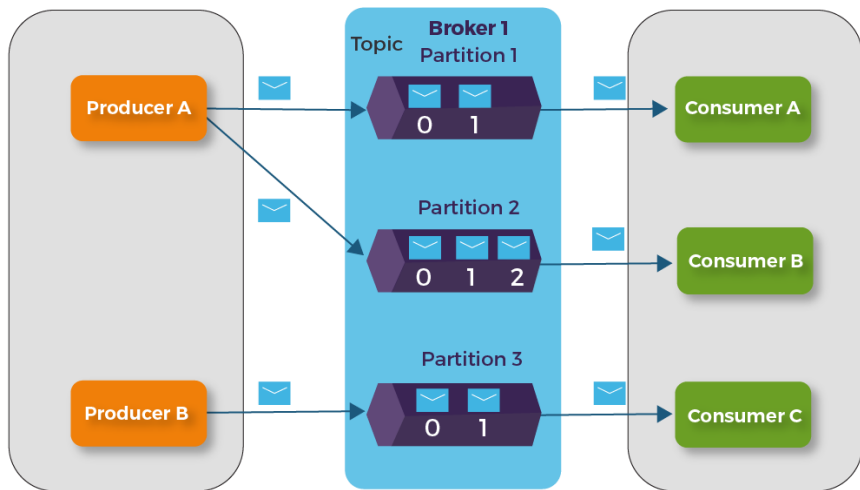
- Producers send records (also referred to as messages) to topics
- Producers select the partition to send the messages to, per topic
- Producers can implement a priority system, i.e., sending a record to certain partition depending on the priority of the record
- Producers don't wait for acknowledgements from brokers and send messages as fast as brokers can handle

Kafka Broker



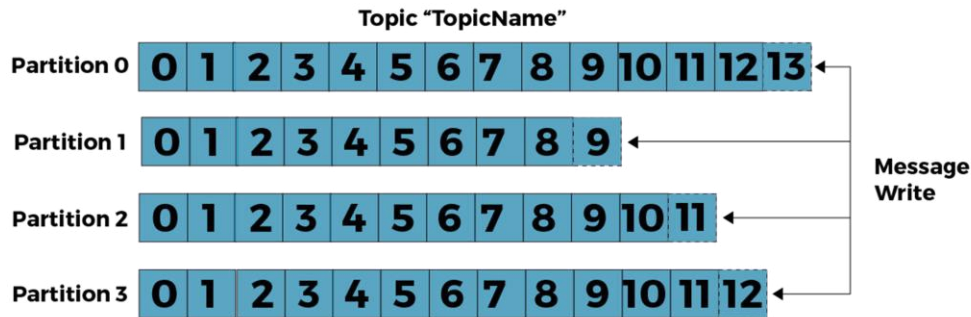
- A cluster typically consists of multiple brokers to maintain the load balance
- A broker on receiving messages from a producer assigns offsets to them and commits the messages for storage on the disk
- It serves consumers by responding to fetch requests for partitions
- One broker instance can handle thousands of reads–writes per second and terabytes 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 them in an append-only fashion
- Reading messages can be done in the order from beginning to end or by skipping or rewinding to any point in the partition by providing an offset value
- An offset value is a sequential ID provided to messages
- Partitions provide redundancy and scalability
- Partitions can be hosted on different servers, i.e., a single topic can be scaled horizontally across multiple servers, thus enhancing the performance

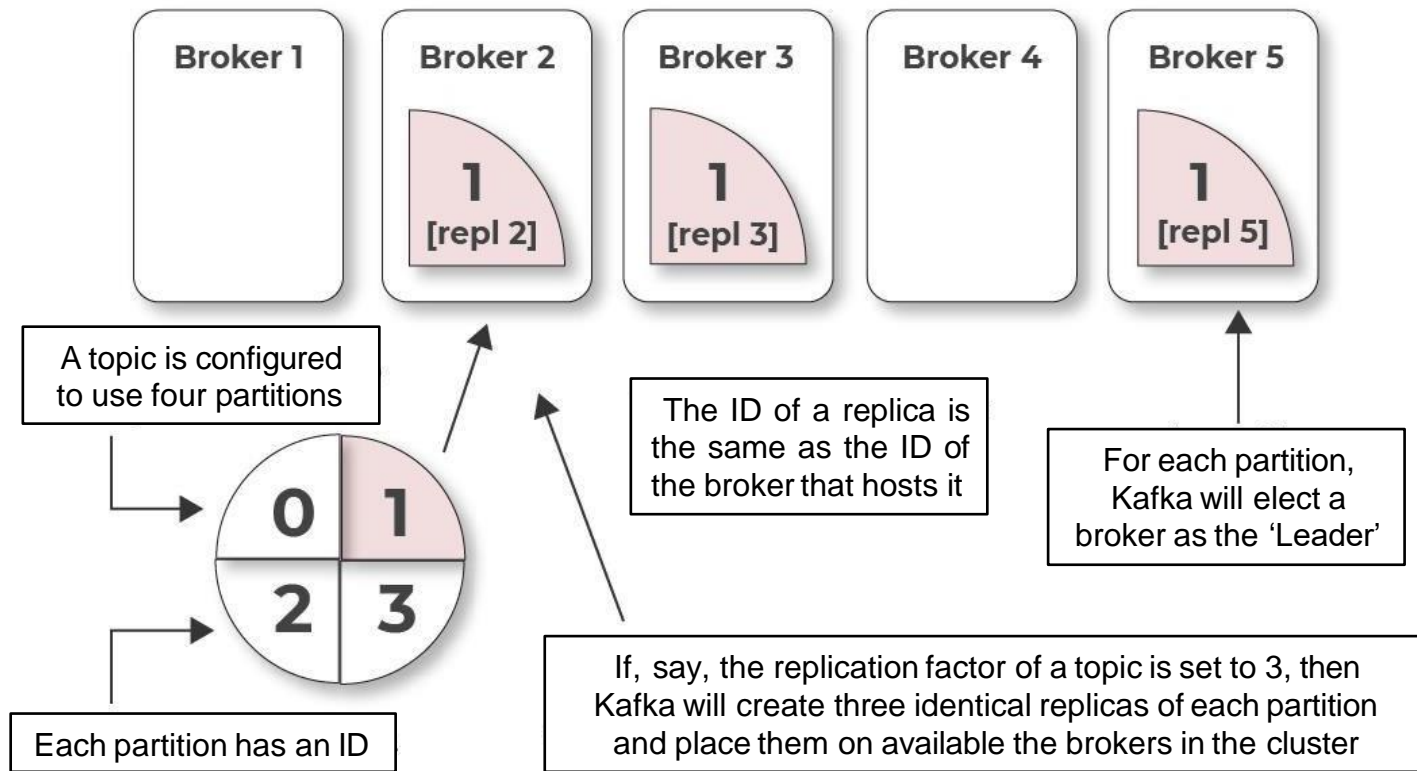
Kafka Topics and Partitions



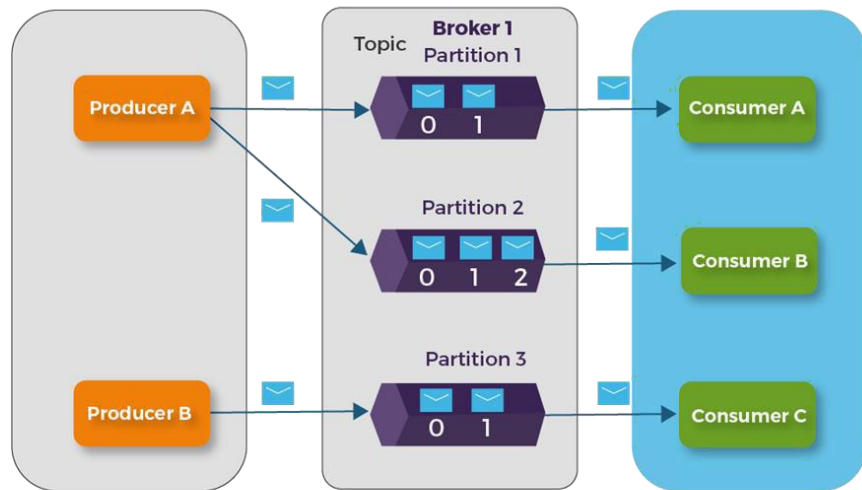
The Representation of a Topic with Multiple Partitions

- 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 (default behavior)

Kafka Topics and Partitions

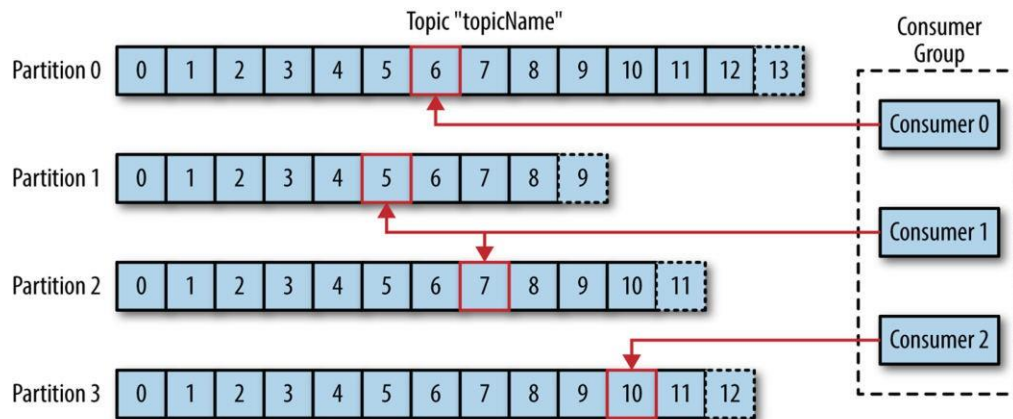


Kafka Consumer



- A 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 track of the offset of these messages
- Consumers work as part of a consumer group, which is 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

- The figure shows 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

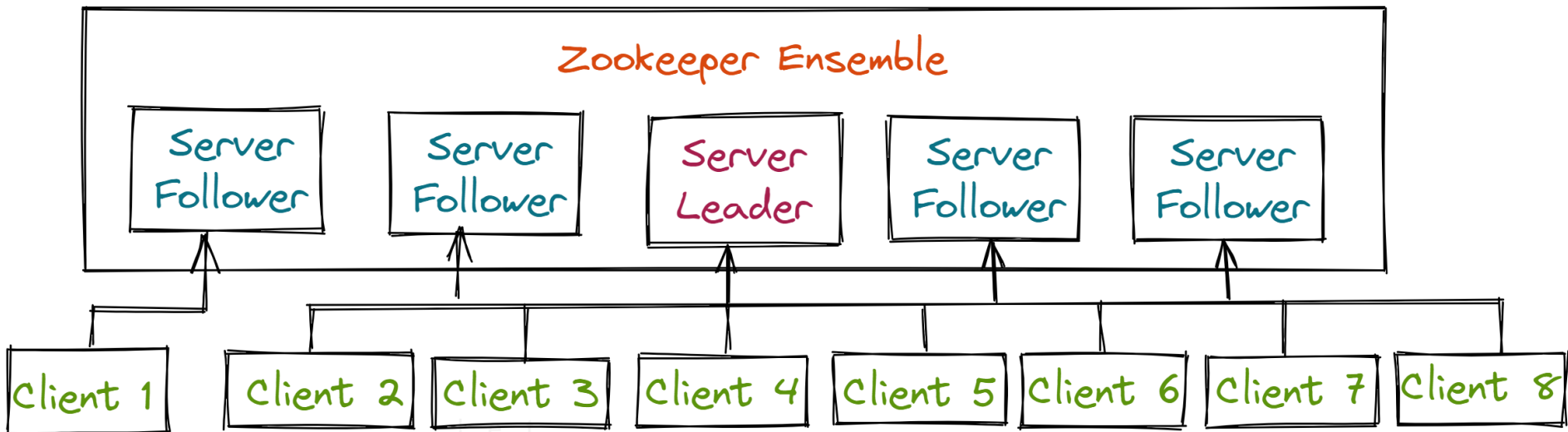
- Developed originally at Yahoo!, ZooKeeper facilitates synchronization in the process by maintaining a status on its servers that store information in local log files
- The ZooKeeper servers are capable of supporting a large Hadoop cluster



Apache
Zookeeper

Architecture of ZooKeeper

ZooKeeper Service Zookeeper Architecture



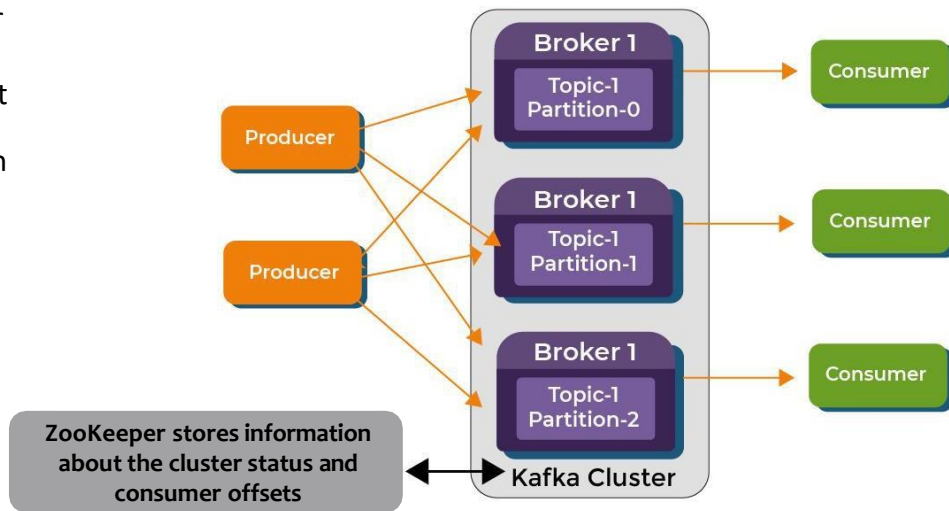
Architecture of ZooKeeper

Following constituents form the architecture of ZooKeeper

Apache ZooKeeper Basics	Tasks Performed
Server Applications	Facilitate interaction with client applications through a common interface
Client Applications	Consist of tools that interact with distributed applications
ZooKeeper Nodes	Systems on which a cluster runs
Znode	Can be updated and/or modified by any node in the cluster

ZooKeeper and Kafka

- Kafka brokers coordinate with each other using ZooKeeper
- Producers and consumers are notified by ZooKeeper about the presence of a new broker or about the failure of one 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

Producers will send messages to a topic at regular intervals

Brokers store the messages in partitions configured for that particular topic

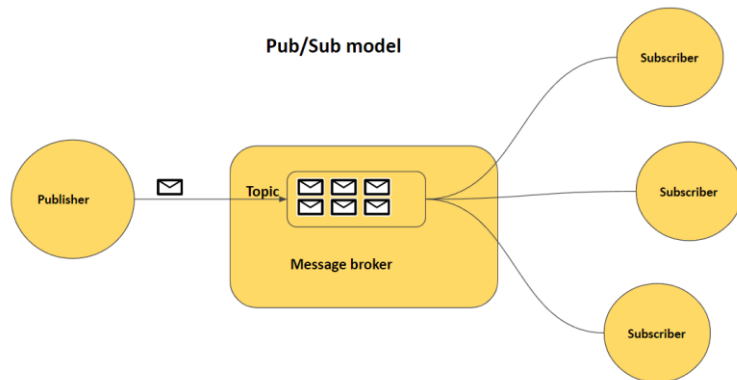
If a producer sends two messages and there is two partitions, Kafka will store one message in the first partition and the second message in the second one

A consumer always subscribes to a specific topic

When a 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 (e.g., in every 100 ms)

Publish/Subscribe Pattern



Workflow of Pub/Sub Messaging

As soon as a message is received from the producer, it is forwarded to consumers

On receiving the message, consumers will process it

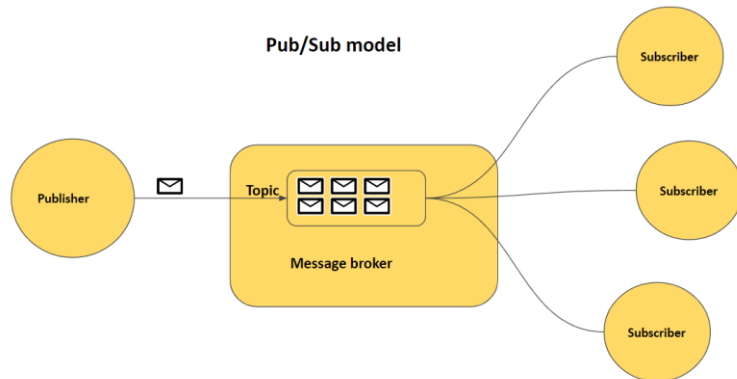
Once the message is processed, an acknowledgement is sent to the broker

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 outages, since the offsets are maintained in ZooKeeper

The flow repeats until the consumer stops the request

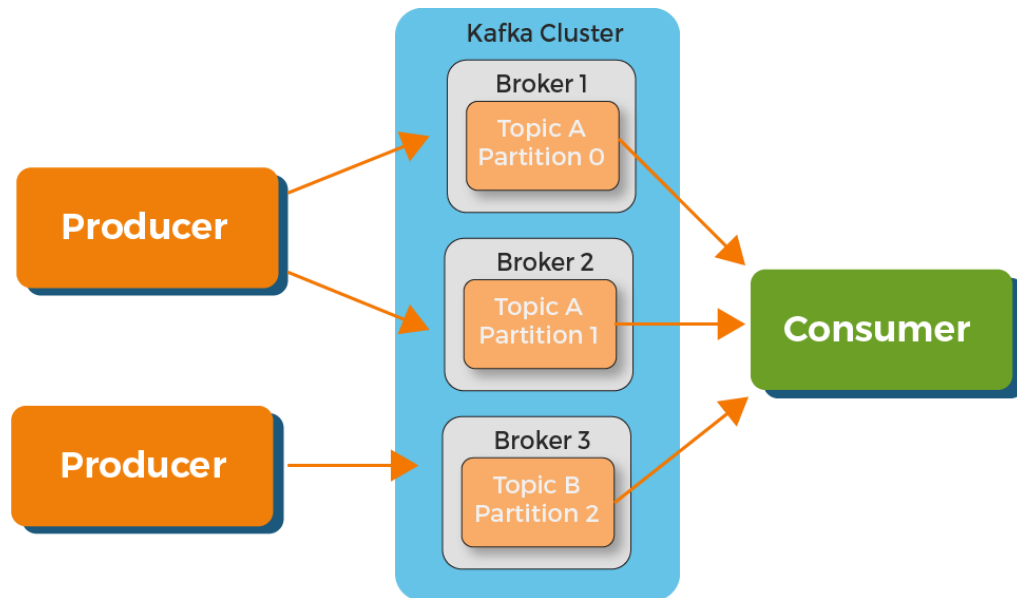
Publish/Subscribe Pattern



Kafka Clusters

Kafka Clusters

- A single Kafka server works well for the local development environment
- 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
- Kafka clusters are effective for applications that involve large-scale message processing



Kafka Command-line Tools

- Kafka clusters can run against the following broker set-ups:
 - Single broker cluster
 - Multiple brokers 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	It is the command-line client to send messages to the Kafka cluster
kafka-console-consumer.sh	It is the command-line client to consume messages from the Kafka cluster

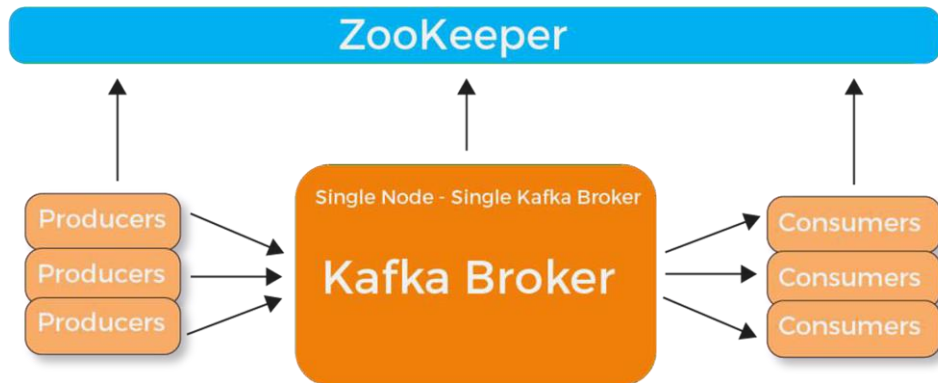
Types of Kafka Clusters

Types of Kafka Cluster

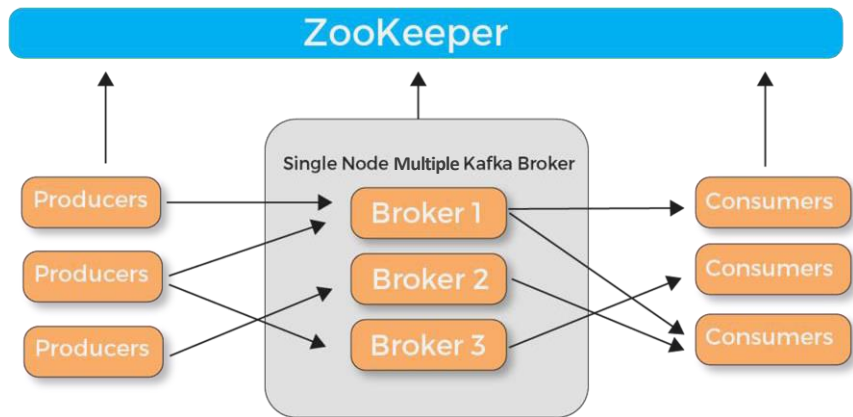
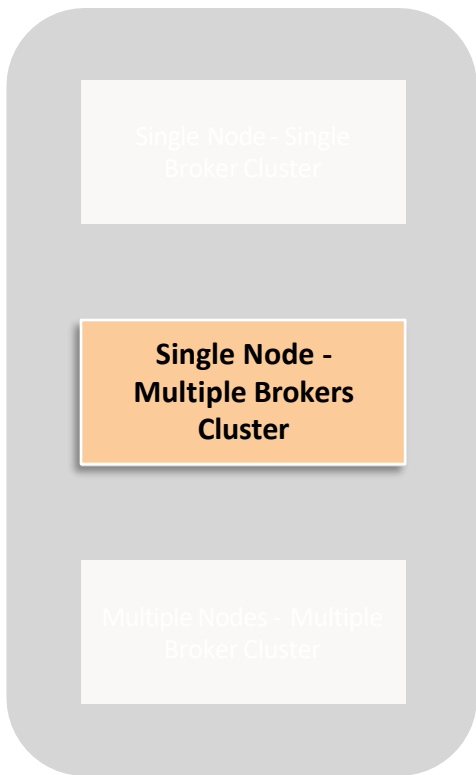
**Single Node - Single
Broker Cluster**

Single Node - Multiple
Broker Cluster

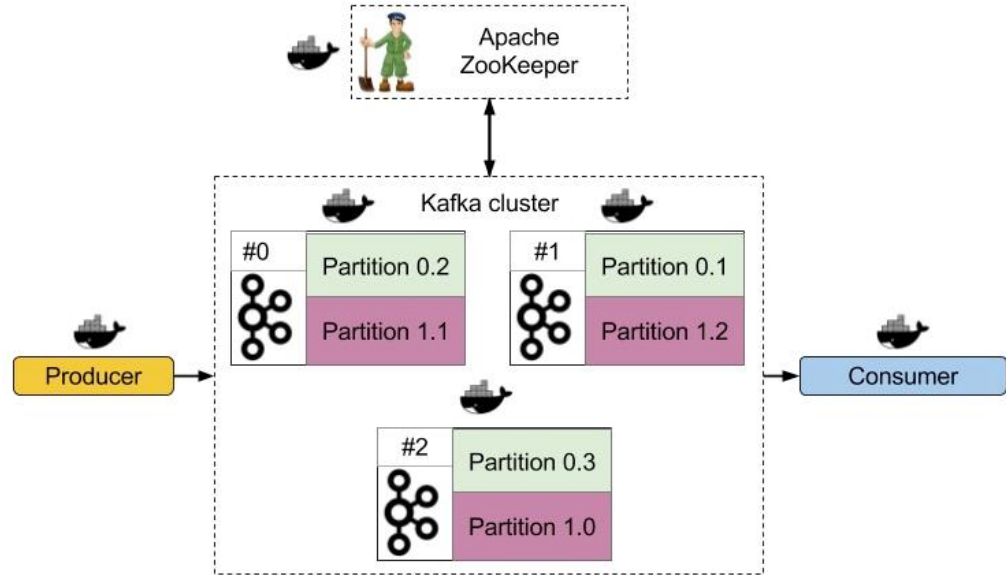
Multiple Nodes - Multiple
Broker Cluster



Types of Kafka Cluster



Types of Kafka Cluster



Multiple Nodes - Multiple
Brokers Cluster

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  
ookeeper.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 `ZooKeeper daemon` and the other one is `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

- Create 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 example
```

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 localhost:9092 --topic Example1 --from-beginning  
hello  
this is my first example
```

Configuring Single Node - Multiple Brokers Cluster

Multiple Brokers Setup

- Open your terminal and start ZooKeeper, after which start the 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  
ookeeper.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)
```

Multiple Brokers 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
```

Multiple Brokers Setup

Open server1.properties file and make the following changes:

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

Open server2.properties file and make the following changes:

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

```
broker.id=1

@
THOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and defaults

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

```
broker.id=2

@
THOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the license.

# see kafka.server.KafkaConfig for additional details and defaults
```

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

Multiple Brokers Setup

Start Multiple Kafka Brokers

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

```
[training@ip-172-31-27-203 ~]$ kafka-server-start.sh kafka/config/server1.properties  
[2018-11-23 08:45:19,379] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$  
)  
[2018-11-23 08:45:19,770] INFO starting (kafka.server.KafkaServer)  
[2018-11-23 08:45:19,771] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)  
[2018-11-23 08:45:19,790] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
```

```
[training@ip-172-31-27-203 ~]$ kafka-server-start.sh kafka/config/server2.properties  
[2018-11-23 08:46:19,144] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$  
trollerRegistration$)  
[2018-11-23 08:46:19,543] INFO starting (kafka.server.KafkaServer)  
[2018-11-23 08:46:19,544] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)  
[2018-11-23 08:46:19,563] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
```

Multiple Brokers 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 --replication-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
```

Multiple Brokers Setup

- Start the producer to send messages

```
kafka-console-producer.sh --broker-list localhost:9093,localhost:9094,localhost:9092 --topic EKample2
```

```
[training@ip-172-31-27-203 ~]$ kafka-console-producer.sh --broker-list localhost:9093,localhost:9094,localhost:9092 --topic EKample2
c Example2
>Bonjour
>Do you understand kafka now?
>Hope you liked the session !!
```

Multiple Brokers 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 Example2 --from-beginning
```

```
Bonjour
```

```
Do you understand kafka now?
```

```
Hope you liked the session !!
```

Basic Topic Operations

Basic Topic Operations

Modifying
a Topic

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

Deleting a
Topic

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --zookeeper localhost:2181 --alter  
--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!
```

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
```

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

Deleting a
Topic

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --zookeeper localhost:2181 --delete --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

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

Deleting a
Topic

Kafka

Performance Turning

Kafka Performance Tuning

- Performance tuning involves two important metrics:
 - **Latency**: How long it takes to process one event
 - **Throughput**: How many events arrive within a specific amount of time
- Kafka is optimized for both
- Kafka is well tuned if it has just enough brokers to handle the topic throughput with the latency required to process information as it arrives
- The best balance between latency and throughput for a Kafka cluster is achieved by:
 - Tuning producers, brokers, and consumers to send, process, and receive the largest possible batches within a manageable amount of time



Tuning Brokers

- Topics are divided into partitions
- Each partition has a leader
- Standard configuration for reliability:
 - A topic will consist of a leader partition and two or more follower partitions
 - If the leaders are not balanced properly, one might be overworked, compared to others
 - Have enough replication sets to preserve your data
 - For each topic, it is recommended to start with one partition per physical storage disk and one consumer per partition



Tuning Producers

- Two parameters are particularly important for latency and throughput: **batch size** and **linger time**
- Batch size:
 - **batch.size** measures the batch size in total bytes
 - It controls the number of bytes of data to collect before sending messages to the broker
 - Set this value as high as possible, but do not exceed the available memory
 - The default value is 16384



Tuning Producers

- Linger time:
 - **linger.ms** sets the maximum time to buffer data in the asynchronous mode
 - For example, setting it to 100 means that it batches 100 ms of messages to send at once
 - This improves throughput, but buffering adds message delivery latency



Tuning Consumers

- The maximum number of consumers in a consumer group for a topic is equivalent to the number of partitions
- You will need enough partitions to handle all consumers required to keep up with producers
- Consumers in the same consumer group split the partitions among them
- Adding more consumers to a group can enhance performance
- Adding consumer groups does not affect performance



Mirroring Data Between Kafka Clusters

- Mirroring increases the throughput
- It is recommended to use in-cluster replication in order to obtain more fault-tolerance

Example: Mirroring a single topic (named **my-topic**) from an input cluster

```
bin/kafka-mirror-maker.sh  
  --consumer.config consumer.properties  
  --producer.config producer.properties --whitelist my-topic
```



Apache Kafka Applications

Kafka Applications



Twitter uses [Storm-Kafka](#)
as part of its stream
processing infrastructure

Kafka Applications



Twitter uses [Storm-Kafka](#)
as part of its stream
processing infrastructure



Netflix uses Kafka for [real-time monitoring](#) and [event processing](#)

Kafka Applications



Twitter uses [Storm—Kafka](#) as part of its stream processing infrastructure



Netflix uses Kafka for [real-time monitoring](#) and [event processing](#)



It is used at LinkedIn for [activity stream data](#) and for [operational metrics](#)

Kafka Applications



Twitter uses [Storm-Kafka](#) as part of its stream processing infrastructure



Netflix uses Kafka for [real-time monitoring](#) and [event processing](#)



It is used at LinkedIn for [activity stream data](#) and for [operational metrics](#)

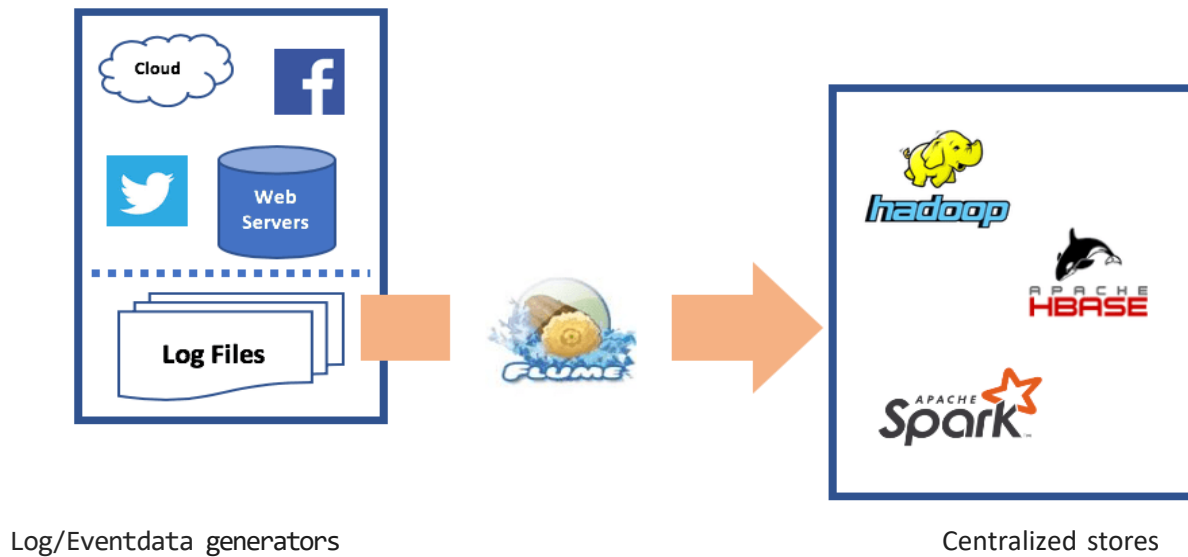


Oracle provides [native connectivity](#) to Kafka from its Enterprise Service Bus product, [OSB](#) (Oracle Service Bus)

Introduction to **Apache Flume**

What is Apache Flume?

Apache Flume is an open-source tool, reliable and highly available for aggregating, collecting, and moving large amounts of data from many different sources into a centralized data storage



Applications of Flume



Flume ingests log data from multiple web servers into a centralized store (HDFS, Hbase, etc.) efficiently

Flume supports a large set of sources and destinations types

It supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.

Apache Flume can be scaled, horizontally

Features of Flume

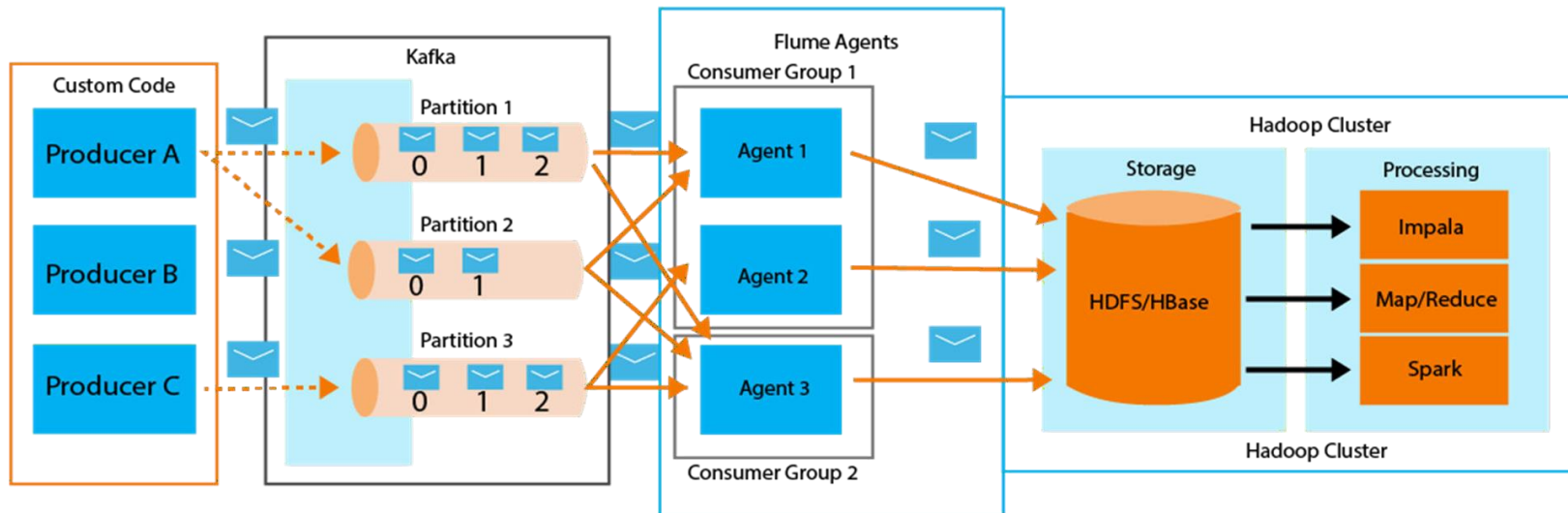
Assume that an e-commerce web application wants to analyze its customer behavior from a particular region. To do so, it would need to move the available log data into Hadoop for analysis. Here, Apache Flume comes to its rescue

Flume can be used to move the log data generated by application servers into HDFS at a higher speed

The new integration between Flume and Kafka offers the sub-second-latency event processing without the need for dedicated infrastructure. Let us see how!

How does **Flume**
integrate with **Kafka**?

Kafka-Flume Integration



Using **Kafka** as a Source for **Flume**

Using Kafka as a Source for Flume

- You will pass a message to a Kafka producer, which will go through the Flume channel (in-memory) and get stored in the Flume sink (HDFS)

```
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  
ookeeper.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)
```

Using Kafka as a Source for Flume

- Create a Kafka topic

```
kafka-topics.sh --create --topic flume --bootstrap-server localhost:9092 --  
partitions 1 --replication-factor 1
```

```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --create --zookeeper localhost:21  
81 --replication-factor 1 --partitions 1 --topic Flume_kafka_topic1  
WARNING: Due to limitations in metric names, topics with a period ('.') or under  
score ('_') could collide. To avoid issues it is best to use either, but not bot  
h.  
Created topic "Flume_kafka_topic1".
```

- List of topics


```
[training@ip-172-31-27-203 ~]$ kafka-topics.sh --list --zookeeper localhost:2181  
Example1  
Example2  
Flume_kafka_topic  
Flume_kafka_topic1  
__consumer_offsets  
my-kafka-topic
```


Using Kafka as a Source for Flume

- Setup a Flume configuration file

```
cd flume
cd conf
vi excercise6.conf
```

```
[training@ip-172-31-27-203 conf]$ vi excercise6.conf
```



Using Kafka as a Source for Flume

- Contents of the Flume configuration file

```
## Configuring Components
a1.sources=source1
a1.channels=channel1
a1.sinks=sink1
## Configuring Source
a1.sources.source1.type=org.apache.flume.source.kafka.KafkaSource
a1.sources.source1.zookeeperConnect=localhost:2181
a1.sources.source1.topic=flume_kafka_topic1
a1.sources.source1.groupId=flume
a1.sources.source1.channels=channel1
a1.sources.source1.interceptors=i1
a1.sources.source1.interceptors.i1.type=timestamp
a1.sources.source1.kafka.consumer.timeout.ms=100
## Configuring Channel
a1.channels.channel1.type=memory
a1.channels.channel1.capacity=10000
a1.channels.channel1.transactionCapacity=1000
```

Using Kafka as a Source for Flume

- Start a Flume agent

```
flume-ng agent --conf conf --conf-file exercise6.conf --name a1
```

```
[training@ip-172-31-27-203 conf]$ flume-ng agent --conf conf --conf-file exercise6.conf --name a1
Info: Including Hadoop libraries found via (/home/training/hadoop/bin/hadoop) for HDFS access
Info: Excluding /home/training/hadoop/share/hadoop/common/lib/slf4j-api-1.7.10.jar from classpath
Info: Excluding /home/training/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar from classpath
Info: Including HBASE libraries found via (/home/training/hbase/bin/hbase) for HBASE access
Info: Excluding /home/training/hbase/lib/slf4j-api-1.7.25.jar from classpath
Info: Excluding /home/training/hbase/lib/slf4j-log4j12-1.7.25.jar from classpath
Info: Excluding /home/training/hadoop/share/hadoop/common/lib/slf4j-api-1.7.10.jar from classpath
Info: Excluding /home/training/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar from classpath
Info: Including Hive libraries found via (/home/training/hive) for Hive access
```

Using Kafka as a Source for Flume

- Pass your messages from the Kafka producer

```
kafka-conso  -gproducer.sh --broker-list  oqplhost:9092 --topic Flume_kafka_topic1
```

```
[training@ip-172-31-27-203 ~]$ kafka-console-producer.sh --broker-list localhost:9092 --topic Flume_kafka_topic1
>sfzsd
>s
>ad
```

Using Kafka as a Source for Flume

- Store those messages in the sink (HDFS)

```
hdfs dfs -ls /tmp/kafkatest
```

```
[training@ip-172-31-27-203 ~]$ hadoop fs -ls /tmp/kafkatest
18/11/23 18:20:24 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 5 items
-rw-r--r--    1 training supergroup          34 2018-11-23 17:37 /tmp/kafkatest/FlumeData.1542994665885
-rw-r--r--    1 training supergroup          12 2018-11-23 17:37 /tmp/kafkatest/FlumeData.1542994665886
-rw-r--r--    1 training supergroup          33 2018-11-23 17:38 /tmp/kafkatest/FlumeData.1542994686324
-rw-r--r--    1 training supergroup          34 2018-11-23 17:38 /tmp/kafkatest/FlumeData.1542994686325
-rw-r--r--    1 training supergroup           5 2018-11-23 17:38 /tmp/kafkatest/FlumeData.1542994686326
```

Using Kafka as a Source for Flume

- Store those messages in the sink (HDFS)

```
hadoop fs -cat /tmp/kafkatest/<file name>
```

```
[training@ip-172-31-27-203 ~]$ hadoop fs -cat /tmp/kafkatest/FlumeData.154299466  
5885  
18/11/23 18:12:11 WARN util.NativeCodeLoader: Unable to load native-hadoop libra  
ry for your platform... using builtin-java classes where applicable  
sfzsd  
s  
ad
```

Quiz!

Question #1

At which firm did Kafka originate before it became open-sourced?

A LinkedIn

B Facebook

C Google

D Yahoo!

Answer #1

At which firm did Kafka originate before it became open-sourced?

A

LinkedIn



B

Facebook

C

Google

D

Yahoo!

Question #2

Kafka makes data pipelines more complex.

A

True

B

False

Answer #2

Kafka makes data pipelines more complex.

A

True

B

False



Question #3

A Kafka cluster is effective for large-scale message processing.

A

True

B

False

Answer #3

A Kafka cluster is effective for large-scale message processing.

A

True



B

False



+91-9030485102



rganesh0203@gmail.com



https://topmate.io/rganesh_0203