Programming for Big Data

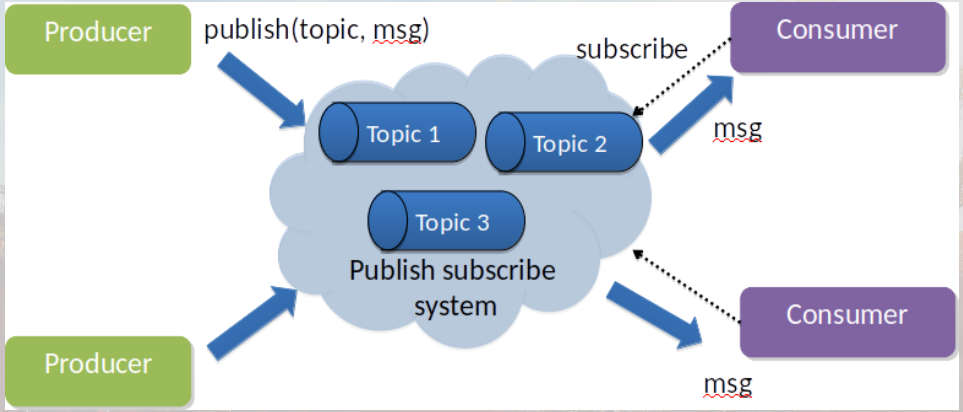
Apache Kafka and Flume

Saeed Iqbal Khattak

Centre for Healthcare Modelling & Informatics
Faculty of Information Technology,
University of Central Punjab, Lahore

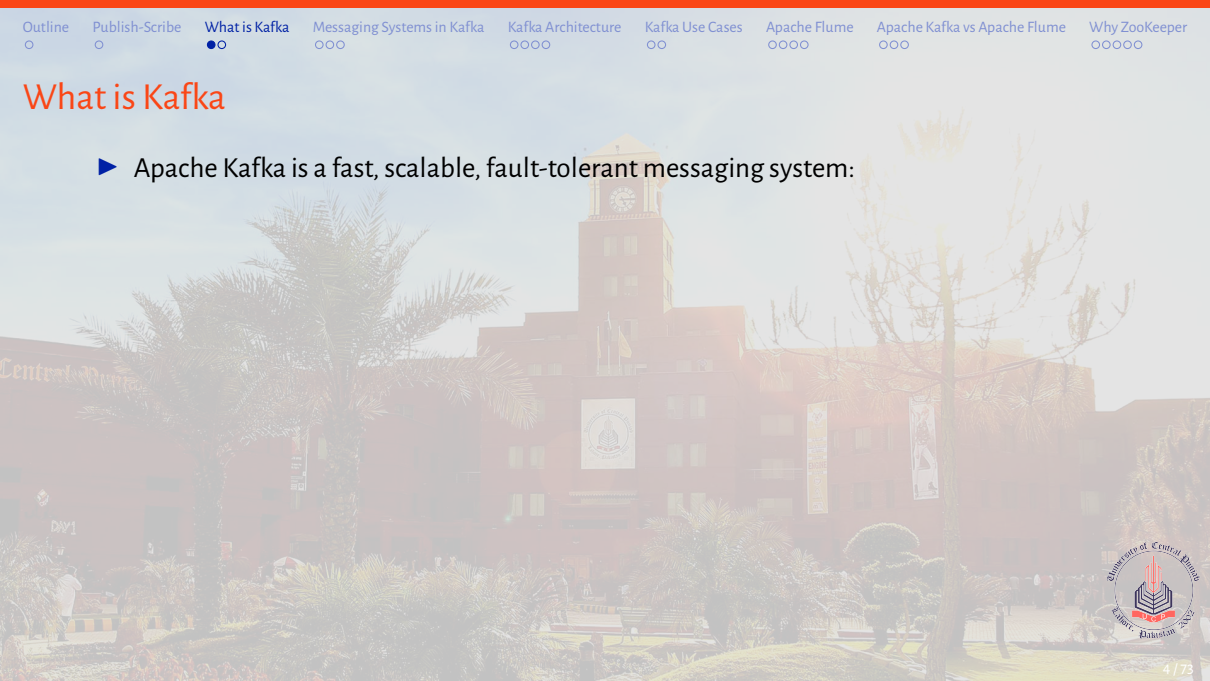
June 29, 2021

Outline


Producer
publish(topic, msg)
subscribe
Consumer
Topic 1
Topic 2
msg
Topic 3
Publish subscribe system
Producer
Consumer
msg

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:



What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.
  3. it allows a large number of permanent or ad-hoc consumers.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.
  3. it allows a large number of permanent or ad-hoc consumers.
  4. it is highly available and resilient to node failures and supports automatic recovery.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.
  3. it allows a large number of permanent or ad-hoc consumers.
  4. it is highly available and resilient to node failures and supports automatic recovery.
- Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.
  3. it allows a large number of permanent or ad-hoc consumers.
  4. it is highly available and resilient to node failures and supports automatic recovery.
- Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time.
- Streaming data is data that is continuously generated by thousands of data sources, which typically send the data records in simultaneously.

What is Kafka

- Apache Kafka is a fast, scalable, fault-tolerant messaging system:
  1. enables communication between producers and consumers using message-based topics.
  2. it designs a platform for high-end new generation distributed applications.
  3. it allows a large number of permanent or ad-hoc consumers.
  4. it is highly available and resilient to node failures and supports automatic recovery.
- Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time.
- Streaming data is data that is continuously generated by thousands of data sources, which typically send the data records in simultaneously.
- Kafka provides three main functions to its users:
  1. Publish and subscribe to streams of records.
  2. Effectively store streams of records in the order in which records were generated.
  3. Process streams of records in real time.

- Before moving deep into the Kafka, you must aware of the main terminologies:

Message: In Kafka often we consider data as a set of messages. A message is a simple array of bytes, e.g. csv file.

Producer: Producer is an application that sends messages. It does not send messages directly to the recipient. It send messages only to the Kafka server.

Consumer: It is an application that reads messages from the Kafka server. (i.e. consumers are the recipients.) Consumers should have the permission to read the messages.

Broker: The broker is a Kafka server. One can say that all Kafka does is act as a message broker between producer and consumer, because producer and consumer do not connect directly.

Cluster: Kafka is a distributed system, it act as a cluster. That is, a group of computers sharing workload for common purpose. Each instance contains a Kafka broker.

Topics: A stream of messages belonging to a particular category is called a topic. Data is stored in topics.

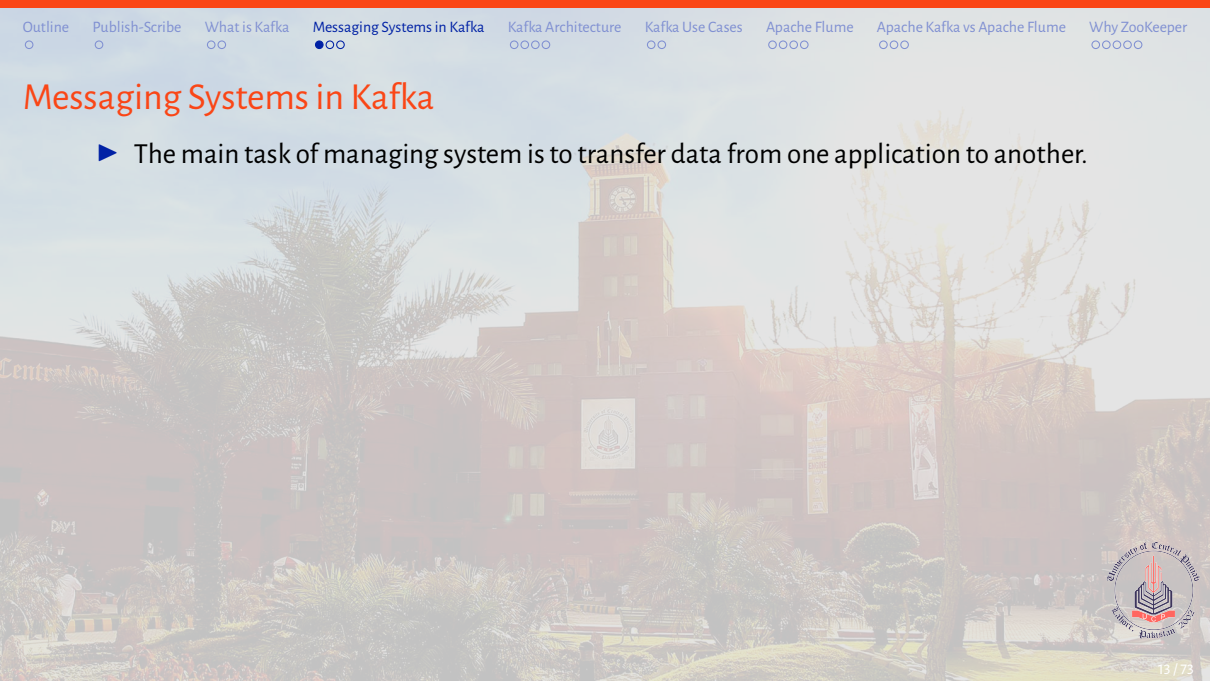
Partitions: Kafka Brokers will store messages for a topic. But the capacity of data can be enormous and it may not be possible to store in a single computer.

Offsets: Offset is a sequence of ids given to messages as the arrive at a partition. Once the offset is assigned it will never be changed.

Zookeeper: Zookeeper serves as the coordination interface between the Kafka brokers and consumers.

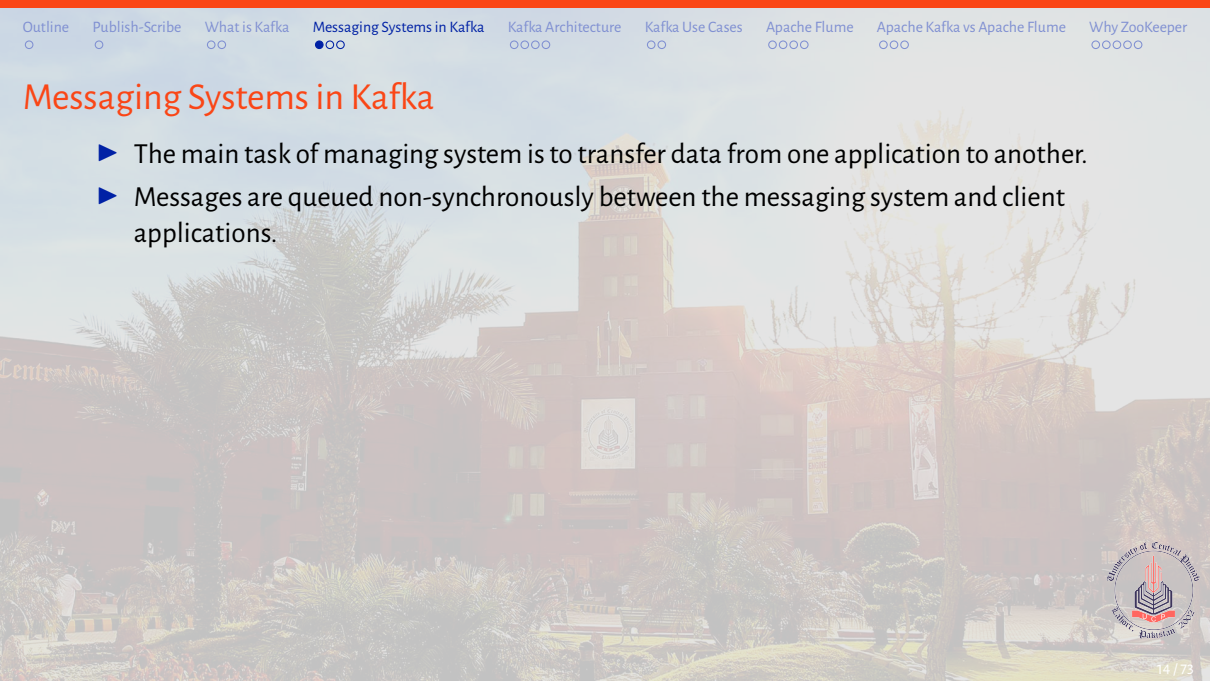
Messaging Systems in Kafka

- The main task of managing system is to transfer data from one application to another.



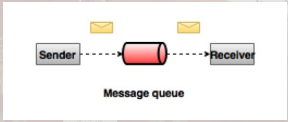
Messaging Systems in Kafka

- The main task of managing system is to transfer data from one application to another.
- Messages are queued non-synchronously between the messaging system and client applications.

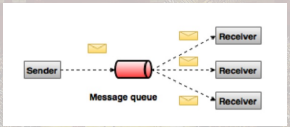


Messaging Systems in Kafka

- The main task of managing system is to transfer data from one application to another.
- Messages are queued non-synchronously between the messaging system and client applications.
- There are two types of messaging patterns available:
  1. Point to point messaging system.



  2. Publish-subscribe messaging system



Point to point messaging system

- More than one sender can produce and send messages to a queue. Senders can share a connection or use different connections, but they can all access the same queue.

Point to point messaging system

- More than one sender can produce and send messages to a queue. Senders can share a connection or use different connections, but they can all access the same queue.
- More than one receiver can consume messages from a queue, but each message can be consumed by only one receiver. Thus, Message 1, Message 2, and Message 3 are consumed by different receivers. (This is a message queue extension.)

Point to point messaging system

- More than one sender can produce and send messages to a queue. Senders can share a connection or use different connections, but they can all access the same queue.
- More than one receiver can consume messages from a queue, but each message can be consumed by only one receiver. Thus, Message 1, Message 2, and Message 3 are consumed by different receivers. (This is a message queue extension.)
- Senders and receivers have no timing dependencies; the receiver can consume a message whether or not it was running when the sender produced and sent the message.

Point to point messaging system

- More than one sender can produce and send messages to a queue. Senders can share a connection or use different connections, but they can all access the same queue.
- More than one receiver can consume messages from a queue, but each message can be consumed by only one receiver. Thus, Message 1, Message 2, and Message 3 are consumed by different receivers. (This is a message queue extension.)
- Senders and receivers have no timing dependencies; the receiver can consume a message whether or not it was running when the sender produced and sent the message.
- **The PTP messaging model can be further categorized into two types:**
  1. Fire-and-forget model
  2. Request/reply model

Publish-Subscribe Messaging System

- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.

Publish-Subscribe Messaging System

- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.
- In this messaging system, messages continue to remain in a Topic.

Publish-Subscribe Messaging System

- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.
- In this messaging system, messages continue to remain in a Topic.
- Contrary to Point to point messaging system, consumers can take more than one topic and consume every message in that topic.

Publish-Subscribe Messaging System

- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.
- In this messaging system, messages continue to remain in a Topic.
- Contrary to Point to point messaging system, consumers can take more than one topic and consume every message in that topic.
- Messages are shared through a channel called atopic. A topic is a centralized place where producers can publish, and subscribers can consume, messages.

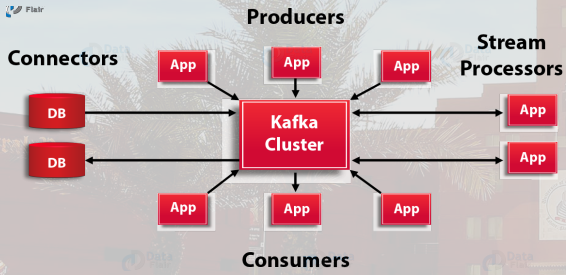
Publish-Subscribe Messaging System

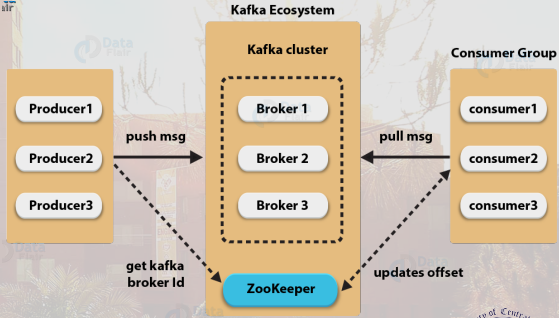
- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.
- In this messaging system, messages continue to remain in a Topic.
- Contrary to Point to point messaging system, consumers can take more than one topic and consume every message in that topic.
- Messages are shared through a channel called atopic. A topic is a centralized place where producers can publish, and subscribers can consume, messages.
- Each message is delivered to one or more message consumers, called subscribers.

Publish-Subscribe Messaging System

- A Pub/Sub messaging model is used when you need to broadcast an event or message to many message consumers.
- In this messaging system, messages continue to remain in a Topic.
- Contrary to Point to point messaging system, consumers can take more than one topic and consume every message in that topic.
- Messages are shared through a channel called atopic. A topic is a centralized place where producers can publish, and subscribers can consume, messages.
- Each message is delivered to one or more message consumers, called subscribers.
- The Publisher generally does not know and is not aware of which subscribers are receiving the topic messages.

Kafka Architecture





Kafka Architecture

- Apache Kafka Architecture has four core APIs, Producer, Consumer, Streams, and Connector API.

Producer: In order to publish a stream of records to one or more Kafka topics, the Producer API allows an application.

Consumer: This API permits an application to subscribe to one or more topics.

Streams: It consuming an input stream from one or more topics and producing an output stream to one or more output topics.

Connector: While it comes to building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems, we use the Connector API.

Kafka Architecture – Kafka Topics

- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.

Kafka Architecture – Kafka Topics

- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.
  2. Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.

Kafka Architecture – Kafka Topics

- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.
  2. Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.
  3. Basically, at first, a producer writes its messages to the topics. Then consumers read those messages from topics.

Kafka Architecture – Kafka Topics

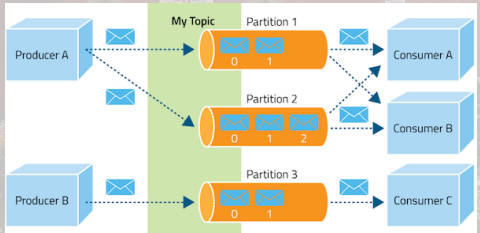
- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.
  2. Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.
  3. Basically, at first, a producer writes its messages to the topics. Then consumers read those messages from topics.
  4. In a Kafka cluster, a topic is identified by its name and must be unique.

Kafka Architecture – Kafka Topics

- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.
  2. Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.
  3. Basically, at first, a producer writes its messages to the topics. Then consumers read those messages from topics.
  4. In a Kafka cluster, a topic is identified by its name and must be unique.
  5. There can be any number of topics, there is no limitation.

Kafka Architecture – Kafka Topics

- The topic is a logical channel to which producers publish message and from which the consumers receive messages.
  1. A topic defines the stream of a particular type/classification of data, in Kafka.
  2. Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.
  3. Basically, at first, a producer writes its messages to the topics. Then consumers read those messages from topics.
  4. In a Kafka cluster, a topic is identified by its name and must be unique.
  5. There can be any number of topics, there is no limitation.
  6. We can not change or update data, as soon as it gets published.



Kafka Architecture – Kafka Partitions

- In a Kafka cluster, Topics are split into Partitions and also replicated across brokers/clusters.
  1. However, to which partition a published message will be written, there is no guarantee about that.

Kafka Architecture – Kafka Partitions

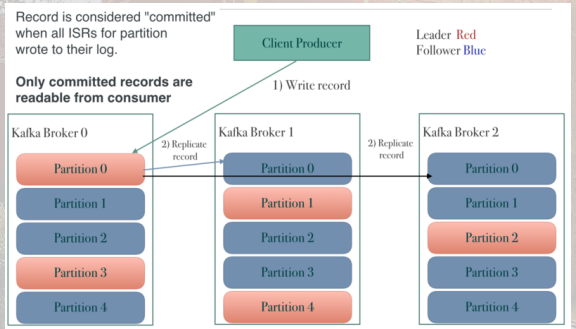
- In a Kafka cluster, Topics are split into Partitions and also replicated across brokers/clusters.
  1. However, to which partition a published message will be written, there is no guarantee about that.
  2. In one partition, messages are stored in the sequenced fashion.

Kafka Architecture – Kafka Partitions

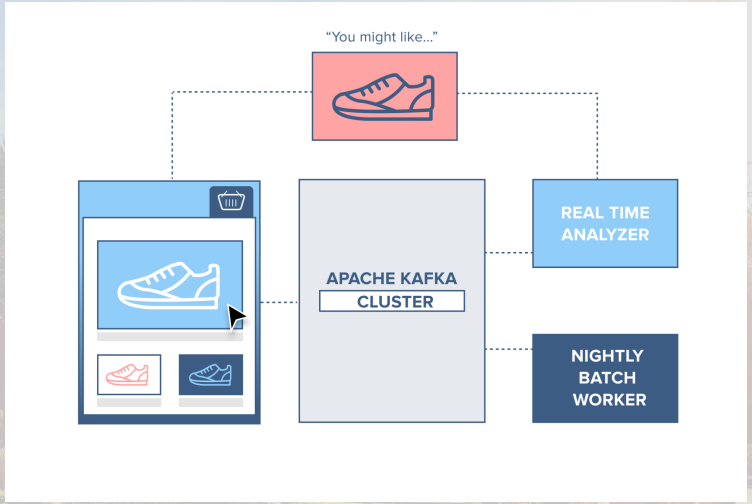
- In a Kafka cluster, Topics are split into Partitions and also replicated across brokers/clusters.
  1. However, to which partition a published message will be written, there is no guarantee about that.
  2. In one partition, messages are stored in the sequenced fashion.
  3. In a partition, each message is assigned an incremental id, also called offset.

Kafka Architecture – Kafka Partitions

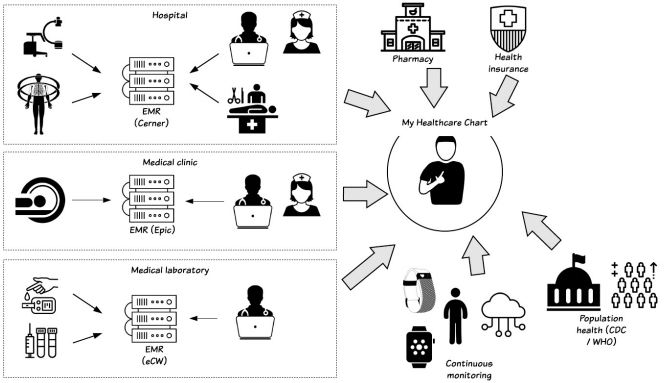
- In a Kafka cluster, Topics are split into Partitions and also replicated across brokers/clusters.
  1. However, to which partition a published message will be written, there is no guarantee about that.
  2. In one partition, messages are stored in the sequenced fashion.
  3. In a partition, each message is assigned an incremental id, also called offset.
  4. There can be any number of Partitions, there is no limitation.



Kafka Use Cases – eshop



Kafka Use Cases – Healthy AI



Apache Flume

- Apache Flume is an open-source tool:
  1. For collecting, aggregating, and moving huge amounts of streaming data from the external web servers to the central store, say HDFS, HBase, etc.

Apache Flume

- Apache Flume is an open-source tool:
  1. For collecting, aggregating, and moving huge amounts of streaming data from the external web servers to the central store, say HDFS, HBase, etc.
  2. The main purpose of designing Apache Flume is to move streaming data generated by various applications to Hadoop Distributed FileSystem.

Apache Flume

- Apache Flume is an open-source tool:
  1. For collecting, aggregating, and moving huge amounts of streaming data from the external web servers to the central store, say HDFS, HBase, etc.
  2. The main purpose of designing Apache Flume is to move streaming data generated by various applications to Hadoop Distributed FileSystem.
  3. Apache Flume is used to collect log data present in log files from web servers and aggregating it into HDFS for analysis.

Features of Apache Flume

- Apache Flume is a robust, fault-tolerant, and highly available service.

Features of Apache Flume

- Apache Flume is a robust, fault-tolerant, and highly available service.
- It is a distributed system with tunable reliability mechanisms for fail-over and recovery.

Features of Apache Flume

- Apache Flume is a robust, fault-tolerant, and highly available service.
- It is a distributed system with tunable reliability mechanisms for fail-over and recovery.
- Apache Flume is horizontally scalable.

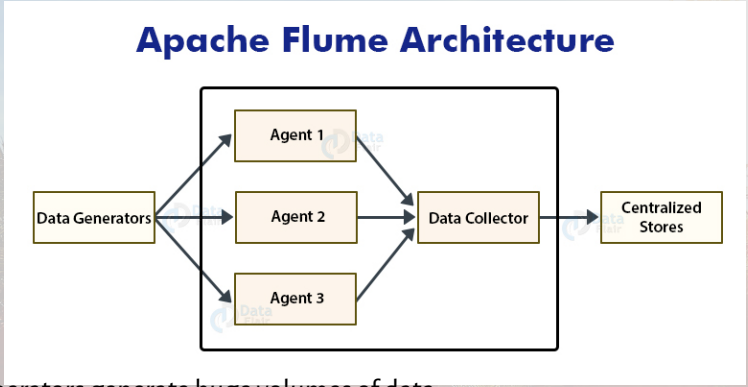
Features of Apache Flume

- Apache Flume is a robust, fault-tolerant, and highly available service.
- It is a distributed system with tunable reliability mechanisms for fail-over and recovery.
- Apache Flume is horizontally scalable.
- Apache Flume provides support for large sets of sources, channels, and sinks.

Features of Apache Flume

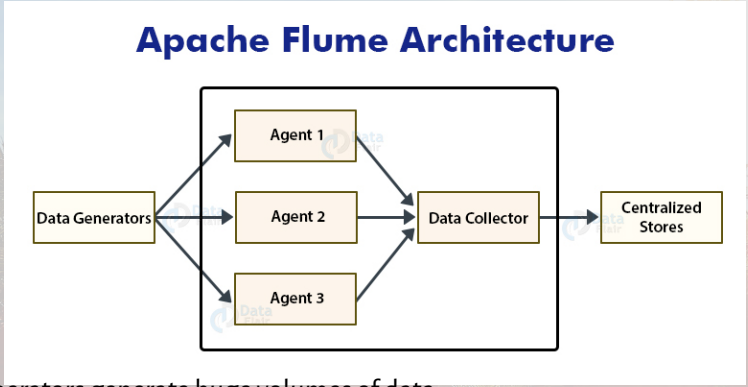
- Apache Flume is a robust, fault-tolerant, and highly available service.
- It is a distributed system with tunable reliability mechanisms for fail-over and recovery.
- Apache Flume is horizontally scalable.
- Apache Flume provides support for large sets of sources, channels, and sinks.
- Apache Flume can efficiently ingest log data from various servers into a centralized repository.

Apache Flume Architecture



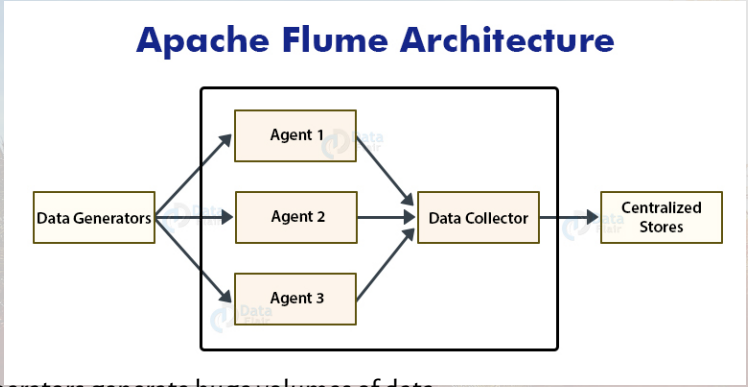
- Data generators generate huge volumes of data.

Apache Flume Architecture



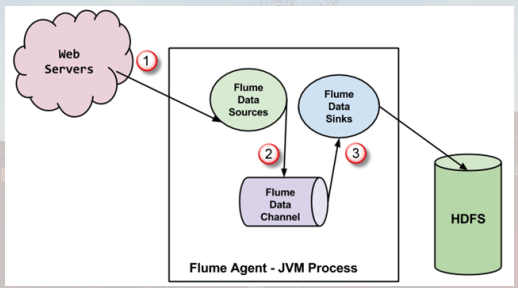
- Data generators generate huge volumes of data.
- Data generators are Facebook, Twitter, e-commerce sites, or various other external sources.

Apache Flume Architecture



- Data generators generate huge volumes of data.
- Data generators are Facebook, Twitter, e-commerce sites, or various other external sources.
- Data collector collects data from the agents, aggregates them, and pushes them into a centralized repository such as HBase or HDFS.

Apache Flume Architecture



- The events generated by external source (WebServer) are consumed by Flume Data Source.
- Flume Source receives an event and stores it into one or more channels. The channel acts as a store which keeps the event until it is consumed by the flume sink.
- Flume sink removes the event from a channel and stores it into an external repository like e.g., HDFS.

Kafka vs Flume

- **Kafka** runs as a cluster and handles incoming high volume data streams in real time
- **Flume** is a tool to collect log data from distributed web servers. The data collected will land into HDFS for further analysis

Kafka vs Flume

- **Kafka** runs as a cluster and handles incoming high volume data streams in real time
- **Flume** is a tool to collect log data from distributed web servers. The data collected will land into HDFS for further analysis
- **Kafka** stores a stream of records into different categories or topics.
- **Flume** is highly efficient and robust in processing log files, both in batch and real-time processing.

Kafka vs Flume

- **Kafka** runs as a cluster and handles incoming high volume data streams in real time
- **Flume** is a tool to collect log data from distributed web servers. The data collected will land into HDFS for further analysis
- **Kafka** stores a stream of records into different categories or topics.
- **Flume** is highly efficient and robust in processing log files, both in batch and real-time processing.
- **Kafka** will treat each topic partition as an ordered set of messages.
- **Flume** can take in streaming data from multiple sources for storage and analysis for use in HBase or Hadoop.

Kafka vs Flume

- **Kafka** runs as a cluster and handles incoming high volume data streams in real time
- **Flume** is a tool to collect log data from distributed web servers. The data collected will land into HDFS for further analysis
- **Kafka** stores a stream of records into different categories or topics.
- **Flume** is highly efficient and robust in processing log files, both in batch and real-time processing.
- **Kafka** will treat each topic partition as an ordered set of messages.
- **Flume** can take in streaming data from multiple sources for storage and analysis for use in HBase or Hadoop.
- **Kafka** will treat each topic partition as an ordered set of messages.
- **Flume** can take in streaming data from multiple sources for storage and analysis for use in HBase or Hadoop.

Kafka vs Flume

- Based on publish-subscribe architecture and does not track messages read by subscribers and who is the publisher.
- Ensures guaranteed data delivery because both the receiver and sender agents evoke the transaction to ensure guaranteed semantics.

Kafka vs Flume

- Based on publish-subscribe architecture and does not track messages read by subscribers and who is the publisher.
- Ensures guaranteed data delivery because both the receiver and sender agents evoke the transaction to ensure guaranteed semantics.
- **Kafka** can support a large number of publishers and subscribers and store large amounts of data.
- It can scale horizontally.

Kafka vs Flume

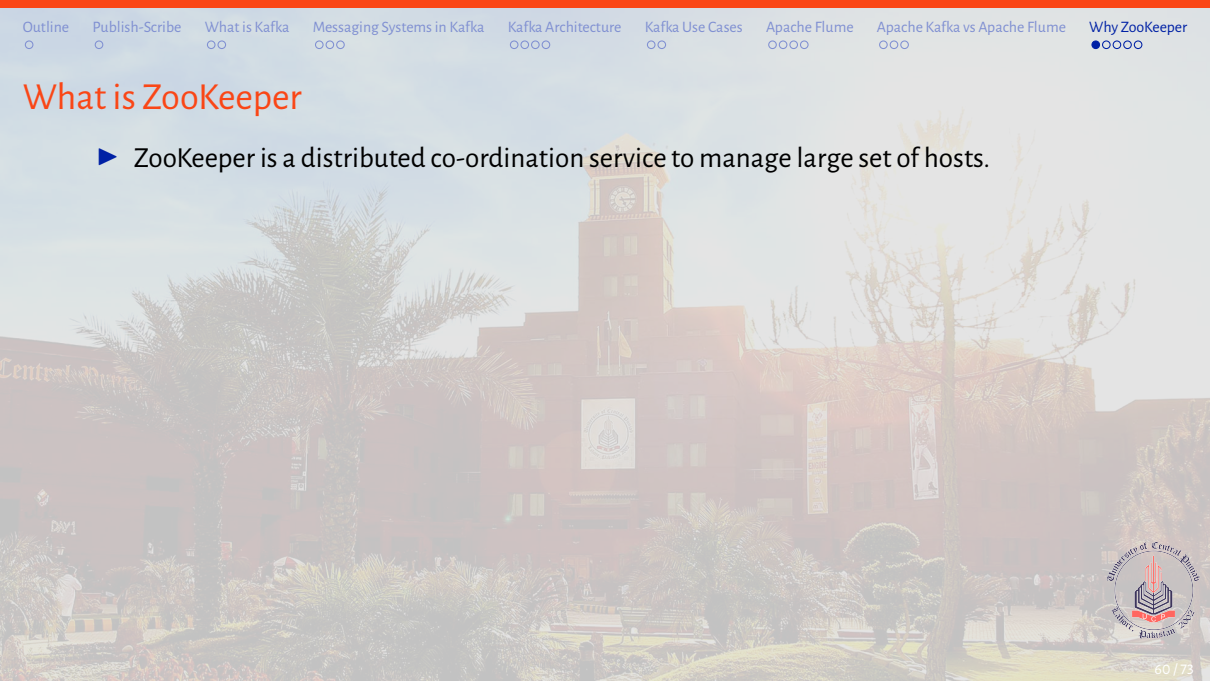
- Based on publish-subscribe architecture and does not track messages read by subscribers and who is the publisher.
- Ensures guaranteed data delivery because both the receiver and sender agents evoke the transaction to ensure guaranteed semantics.
- **Kafka** can support a large number of publishers and subscribers and store large amounts of data.
- It can scale horizontally.
- An efficient, fault-tolerant and scalable messaging system
- **Flume** is a service or tool for gathering data into Hadoop

Assignment 4:

Setup Flume library and Load&stream data from twitter using Flume.

What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.



What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.

What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.
- The common services provided by ZooKeeper are as follows:
  1. **Naming service:** Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.

What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.
- The common services provided by ZooKeeper are as follows:
  1. **Naming service:** Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
  2. **Configuration management:** Latest and up-to-date configuration information of the system for a joining node.

What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.
- The common services provided by ZooKeeper are as follows:
  1. **Naming service:** Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
  2. **Configuration management:** Latest and up-to-date configuration information of the system for a joining node.
  3. **Cluster management:** Joining / leaving of a node in a cluster and node status at real time.

What is ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.
- The common services provided by ZooKeeper are as follows:
  1. **Naming service:** Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
  2. **Configuration management:** Latest and up-to-date configuration information of the system for a joining node.
  3. **Cluster management:** Joining / leaving of a node in a cluster and node status at real time.
  4. **Leader election** Electing a node as leader for coordination purpose.

What is ZooKeeper

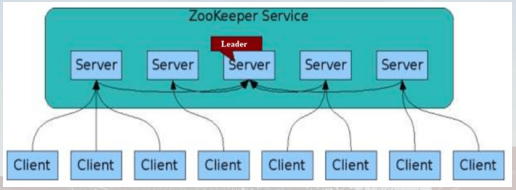
- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Coordinating and managing a service in a distributed environment is a complicated process.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.
- The common services provided by ZooKeeper are as follows:
  1. **Naming service:** Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
  2. **Configuration management:** Latest and up-to-date configuration information of the system for a joining node.
  3. **Cluster management:** Joining / leaving of a node in a cluster and node status at real time.
  4. **Leader election** Electing a node as leader for coordination purpose.
  5. **Locking and synchronization service** Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.

ZooKeeper Architecture

Before going deep into the working of ZooKeeper, let us take a look at the fundamental concepts of ZooKeeper.

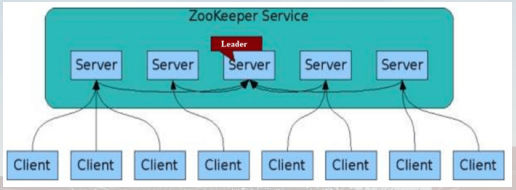
- Architecture
- Hierarchical namespace
- Session
- Watches

ZooKeeper Architecture



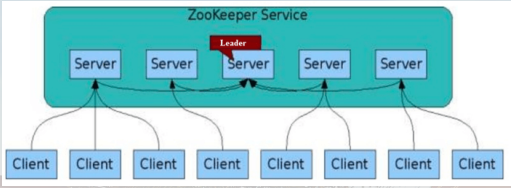
- **Clients**, one of the nodes in our distributed application cluster, access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive.

ZooKeeper Architecture



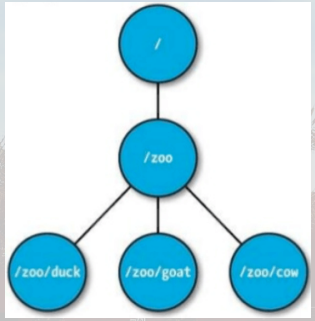
- **Clients**, one of the nodes in our distributed application cluster, access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive.
- **Ensemble:** Group of ZooKeeper servers. The minimum number of nodes that is required to form an ensemble is 3.

ZooKeeper Architecture



- **Clients**, one of the nodes in our distributed application cluster, access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive.
- **Ensemble:** Group of ZooKeeper servers. The minimum number of nodes that is required to form an ensemble is 3.
- **Server**, one of the nodes in our ZooKeeper ensemble, provides all the services to clients. Gives acknowledgement to client to inform that the server is alive.
- **Leader:** Server node which performs automatic recovery if any of the connected node failed. Leaders are elected on service startup.
- **Follower:** Server node which follows leader instruction.

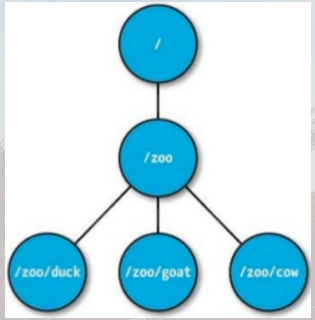
Zookeeper Data Model



- The zookeeper data model follows a Hierarchal namespace where each node is called a ZNode (**Persistence, Ephemeral and Sequential**). A node is a system where the cluster runs.

- Every znode in the ZooKeeper data model maintains a stat structure. It consists of **Version number**, **Action control list (ACL)**, **Timestamp**, and **Data length**.

Zookeeper Data Model



- The zookeeper data model follows a Hierarchal namespace where each node is called a ZNode (**Persistence, Ephemeral and Sequential**). A node is a system where the cluster runs.

- Every znode in the ZooKeeper data model maintains a stat structure. It consists of **Version number**, **Action control list (ACL)**, **Timestamp**, and **Data length**.
- **Sessions:** Requests in a session are executed in FIFO order. Once a client connects to a server, the session will be established and a session id is assigned to the client.
- **Watches:** are a simple mechanism for the client to get notifications about the changes in the ZooKeeper ensemble.

Thank You