Celcom – Kafka Integration

Kafka Training

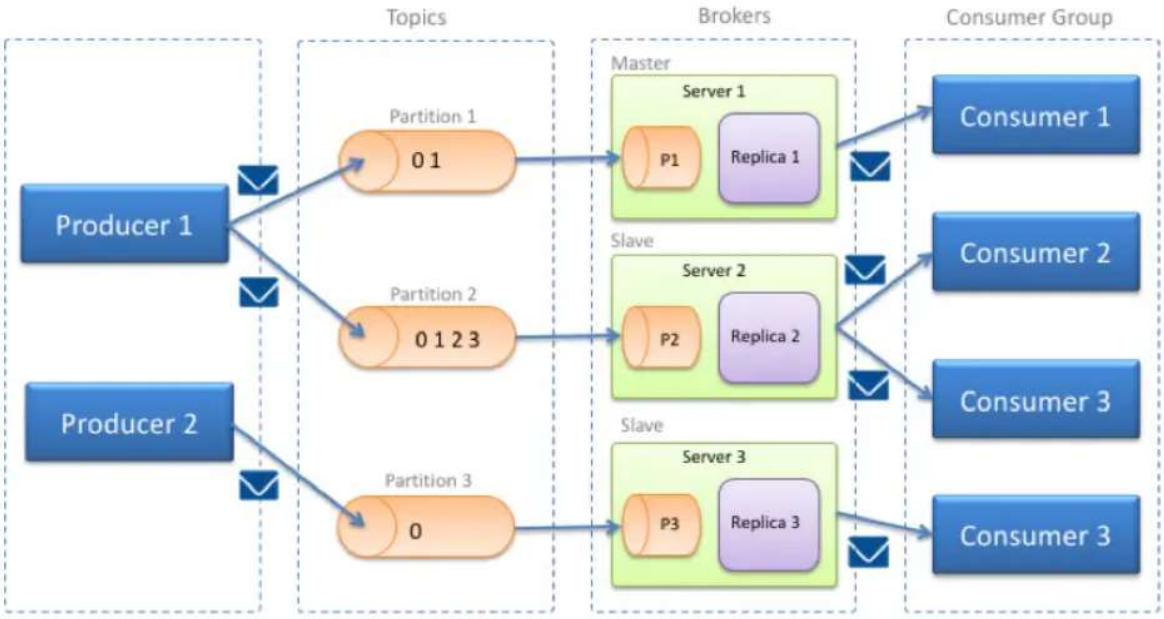
AXIATA DIGITAL LABS

Table of Contents

1: Apache Kafka Tutorial

- We will start from its basic concept and cover all the major topics related to Apache Kafka.
- Before moving on to this Kafka tutorial, you can learn the theory part here and for implementing the concepts through real-time projects,
- In our case we are considering Cloudera Kafka guideline and principles to implement accordingly

Reference: https://docs.cloudera.com/documentation/enterprise/6/6.1/topics/kafka.html



1.1: What is kafka?

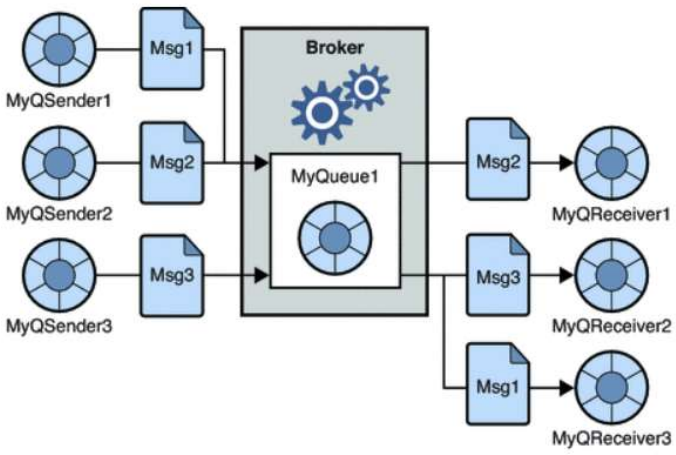
- Apache Kafka is a fast, scalable, fault-tolerant messaging system which enables communication between producers and consumers using message-based topics.
- In simple words, it designs a platform for high-end new generation distributed applications.
- It allows many permanent or ad-hoc consumers. One of the best features of Kafka is, it is highly available and resilient to node failures and supports automatic recovery.
- This feature makes Apache Kafka ideal for communication and integration between components of large-scale data systems in real-world data systems.

1.2: Messaging Systems in Kafka

- The main task of managing system is to transfer data from one application to another so that the applications can mainly work on data without worrying about sharing it.
- Distributed messaging is based on the reliable message queuing process. Messages are queued non-synchronously between the messaging system and client applications.

➢ There are two types of messaging patterns available

- Point to point messaging system



- In this messaging system, messages continue to remain in a queue. More than one consumer can consume the messages in the queue but only one consumer can consume a message.
- After the consumer reads the message in the queue, the message disappears from that queue.

- Publish-subscribe messaging system

- 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.
- Message producers are known as publishers and Kafka consumers are known as subscribers.

1.3: History of Apache Kafka

- Previously, LinkedIn was facing the issue of low latency ingestion of huge amount of data from the website into a lambda architecture which could be able to process real-time events.
- As a solution, Apache Kafka was developed in the year 2010, since none of the solutions was available to deal with this drawback, before.
- However, there were technologies available for batch processing, but the deployment details of those technologies were shared with the downstream users.
- Hence, when it comes to Real-time Processing, those technologies were not enough suitable. Then, in the year 2011 Kafka was made public.

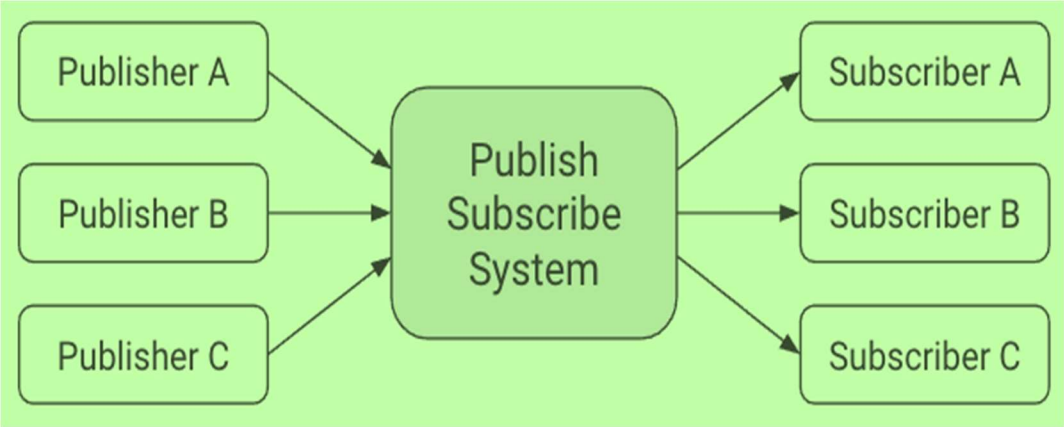
- Need of Apache Kafka Cluster
  - There is an enormous volume of data used in Big Data. For data, there are two main challenges.
  - One is how to collect and manage a large volume of data and the other one is the analysis of the collected data. For dealing with such challenges, you need to have a messaging system.
  - There are many benefits of Apache Kafka that justifies the usage of Apache Kafka:
  - Tracking web activities by storing/sending the events for real-time processes.
  - Alerting and reporting the operational metrics.
  - Transforming data into the standard format.
  - Continuous processing of streaming data to the topics.
  - Therefore, this technology is giving a tough competition to some of the most popular applications like ActiveMQ, RabbitMQ, AWS, etc because of its wide use.

- Audience for Kafka Tutorial
  - Professionals who are aspiring to make a career in Big Data Analytics using Apache Kafka messaging system should refer to this Kafka Tutorial.
  - It will give you a complete understanding of Apache Kafka.
  - As prerequisites to Kafka, you must have a good understanding of Java, Scala, Distributed messaging system, and Linux environment, before proceeding with this Apache Kafka Tutorial.

- Ideal Publish-Subscribe System

  Learn about Kafka's architecture and how it compares to an ideal publish-subscribe system.

  The ideal publish-subscribe system is straightforward: Publisher A's messages must make their way to Subscriber A, Publisher B's messages must make their way to Subscriber B, and so on.



An ideal system has the benefit of:

- Unlimited Lookback. A new Subscriber A1 can read Publisher A's stream at any point in time.
- Message retention - no messages are lost.
- Unlimited storage. The publish-subscribe system has unlimited storage of messages.
- No Downtime, the publish-subscribe system is never down.
- Unlimited Scaling. The publish-subscribe system can handle any number of publishers and/or subscribers with constant message delivery latency.

Kafka's architecture however deviates from this ideal system. Some of the key differences are:

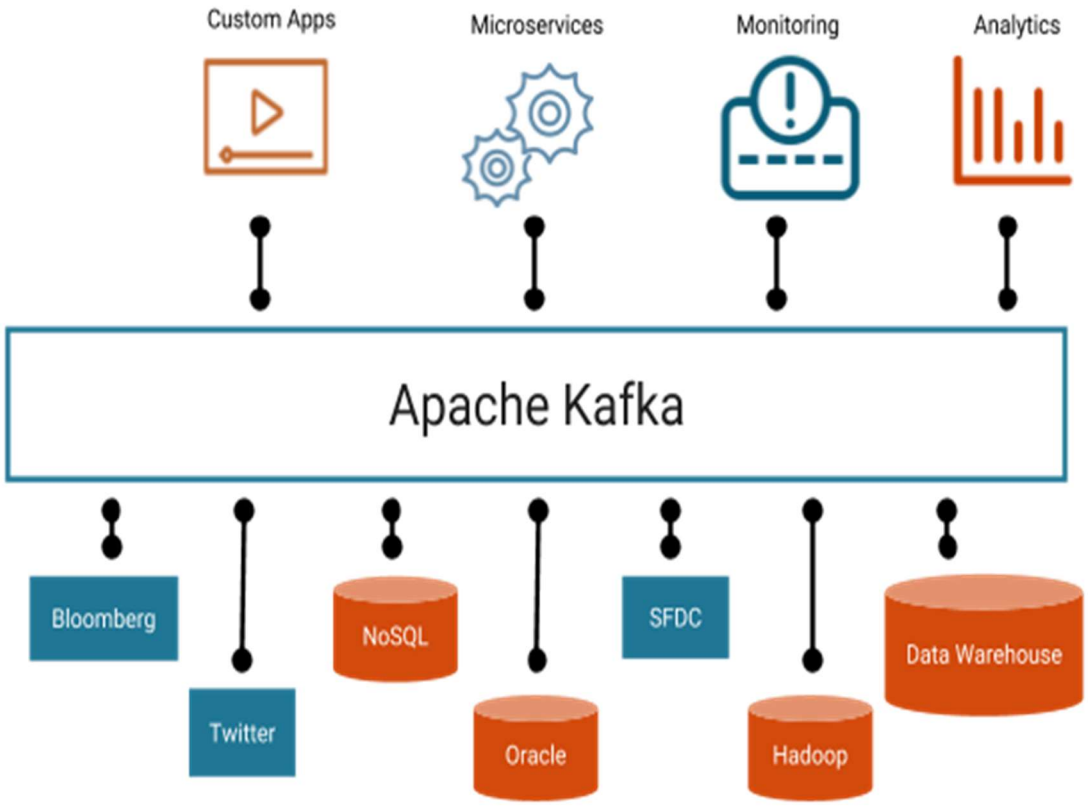
- Messaging is implemented on top of a replicated, distributed commit log.
- The client has more functionality and, therefore, more responsibility.
- Messaging is optimized for batches instead of individual messages.
- Messages are retained even after they are consumed; they can be consumed again.

The results of these design decisions are:

- Extreme horizontal scalability
- Very high throughput
- High availability
- Different semantics and message delivery guarantees

2: Terminology of Kafka

- Kafka uses its own terminology when it comes to its basic building blocks and key concepts.
- The usage of these terms might vary from other technologies. The following provides a list and definition of the most important concepts of Kafka:



- **Broker:** A broker is a server that stores messages sent to the topics and serves consumer requests.
- **Topic:** A topic is a queue of messages written by one or more producers and read by one or more consumers.
- **Producer:** A producer is an external process that sends records to a Kafka topic.
- **Consumer:** A consumer is an external process that receives topic streams from a Kafka cluster.
- **Client:** Client is a term used to refer to either producers or consumers.
- **Record:** A record is a publish-subscribe message. A record consists of a key/value pair and metadata including a timestamp.
- **Partition:** Kafka divides records into partitions. Partitions can be thought of as a subset of all the records for a topic.

3: Advantages of Kafka

- It is very important to know the limitations of any technology before using it, same in case of advantages.

➢ **Advantages of Kafka**

Basically, these Kafka advantages are making Kafka ideal for our data lake implementation

- **High-throughput**
  Without having not so large hardware, Kafka is capable of handling high-velocity and high-volume data. Also, able to support message throughput of thousands of messages per second.

- **Low Latency**
  It can handle these messages with the very low latency of the range of milliseconds, demanded by most of the new use cases.

- **Fault-Tolerant**
  One of the best advantages is Fault Tolerance. There is an inherent capability in Kafka, to be resistant to node/machine failure within a cluster.

- **Durability**
  Here, durability refers to the persistence of data/messages on disk. Also, messages replication is one of the reasons behind durability, hence messages are never lost.

- **Scalability**
  Without incurring any downtime on the fly by adding additional nodes, Kafka can be scaled-out. Moreover, inside the Kafka cluster, the message handling is fully transparent, and these are seamless.

- **Distributed**
  The distributed architecture of Kafka makes it scalable using capabilities like replication and partitioning.

- **Message Broker Capabilities**
  Kafka tends to work very well as a replacement for a more traditional message broker. Here, a message broker refers to an intermediary program, which translates messages from the formal messaging protocol of the publisher to the formal messaging protocol of the receiver.

- **High Concurrency**
  Kafka can handle thousands of messages per second and that too in low latency conditions with high throughput. In addition, it permits the reading and writing of messages into it at high concurrency.

- **By Default, Persistent**
  As we discussed above that the messages are persistent, that makes it durable and reliable.

- **Consumer Friendly**
  It is possible to integrate with the variety of consumers using Kafka. The best part of Kafka is, it can behave or act differently according to the consumer, that it integrates with because each customer has a different ability to handle these messages, coming out of Kafka. Moreover, Kafka can integrate well with a variety of consumers written in a variety of languages.

- **Batch Handling Capable (ETL like functionality)**
  Kafka could also be employed for batch-like use cases and can also do the work of a traditional ETL, due to its capability of persists messages.

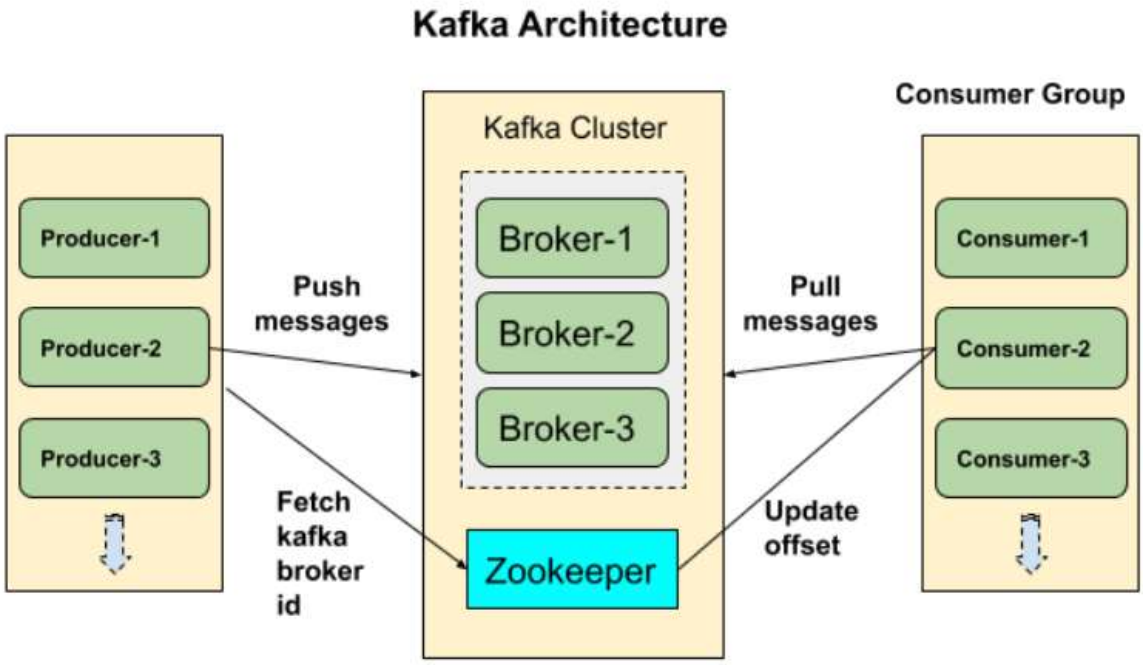
- **Variety of Use Cases**
  It can manage the variety of use cases commonly required for a Data Lake. For example, log aggregation, web activity tracking, and so on.

- **Real-Time Handling**
  Kafka can handle real-time data pipeline. Since we need to find a technology piece to handle real-time messages from applications, it is one of the core reasons for Kafka as our choice.

4: **Kafka Architecture**

we will learn about Kafka Broker, Kafka Consumer, Zookeeper, and Kafka Producer. And we will see some fundamental concepts of Kafka.

- The below diagram shows the cluster diagram of Apache Kafka



- **Kafka Broker**

    - Basically, to maintain load balance Kafka cluster typically consists of multiple brokers. However, these are stateless, hence for maintaining the cluster state they use Zookeeper.

    - Although, one Kafka Broker instance can handle hundreds of thousands of reads and writes per second. Whereas, without performance impact, each broker can handle TB of messages. In addition, make sure Zookeeper performs Kafka broker leader election.

- **Kafka – Zookeeper**

    - For the purpose of managing and coordinating, Kafka broker uses Zookeeper. Also, uses it to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.

    - As soon as Zookeeper send the notification regarding presence or failure of the broker then producer and consumer, take the decision and starts coordinating their task with some other broker.

- **Kafka Producers**

    - Further, Producers in Kafka push data to brokers. Also, all the producers search it and automatically sends a message to that new broker, exactly when the new broker starts.

    - However, keep in mind that the Kafka producer sends messages as fast as the broker can handle, it doesn't wait for acknowledgments from the broker.

- **Kafka Consumers**

    - Basically, by using partition offset the Kafka Consumer maintains that how many messages have been consumed because Kafka brokers are stateless. Moreover, you can assure that the consumer has consumed all prior messages once the consumer acknowledges a message offset.

    - Also, in order to have a buffer of bytes ready to consume, the consumer issues an asynchronous pull request to the broker. Then simply by supplying an offset value, consumers can rewind or skip to any point in a partition. In addition, Zookeeper notifies Consumer offset value.

5: Setup Kafka Cluster: Apache Kafka

we will learn Kafka multi-node cluster setup and Kafka multi-broker cluster setup. Also, we will see Kafka Zookeeper cluster setup.

5.1: Kafka Cluster Setup

- In order to gain better reliability and high availability of the Kafka service, we need to setup Kafka in cluster mode at very first.
- In our cloudera manager application we must download latest parcel for kafka and it should be distributed between related kafka brokers

➢ Now, follow several steps to set up Kafka Cluster:

- Make a folder of name "logs". In this folder, all the Kafka logs will be stored.
- Then, open the server. properties file, ongoing to the config directory. Here, we will find the file, which contains Kafka broker configurations.
- Further, set broker.id to 1. Make sure it is the id of the broker in a Kafka Cluster, so for each broker, it must be unique.
- Then, listener's configuration and set it to PLAINTEXT://localhost:9092. It says, for connection requests, the Kafka broker will be listening on port 9092.
    - If we are going to setup secure manner. We have to uses (SASL_SSL) self-certificate authentication within the brokers.
    - SASL_SSL://localhost:9093. It says, for connection requests, the Kafka broker will be listening on port 9093.
- Also, set the Apache Zookeeper address, in the zookeeper. Connect configuration. However, if Zookeeper is running in a Kafka cluster, then ensure to give the address as a, i.e.:localhost:2181
- Basically, these are some general configurations that we need to be set up for the Production environment.
- Therefore, for all brokers, our configuration is ready. Now, run the command ./bin/kafka-server-start.sh config/server.properties, ongoing to the home directory of each Kafka folder.

- Execute the command (all as one line):

```
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor

3 --partitions 5 --topic demo
```

- Here with a replication factor of three for each partition, 5 partitions are created.
- On defining a replication factor of three, there will be one leader and two followers, for a partition.
- Also, at the time when message or record is sent to the leader, it is copied in followers.

- Execute this command:

```
./bin/kafka-topics.sh --describe --topic demo --zookeeper localhost:2181
```

It helps us to know that which broker is the leader or follower for which partition.

- Output:

```
Topic:demoPartitionCount:50ReplicationFactor:3Configs:
Topic:          demoPartition:          0Leader:          3Replicas:          2,3,1Isr:          2,3,1
Topic:          demoPartition:          1Leader:          3Replicas:          3,1,2Isr:          3,1,2
Topic:          demoPartition:          2Leader:          3Replicas:          1,2,3Isr:          1,2,3
Topic:          demoPartition:          3Leader:          3Replicas:          2,1,3Isr:          2,1,3
Topic:          demoPartition:          4Leader:          3Replicas:          3,2,1Isr:          3,2,1
Topic:          demoPartition:          5Leader:          3Replicas:          1,3,2Isr:          1,3,2
```

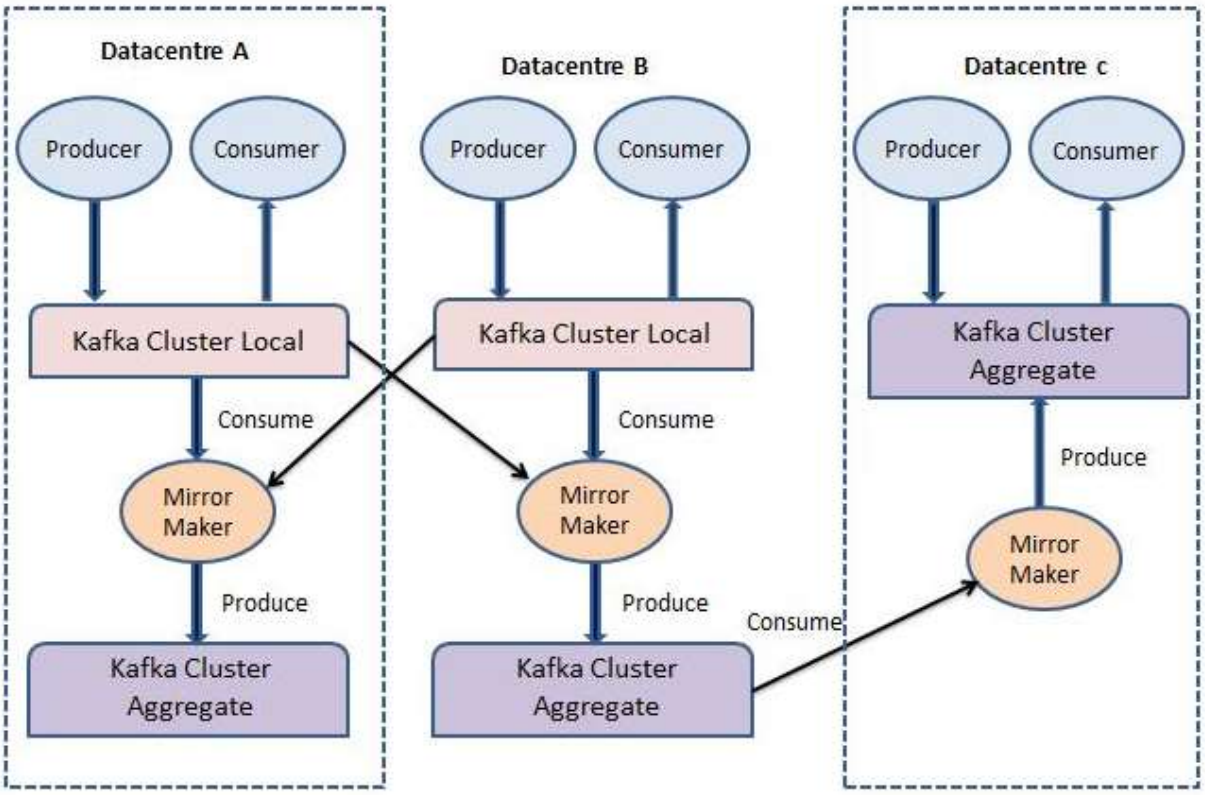
And here ISR refers to **I**n **S**ync **R**eplicas.

5.2: KafkaMirror Maker

- Apache Kafka has a simple tool in order to replicate data from two data centers. It is called MirrorMaker and at its base, it is a series of consumers (named streams) who are all part of the same consumer group and read data from the range of topics you have selected to replicate.
- It is more than getting tied together by a Kafka consumer and producer. Information will be interpreted from topics in the origin cluster and written in the destination cluster to a topic with the same name. Let us explore more about Kafka MirrorMaker by understanding its architecture

➢ Architecture of Kafka MirrorMaker

- The data duplication mechanism between Kafka clusters is named "mirroring". Mirroring function is commonly used for keeping a separate copy of a Kafka cluster in another data center.
- Kafka's MirrorMaker module reads the data from topics in one or more Kafka clusters source and writes the relevant topics to a Kafka cluster destination (using the same topic names).
- The source and target clusters are entirely independent so that they can have specific partition numbers and different offsets.
- The below figure shows an example of an architecture of Kafka MirrorMaker which aggregates messages from two different clusters into an aggregate cluster and then copying that aggregated cluster to another datacenter.



- Particularly while operating with multiple data centers, it is almost always important to copy messages between them. Therefore, online applications can have access to user activity on both domains.
- For instance, if a user changes the personal information in their account, the update will need to be noticeable irrespective of the data center that shows the search results.
- Kafka clusters can only replicate within a single cluster and not between different clusters. Kafka does not allow replication within multiple clusters.
- Every MirrorMaker operation has one producer. The process is easy. MirrorMaker runs a thread for each consumer.
- Each user collects events from the topics and partitions allocated to him on the source cluster and uses the mutual producer to send events to the target cluster.
- The consumers must tell the producer every 60 seconds (by default) to submit all the events to Kafka and wait before Kafka accepts those events.

- Consumers then contact the Kafka cluster source to assign the offsets for all these events. It means no data loss (messages are acknowledged by Kafka until offsets are committed to the source), and if the MirrorMaker mechanism fails, there will be no more than 60 seconds worth of duplicates.

- It is essentially a consumer-producer that the consumer goes to cluster A and connects to the topic you decide and it gets the data from there and generates all the message in cluster B all the message that received by a topic in cluster A will be available in cluster B in the same topic.

➢ Benefits of Kafka MirrorMaker

Below are the benefits of Kafka MirrorMaker:

- Global and Central Clusters
  - The organization has one or more datacenters in different geographic areas, cities or continents in some instances.
  - Many systems can only operate by connecting with the local cluster, but some applications need data from multiple data centers (otherwise you would not be looking at approaches for cross-data center replication).
  - There are many situations where this is a prerequisite, but the classic example is a business that adjusts pricing based on supply and demand.
  - That organization can have a datacenter in each area where it has a location, gather local supply and demand statistics, then adjust prices appropriately.
  - All this information is then replicated to a central cluster where business analysts will report on their sales across the group.

- Redundancy (DR)

  - Programs operate on a single Kafka cluster and do not need data from other sites, but you are worried about the whole cluster's capacity becoming unavailable for some reason.
  - Then you would like to have a second Kafka cluster with all the data in the first cluster, so you can direct your applications to in case of an emergency.

- Cloud Migrations

    - Many companies are running their business these days in both an on-site datacenter and a cloud provider. Sometimes, cloud platform programs operate on multiple regions for flexibility, and different cloud providers are sometimes used.
    - In these situations, at least one Kafka cluster is often present in each on-premises datacenter and in each cloud area.
    - Those Kafka clusters are used by applications in each datacentre and region to transfer data efficiently between the datacenters.
    - For instance, if a new application is introduced in the cloud but needs certain data that is modified by applications running in the on-site datacenter and stored in an on-site database, you can use Kafka Connect to catch changes in the database in the local Kafka cluster and then replicate those changes in the Kafka cluster where the new application is located.
    - It helps to control the effects of cross-data traffic and increases governance and security of the traffic.

5.3: Kafka Console Producer

- A Kafka-console-producer is a program that comes with Kafka packages which are the source of data in Kafka.
- It is used to read data from standard input or command line and write it to a Kafka topic (place holder of messages).
- Kafka-console-producer keeps data into the cluster whenever we enter any text into the console.
- Topics are made of partitions where producers write this data. In other words, “it creates messages from command line input (STDIN)”.

➢ In Kafka, there are two types of producers

    - Sync -It sends messages directly in the background.
    - A sync-It send messages whenever considering the number of messages with higher throughput.

- The producer automatically finds broker and partition where data to write.
- It removes the dependency by connecting to Kafka and then producer that is going to produce messages to respective broker and partitions.
- When there is a broker failure and some reason broker is going down, the producer will automatically recover, this producer provides booster among the partition and broker.
- It’s just well-programmed. simply we don’t have to implement the features.

- We have producer which is sending data to partition 0 of broker 1 of topic A. Producer Know which brokers to write to.
- Basically, if the producer sends data without key, then it will choose a broker based on the round-robin algorithm.
- The producer does load balancer among the actual brokers. The producer used to write data by choosing to receive an acknowledgment of data.

➢ Types of Confirmation or Acknowledgment Node

- ack=0; in this case we don't have actual knowledge about the broker. If the producer sends data to a broker and it's already down there is a chance of data loss and danger to use as well.
- ack=1; This is the default confirmation from the brokers where a producer will wait for a leader that is a broker. In this case, the broker is present and ready to accept data from the producer
- ack=all; In this case we have a combination of Leader and Replicas. if there is any broker is failure the same set of data is present in replica and possibly there is possibly no data loss.

➢ Features of Kafka console Producer

- It is Thread-safe: -In each producer has a buffer space pool that holds records, which is not yet transmitted to the server. The I/O thread which is used to send these records as a request to the cluster.
- Kafka-console producer is Durable: -The acks is responsible to provide a criteria under which the request ace considered complete. With the help of ack=" all", blocking on the full commit of the record, this setting considered as durable setting
- It provides scalability: -The producer maintains buffers of unsent records for each partition. These buffers are sent based on the batch size which also handles many messages simultaneously.
- Kafka-console producers Fault-tolerant: –When there is a node failure down, the producer has an essential feature to provide resistance to a node and recover automatically.
- The Kafka console producer is idempotent, which strengthens delivery semantics from at least once to exactly once delivery.
- It has also used a transactional mode that allows an application to send messages to multiple partitions which includes topic as well automatically.

5.4: Kafka Console Consumer

- The Kafka offers a command utility to view messages from the command line. Kafka offers the utility kafka-console-consumer.sh which helps to read messages from topic on command line.
- Kafka-console-consumer simply reads from a Kafka topic and writes data to console (standard output). By default, it outputs the raw bytes in the message with no formatting (using the Default Formatter).

5.5: Kafka Zookeeper

- Zookeeper is an important part of Apache Kafka. Zookeeper is a cornerstone for so many distributed applications, as it provides fantastic features.
- Zookeeper is used by Apache Kafka to store information regarding the Kafka cluster, as well as user info, in short, we can say that Zookeeper stores metadata about the Kafka cluster.
- It's important to us to understand what Zookeeper is and how Kafka fits with it. We'll see what Zookeeper does in-depth and we'll learn why we need to use it.

➢ What is Kafka Zookeeper?

- Zookeeper is a centralized, open-source software that manages distributed applications. It provides a basic collection of primitives for the implementation of higher-level synchronization, framework management, groups, and naming services.
- It is planned to be programmable and simple to use. The well-known companies that use Zookeeper are Yahoo, Twitter, Netflix, and Facebook these are just a few names.
- It keeps track of information that need to be synchronized across your cluster. Information such as:

    - Which node is the master?
    - Which workers will perform which tasks?
    - Which workers are currently available?

- It's a tool that applications can use to recover from partial failures in your cluster. It also plays an integral part of HBase, High-Availability (HA) MapReduce, Drill, Storm, Solr, and much more.
- Zookeeper is itself a distributed application providing automated code-writing facilities.

The specific services that Zookeeper offers are as follows:

- Naming service: Identifying the nodes by name in a This is DNS-like except with nodes.
- Configuration monitoring: The system's current and up-to-date configuration details for a node that joins.
- Cluster control: Real-time connection / leaving of a node in a cluster and node.
- Leader Election: Selecting a node as lead.

➢ How Kafka uses Zookeeper?

▪ Below are the points to use zookeeper in Kafka:

- Electing Leader: Maintaining the relationship between leader and follower for all partitions is handled by the controller which is one of the brokers If one node goes down, it's the controller who asks other followers to become leaders for a partition to replace the lost Zookeeper selects only a new leader, to make sure that there is only one leader.
- Membership of Cluster: What brokers and a member of the cluster are alive? It is handled by Zookeeper
- Configuration of Topic: How many partitions does each topic has, do the topics exists or not and if exists then where are the followers who are the leader selected
- Quotas: what will be the amount of data for each client to read and write
- ACL: Who has access to read and write and to which topic, how many user groups exist and its members also information about the latest offset from each

6: Kafka Monitoring

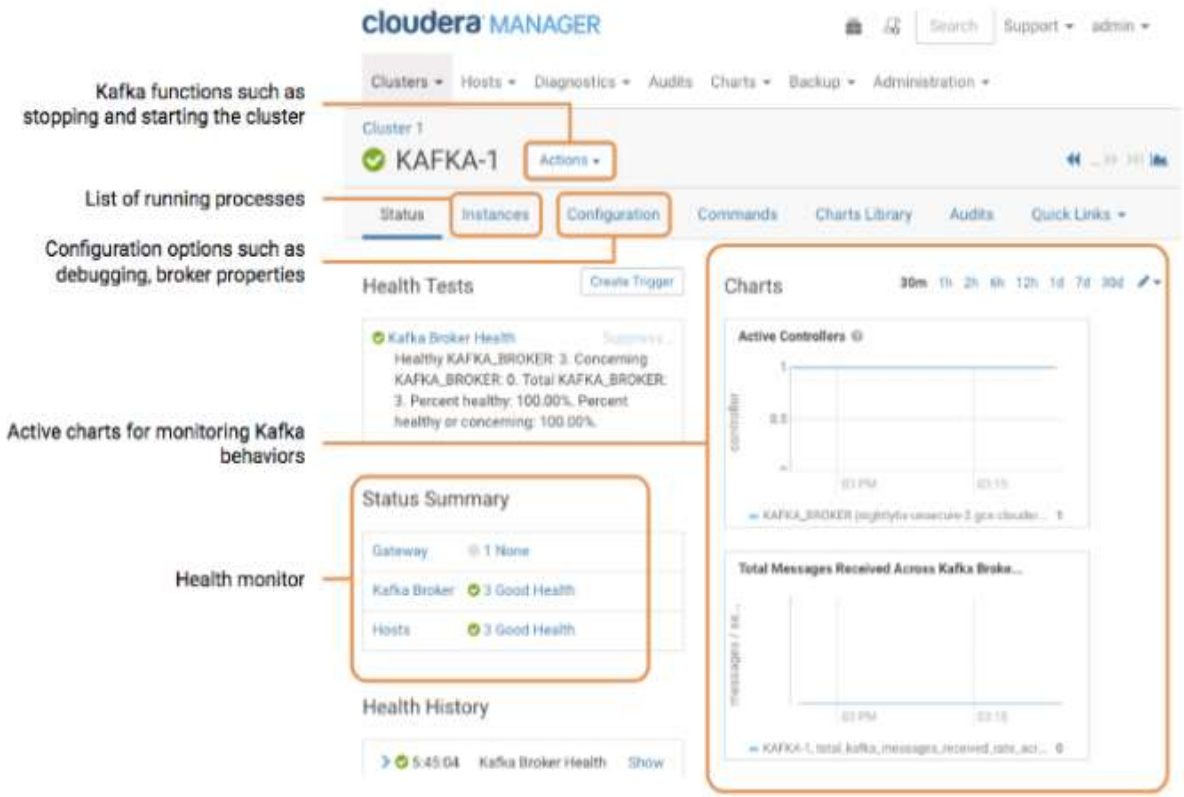
With the increasing demand around Kafka Cluster monitoring and management, several open-source and commercial graphical tools have reached the market, offering a variety of administration and monitoring functionalities.

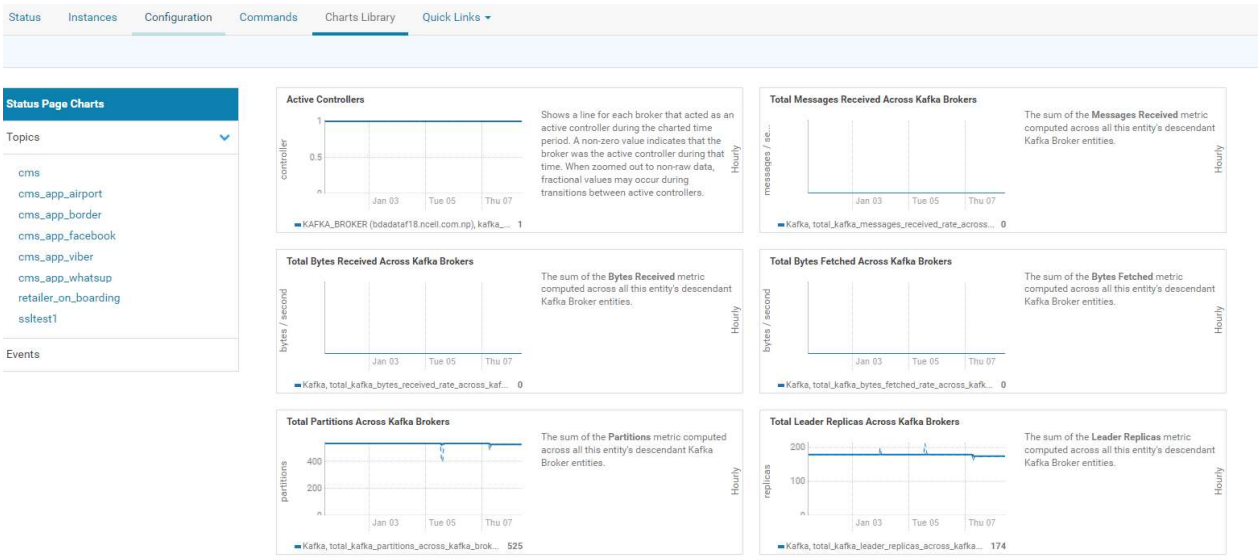
- What are the best monitoring tools for Apache Kafka?

    - Confluent Control Centre
    - Lenses
    - Datadog Kafka Dashboard
    - Cloudera Manager
    - Yahoo Kafka Manager
    - KafDrop
    - LinkedIn Burrow
    - Kafka Tool

- Cloudera Manager

- Kafka in Cloudera Manager is clearly a less rich monitoring tool compared to Confluent, Lenses and Datadog.
- However, it is very convenient for companies that are already customers of Cloudera and need their monitoring mechanisms under the same platform.


cloudera MANAGER
Search
Support
admin
Clusters
Hosts
Diagnostics
Audits
Charts
Backup
Administration
Cluster 1
KAFKA-1
Actions
Status
Instances
Configuration
Commands
Charts Library
Audits
Quick Links
Health Tests
Create Trigger
Kafka Broker Health
Healthy KAFKA_BROKER: 3. Concerning KAFKA_BROKER: 0. Total KAFKA_BROKER: 3. Percent healthy: 100.00%. Percent healthy or concerning: 100.00%.
Charts
30m
Active Controllers
Total Messages Received Across Kafka Broke...
Status Summary
Gateway
1 None
Kafka Broker
3 Good Health
Hosts
3 Good Health
Health History
5:45:04
Kafka Broker Health
Show
Kafka functions such as stopping and starting the cluster
List of running processes
Configuration options such as debugging, broker properties
Active charts for monitoring Kafka behaviors
Health monitor


Status
Instances
Configuration
Commands
Charts Library
Quick Links
Status Page Charts
Topics
cms
cms_app_airport
cms_app_border
cms_app_facebook
cms_app_viber
cms_app_whatsup
retailer_on_boarding
ssltest1
Events
Active Controllers
Shows a line for each broker that acted as an active controller during the charted time period. A non-zero value indicates that the broker was the active controller during that time. When zoomed out to non-raw data, fractional values may occur during transitions between active controllers.
Total Messages Received Across Kafka Brokers
The sum of the Messages Received metric computed across all this entity's descendant Kafka Broker entities.
Total Bytes Received Across Kafka Brokers
The sum of the Bytes Received metric computed across all this entity's descendant Kafka Broker entities.
Total Bytes Fetched Across Kafka Brokers
The sum of the Bytes Fetched metric computed across all this entity's descendant Kafka Broker entities.
Total Partitions Across Kafka Brokers
The sum of the Partitions metric computed across all this entity's descendant Kafka Broker entities.
Total Leader Replicas Across Kafka Brokers
The sum of the Leader Replicas metric computed across all this entity's descendant Kafka Broker entities.
Hourly

7: Kafka Operations

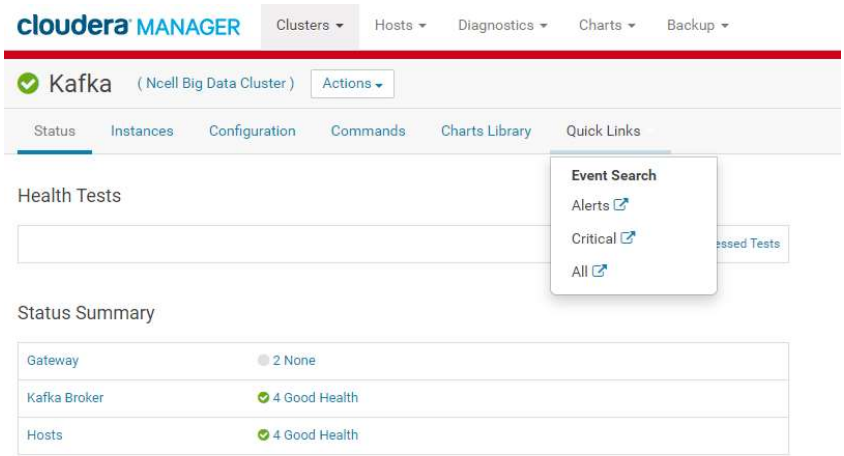
7.1: Initial important facts

- Kafka brokers should be up and running
- DNS resolution should be available with master nodes from kafka brokers
- Kerberos authentication server side and users
- Zookeeper nodes should be up
- Sentry should be up
- TSL and SSL certification verification should be fine
- NTP service should be running all brokers
- Kafka brokers and Zookeeper port should be telnet
- Java heap size Zookeeper consider, if any performance lag
- Disk space alerts for kafka data folders should be placed in order

Kafka application logs verification

- Front end log verification

  Cloudera manager has logs search location



- Backend log verifications

  - /var/log/kafka
  - /var/log/zookeeper

7.2 Essential Kafka commands

- Topics create and describe
  1. kdestroy
  2. unset KAFKA_OPTS
  3. kafka-topics --zookeeper nttdsaamnp003/kafka --create --replication-factor 3 --topic demo --partitions 8
  4. kafka-topics --zookeeper nttdsaamnp003/kafka --describe --topic demo

- Topic List
  1. kafka-topics --zookeeper nttdsaamnp003/kafka --list 2>/dev/null

- Topic alter and delete
  1. kafka-topics --zookeeper nttdsaamnp003/kafka --alter --partitions 10 --topic demo 2>/dev/null
  2. kafka-topics --zookeeper nttdsaamnp003/kafka --describe --topic demo 2>/dev/null
  3. kafka-topics --zookeeper nttdsaamnp003/kafka --delete --topic testdemo 2>/dev/null
  4. kafka-topics --zookeeper nttdsaamnp003/kafka --describe --topic demo 2>/dev/null

4. Permission assigning through sentry
   1. Creating user group and adding AD user(should be executed on all kafka brokers)

      groupadd svc_kafka
      chmod -aG svc_kafka svc_kafka
      id svc_kafka

   2. Before sentry operations, you must kinit with kafka user
      1. cd /var/run/cloudera-scm-agent/process/
      2. select the kafka runtime folder
         ps -ef|grep BRO
         cd {foldername}
      3. kinit -kt kafka.keytab kafka/nttkafkanp003.celcom.com.my@CELCOM.AD
      4. check keytab validation using “klist”

   3. Sentry operations - adding a producer

      1. kafka-sentry -cr -r kafka_demo_w 2>/dev/null
      2. kafka-sentry -gpr -r kafka_demo_w -p "Host=10.5.130.42->Topic=demo->Action=describe"
      3. kafka-sentry -gpr -r kafka_demo_w -p "Host=10.5.130.42->Topic=demo->Action=write"
      4. kafka-sentry -arg -r kafka_demo_w -g svc_kafka
      5. kafka-sentry -lr -g svc_kafka |grep kafka_demo_w

   4. Sentry operations – adding a consumer with consumer group

      1. kafka-sentry -cr -r kafka_demo_r 2>/dev/null
      2. kafka-sentry -gpr -r kafka_demo_r -p "Host=10.5.130.42->Topic=demo->Action=describe"
      3. kafka-sentry -gpr -r kafka_demo_r -p "Host=10.5.130.42->Topic=demo->Action=read"
      4. kafka-sentry -gpr -r kafka_allowed_group_list -p "Host=10.5.130.42->consumergroup=demo_group->Action=describe"
      5. kafka-sentry -gpr -r kafka_allowed_group_list -p "Host=10.5.130.42->consumergroup=demo_group->Action=read"
      6. kafka-sentry -arg -r kafka_demo_r -g svc_kafka
      7. kafka-sentry -lr -g svc_kafka |grep kafka_demo_r

5. Running a console producer
    1. export KAFKA_OPTS="-Djava.security.auth.login.config=/home/kafka_adl/kafka_backup/scripts/jaas.conf";
    2. kafka-console-producer --topic demo --broker-list nttkafkanp001.celcom.com.my:9093,nttkafkanp002.celcom.com.my:9093,nttkafkanp003.celcom.com.my:9093 --producer.config /home/kafka_adl/kafka_backup/scripts/producer-client-ssl.properties;

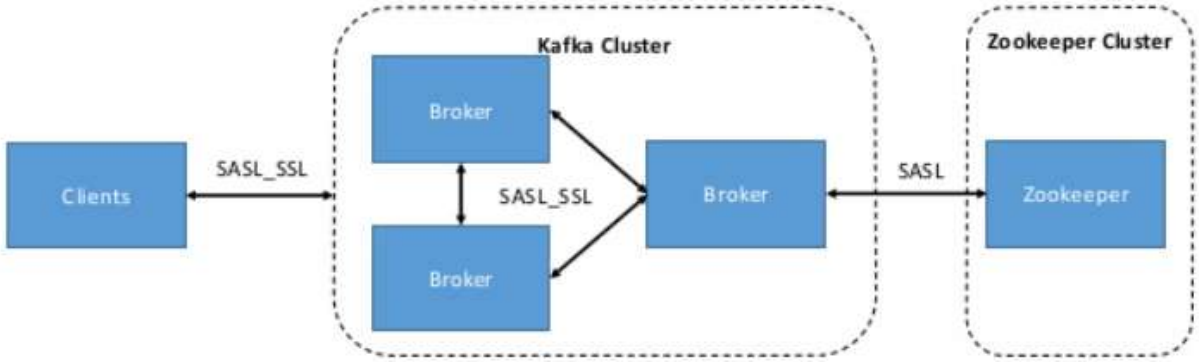
6. Running a console consumer
    1. export KAFKA_OPTS="-Djava.security.auth.login.config=/home/kafka_adl/kafka_backup/scripts/jaas.conf";
    2. kafka-console-consumer --topic demo --bootstrap-server nttkafkanp001.celcom.com.my:9093,nttkafkanp002.celcom.com.my:9093,nttkafkanp003.celcom.com.my:9093 --from-beginning --consumer-property group.id=demo_group --consumer.config /home/kafka_adl/kafka_backup/scripts/producer-client-ssl.properties 2>/dev/null;

8: Kafka Security

We will see Kafka authentication and authorization. Also, we will look at Zookeeper Authentication.

➢ What is Apache Kafka Security?

There are several features added in the Kafka community, in release 0.9.0.0. There is a flexibility for their usage also, like either separately or together, that also enhances security in a Kafka cluster.



So, the list of currently supported security measures is mentioned blow

1. By using either SSL or SASL, authentication of connections to Kafka Brokers from clients, other tools are possible. It supports various SASL mechanisms:

    - SASL/GSSAPI (Kerberos) – starting at version 0.9.0.0
    - SASL/PLAIN – starting at version 0.10.0.0
    - SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512 – starting at version 0.10.2.0

2. Also, offers authentication of connections from brokers to Zookeeper. Moreover, it provides encryption of data which is transferring between brokers and Kafka clients or between brokers and tools using SSL, that includes:

    - Authorization of reading/write operations by clients.
    - Here, authorization is pluggable and supports integration with external authorization services.

- Note: Make sure that security is optional.
- In our cloudera cluster we have implemented sentry authentication for publisher and consumers

Reference - https://docs.cloudera.com/documentation/enterprise/6/6.1/topics/sentry.html

3. Need for Kafka Security

- Basically, Apache Kafka plays the role as an internal middle layer, which enables our back-end systems to share real-time data feeds with each other through Kafka topics.
- Generally, any user or application can write any messages to any topic, as well as read data from any topics, with a standard Kafka setup.
- However, it is a required to implement Kafka security when our company moves towards a shared tenancy model while multiple teams and applications use the same Kafka Cluster, or also when Kafka Cluster starts on boarding some critical and confidential information.

4. Problems: Kafka Security is solving

- There are three components of Kafka Security, Encryption of data in-flight using SSL / TLS
- It keeps data encrypted between our producers and Kafka as well as our consumers and Kafka. However, we can say, it is a very common pattern everyone uses when going on the web. Authentication using SSL or SASL
- To authenticate to our Kafka Cluster, it allows our producers and our consumers, which verifies their identity. It is the very secure way to enable our clients to endorse an identity. That helps well in the authorization.
- In order to determine whether a client would be authorized to write or read to some topic, our Kafka brokers can run our clients against access control lists (ACL).

The End!