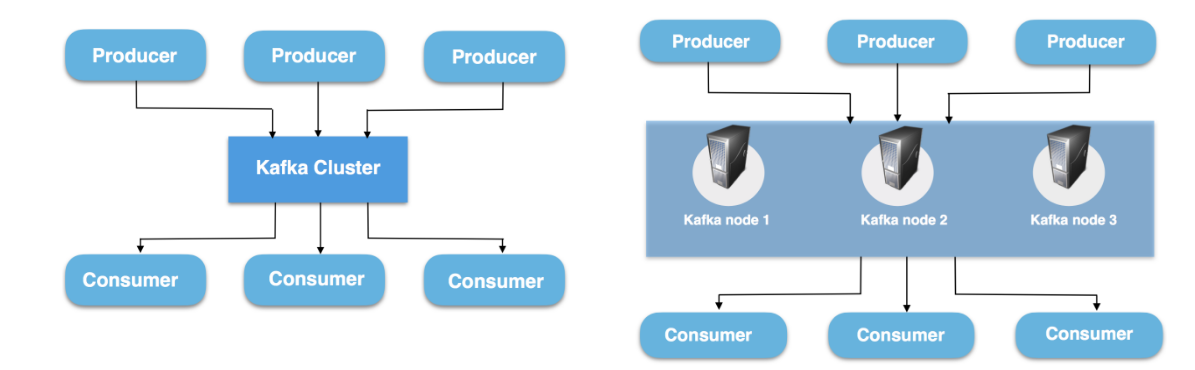


There will be centralized repo, where the entire source sends the data to the centralised repo and the entire consumer will receive the data from this layer, so we get a cleaner connection but we have to make sure that the middle layer (repo) has to fulfil the requirements which were listed as below

1. Delivery guaranteed
2. Distributed Support
3. Performance degrades

Apache Kafka is a publish subscribe messaging system



Basic Components

Kafka Clusters

Kafka brokers form the Kafka clusters

Cluster is a group of computers to share the workload for a common Purpose. Since Kafka is the distributed System so that the cluster has the same meaning for Kafka, each executing one instance for Kafka

Within the Cluster of brokers, one of them is act as the leader which will be elected by zookeeper. It will be manage or control which partition will go to which broker.

Topic will have 5 partitions and the replication is 2. So for a single partition there will be leader and a follower. Whenever there is replication we write the message to the Leader, we read the message from the leader. And the duty of the leader is to replicate whatever content it has to follower.

Incase the Leader Fails , Follower becomes the leader.

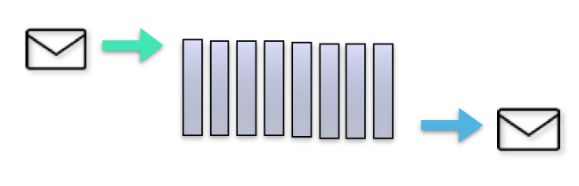
1)Kafka Brokers

It’s a Kafka Server. As per the Name it act as a messaging broker between the Producer and the Consumer .Producer and Consumer wont interact with each other , it will interact with the agent which is nothing but the Kafka Broker.

A single Kafka Server is called Kafka Broker. Broker will receive the message from the producers and it assigns the offset to them and it writes the message to the disks basically it will append the message to the log file.

Whenever consumer wants to read the data, Broker will respond with the message.

1)Messages



A message is a Unit of data. It can include of any information, it could have information about an event happened on your website, or it could be a simple text file.

A message is simply an array of Bytes, as far as Kafka is concerned, message will be serialized while writing in to bytes and de-serialized into message while reading

Message has 2 parts, a) key b) value

2)Batches

In Real Time whenever we write the data in kafka we write the messages in batch and we try to push the message to the kafka in a single data rather than one by one .

Batch Size should be balanced between throughput and latency (Time taken to write the message), if we increase the batch size latency will be increased

3) Schemas

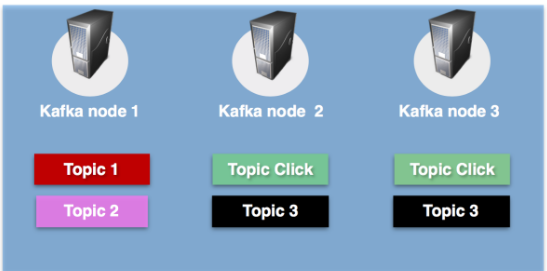
Schema means a Structure how my data looks like, we can

4)Topics and Partitions

Topic is like a directory in the File system / Collections of Messages

Topic is a Feed name to which messages are stored and published. If u wants to send a message you send it into a specific topic and if u want to read a message u read it from a specific topic .Messages are byte array that can store ay objects in any format.

Producer Application writes the data to the topics Consumer Application reads the data from the topic. Messages published to the cluster will stay until a configurable retention period passed by.



Partitions

Topics are split into the configurable no of partitions, as per the Below Image

When we create the Topic we have to make the Decision that how many partitions are needed to store the Topic. Every partition sits on the Single machine.

Offset is a sequence id given to the message as they arrive in a partition. Once the offset is set the number won’t be changed. Kafka Stores the Message in sequential based on the Arrival.

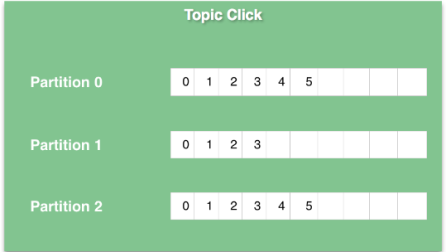
Offsets are local to the Partition; they don’t have any global Offset.

If we want to locate a Message we need to have

1. Topic Name
2. Partition Number
3. Offset

“Topic Click” has 3 Partitions, which can be present in the single machine or it can be present in the Different machine. While writing the Message Producer can choose the partitions.

Every message has an offset, which is on unique identification for the message inside the Partition .Whenever we read the message this offset is maintained which consumer read the message up to which offset. Whenever a new Message is added, the message will be appended to the log file available in the local file system also this offset will be incremented.



Kafka topics are divided into a number of partitions, which contains messages in an unchangeable sequence. Each message in a partition is assigned and identified by its unique offset. As we can see in the pictures - the click topic is replicated to Kafka node 2 and Kafka node 3.

Every partition (replica) has one server acting as a leader, and the rest of them as followers. The leader replica handles all read write requests for the specific partition and the followers replicate the leader. If the leader server fails, one of the follower servers becomes the leader by default. When a producer publishes a message to a partition in a topic, it is forwarded to its leader. The leader appends the message to its commit log and increments its message offset. Kafka only exposes a message to a consumer after it has been committed and each piece of data that comes in will be stacked on the cluster.

5)Kafka Producers

An application that sends the Messages, if we want to send the entire text file, Kafka will send line by line, in terms of Kafka were each line is basically considered as Array of BYTES. Lets if we trigger a query against the database and send the results, where each row will be send as array of Bytes.

Producers create the Message. If we specify the key it will go to the specific partition, if not specify the key, Kafka will see which partitions has less size and push the message. We can write the Custom Producer.

5)Kafka Consumers

An application that reads the data.

Consumers read messages. It will subscribe to one or more topics and reads the messages in order they were published. Consumers will keep track of the message read by using the offset.

Consumer Group –

Many Consumer form a Group to share the workload. 100 of producers send the message to Kafka.

Maximum no of consumer in the Group is the total no of Partition for that Topic.

Kafka won’t allow more that two consumers to read from the same partition simultaneously.

One consumer group has lot of consumers, let’s say we have 8 partition and we have 4 consumers in the consumer group every consumer will read 2 partition. We can distribute the load of reading the data to different machine as long as they belong to the same consumer group

One Partition can be read by single Consumer, It can’t be read by multiple consumers. One Consumer can read multiple partitions.

Every Consumer is maintaining the offset. Consumer 0 is reading the data offset of 6 from the partition 0 is stored in the log of zookeeper.

Producer can be FTP RBDMS

Consumer can be HDFS spark

7)Kafka Retention

Whenever we write the message it will be stored in the local file system. But it will be available up to the retention time. By default retention time is 7 days. We can configure it.

8)Log Compaction

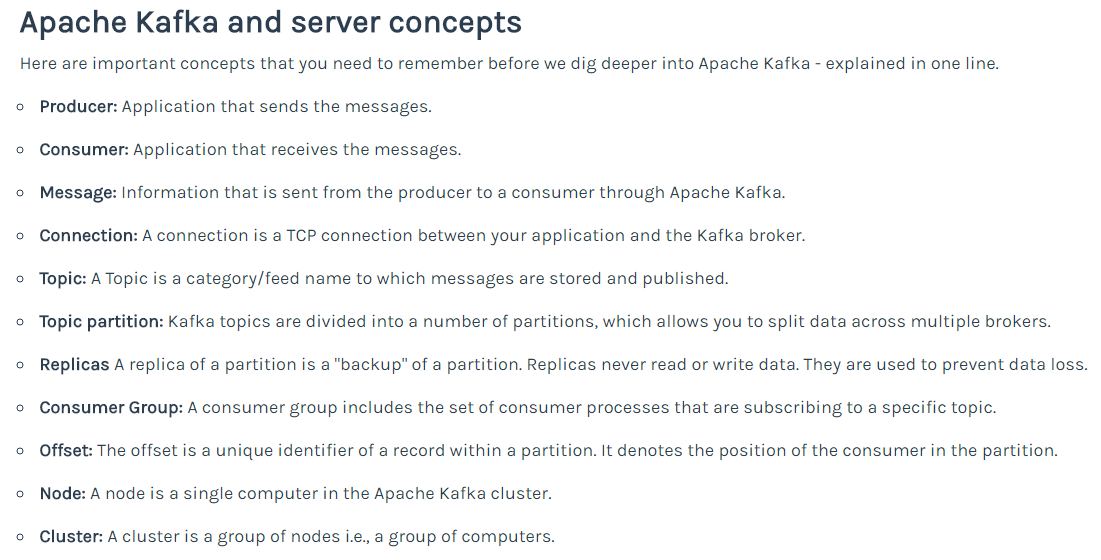
Let’s say for the employee id 123 change his email id over the period of time

123 => 223@gmail

123 => 223@yahoo

123=>223@outlook

We are only interested in the last updated value.log compaction make this possible for us.



Fault Tolerance in Kafka

Kafka will always distribute the data to the cluster. If one or more system got failed in the cluster what happen to the Data?

One simple Solution is to make replication of the data store it in different Machines, so that we can avoid Failures.

We set the Replication Factor at the Topic Level. it applies for all the Partition within the topic.

Normally Replication Factor is 3. This also means we are having 3 copies of the partition in three diff machine for that topic.

How Kafka Makes this Replication / Copy ?

Kafka Follows Leader and Follower approach.

For Every Partition one broker is elected as a Leader, it will take care of all Client Interaction. (i.e) Producer writes the data to the Leader, leader store that result in its local Machine and send back the ACK Signal. Similarly when the consumer wants to read the data consumer send the Request to Leader, it’s the Leader responsibilities to send the requested data back to the consumer.

Kafka Will identify two more Followers to store the copy , these two will store the data from the Leader , they wont communicate with the Producer and Consumer.

Kafka Sever Properties

broker.id => Unique Identification to each broker

Listeners=> it will use this Port to Communicate to the Producers and Consumers if we use multiple cluster no need to Change it, if we are using one node we have to change otherwise all the brokers reading and writing at the same port number

log.dir => to store the Logs

zookeeper.connect=> connection string , zookeeper is the connecting link for the Brokers to from a cluster

Delete.topic.enable => by default its false, so we can’t delete topic unless we set this property

Auto.create.topic.enable=> if we enable this producer will create a topic and sent to it

Default.replication.factor=> default value is 1 ,

Num.partitions=> default values is 1

Log.retention.ms=> Kafka is not a Database, it’s a message Broker Once send we have to clear the data , Default time is 7 days after that message will be deleted

Log.retention.bytes=> we can specify the Space , if we set this property value to 1GB then Kafka will trigger the Clean up activity when the size of the partition size is greater than 1 GB

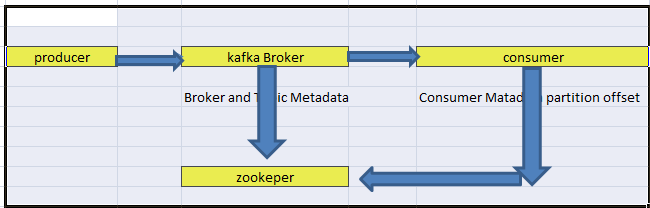
**Kafka Use Cases**

**Installation**

Download the Tar file

Cd /home/Downloads tar xzf confluent.tar mv confluent kafka export KAFKA\_HOME=/home/kafka

Kafka uses zookeeper to store the metadata information about the Kafka cluster as well as the client details.



**Starting the Zookeeper**

./bin/zookeeper-server-start ./etc/kafka/zookeeper.properties

**Broker**

./bin/kafka-server-start ./etc/kafka/server.properties

**Create Topics**

kafka-topics.sh --create \

--zookeeper nn01.itversity.com:2181,nn02.itversity.com:2181,rm01.itversity.com \

--replication-factor 1 \

--partitions 1 \

--topic kafkadg

**Producer**

# Command to produce messages, start typing after running this kakfa-console-producer command

# The messages will be stored in topic kafkadg on the host where brokers are running

kafka-console-producer.sh \

--broker-list nn01.itversity.com:6667,nn02.itversity.com:6667,rm01.itversity.com:6667 \

--topic kafkadg

**CONSUMER**

# Open another shell and then run kafka-console-consumer command to see streaming messages

kafka-console-consumer.sh \

--bootstrap-server nn01.itversity.com:6667,nn02.itversity.com:6667,rm01.itversity.com:6667 \

--zookeeper nn01.itversity.com:2181,nn02.itversity.com:2181,rm01.itversity.com \

--topic kafkaesak \

--from-beginning

To see the key

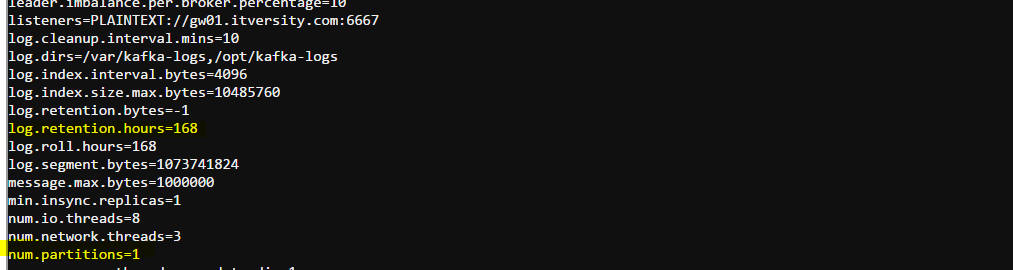
--property print.key =true

**Zookeeper Properties**

dataDir -> it will store the metadata in local directory

client Port-> 2181

**Broker ===> server.properties**

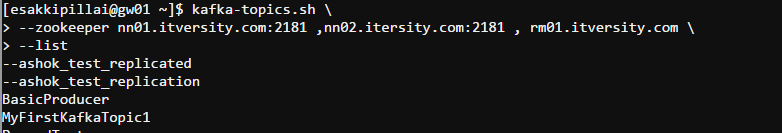


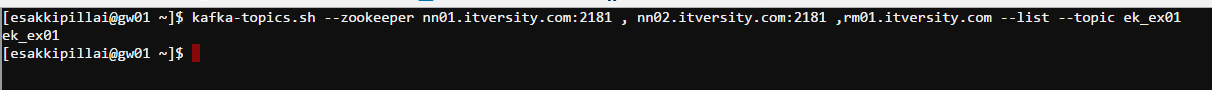
Kafka CLI Commands

1. Start the zookeeper
2. Start the Kafka server
3. List the topics
4. Create a topic
5. Describe a topic
6. Publish message to a topic
7. Consume message
8. Add more partitions to the topic
9. Delete a topic
10. Add Configuration to the topic
11. Remove the configuration from the topic

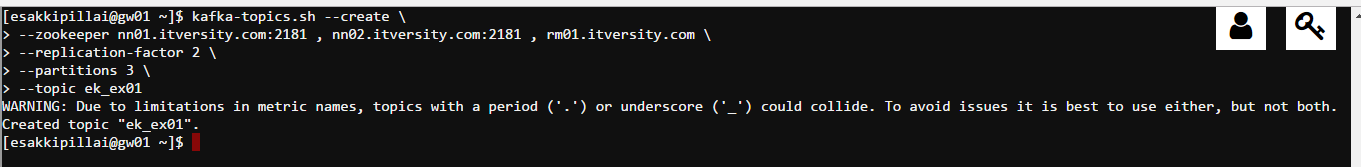
**Scenario 1:- Basic Kafka CLI Commands**

3. List the Topics

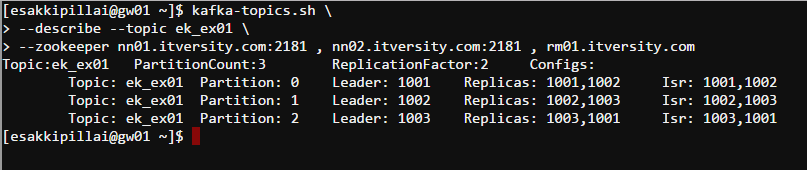




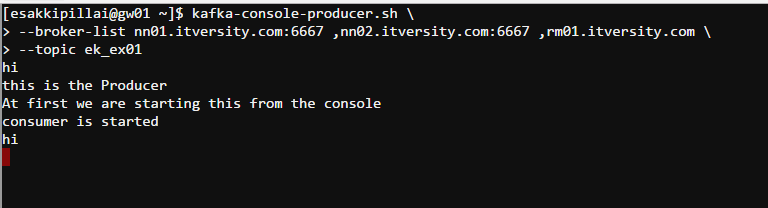
4. Create the Topic



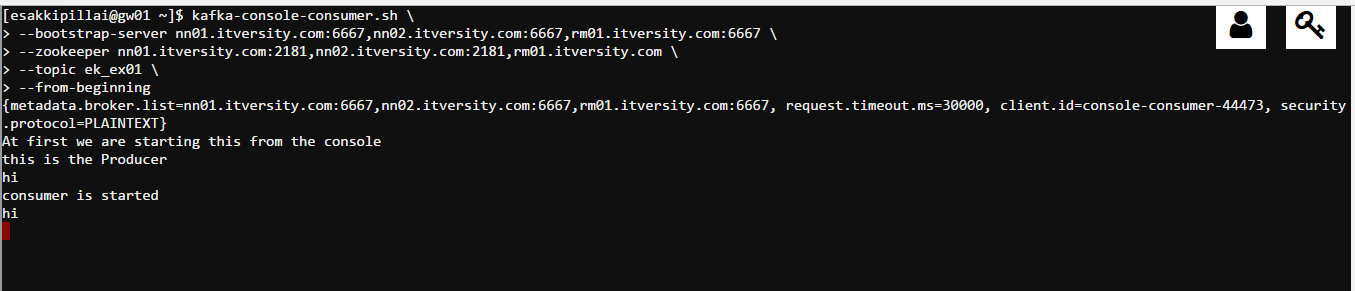
5.Describe the topics



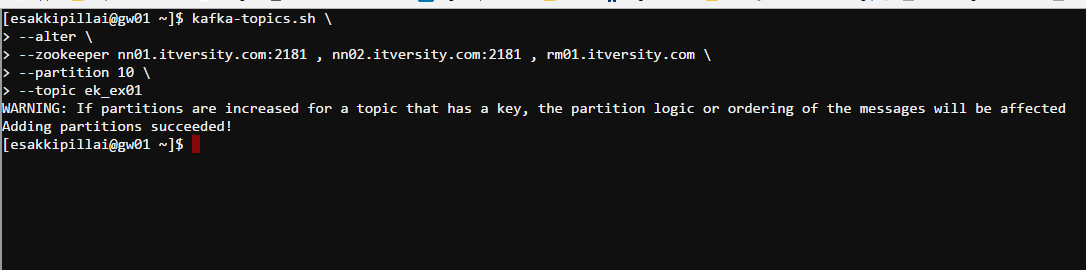
6.Publish the Message

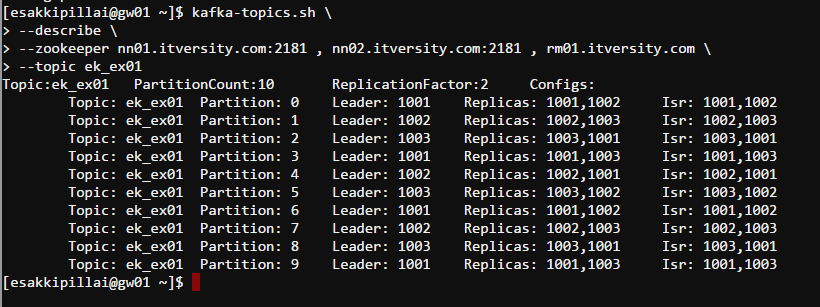


7.Consumer

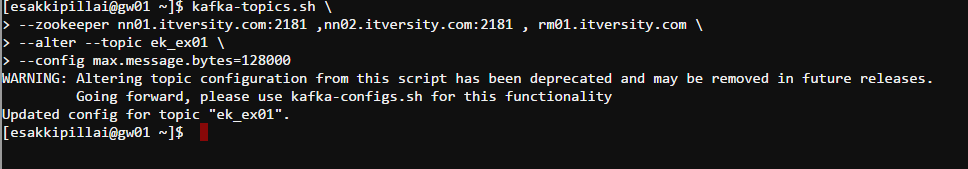


8.Add More Partition to the Topic

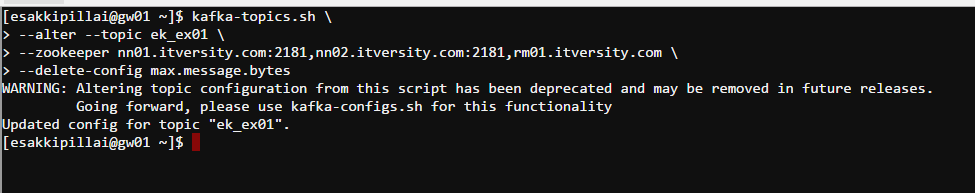




10.Add Configuration to the Topic



11 Remove the configuration from the topic



**Scenario 2 :- Flume And Kafka**

kafka--> flume -> hdfs

# this demo is for kafka Flume

# Describe the Agent

a1.sources=kafkasource

a1.sinks=hdfssink

a1.channels=hdfschannel

# Kafka Source

a1.sources.kafkasource.type=org.apache.flume.source.kafka.KafkaSource

a1.sources.kafkasource.zookeeperConnect=nn01.itversity.com:2181

a1.sources.kafkasource.topic=ek\_ex02

a1.sources.kafkasource.groupid=flume

a1.sources.kafkasource.interceptors = i1

a1.sources.kafkasource.interceptors.i1.type = timestamp

a1.sources.kafkasource.kafka.consumer.timeout.ms = 100

# Channel

a1.channels.hdfschannel.type = memory

a1.channels.hdfschannel.capacity = 10000

a1.channels.hdfschannel.transactionCapacity = 1000

#Describe the hdfs sink

a1.sinks.hdfssink.type = hdfs

a1.sinks.hdfssink.hdfs.path =/user/esakkipillai/flumeKafka/%{topic}/%y-%m-%d

a1.sinks.hdfssink.hdfs.rollInterval = 120

a1.sinks.hdfssink.hdfs.rollSize = 0

a1.sinks.hdfssink.hdfs.rollCount = 0

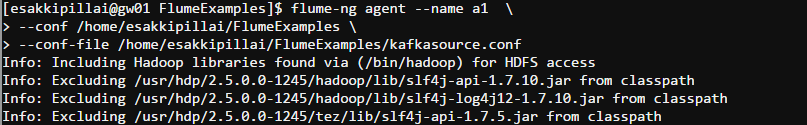
a1.sinks.hdfssink.hdfs.fileType = DataStream

#Combine the sources

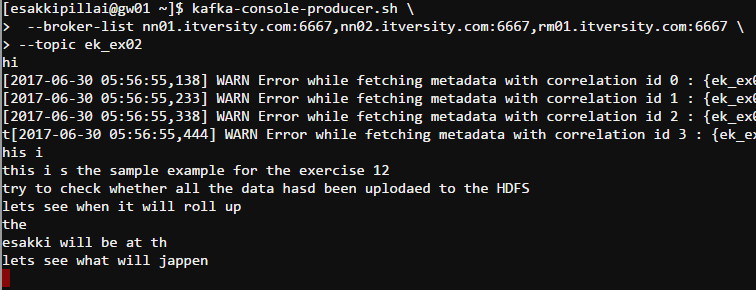
a1.sources.kafkasource.channels=hdfschannel

a1.sinks.hdfssink.channel = hdfschannel

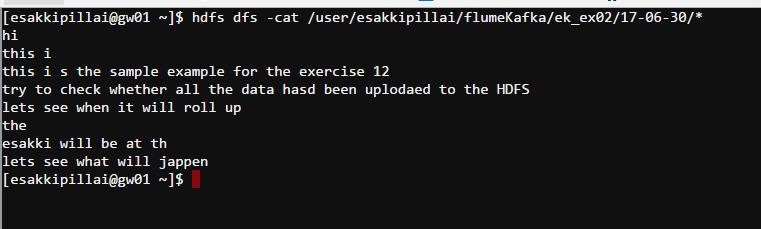
Start the Flume Agent



Start the Kafka Console Producer as of Topic ek\_ex02



Check the Messages at the Hdfs



Kafka as A Sink

### The Following Configuration is used for the Flume Kafka Hdfs

# Describe the Agent Here

ag1.sources = execsource

ag1.sinks = hdfssink kafkasink

ag1.channels = kafkachannel hdfschannel

#Desccribe the Source

ag1.sources.execsources.type = exec

ag1.sources.execsources.command = tail -F /opt/gen\_logs/logs/access.log

#Describe the Hdfssink

ag1.sinks.hdfssink.type= hdfs

ag1.sinks.hdfssink.hdfs.path= /user/esakkipillai/flumeKafka

ag1.sinks.hdfssink.hdfs.filePrefix = execsource-%d-%m-%y

ag1.sinks.hdfssink.hdfs.fileSuffix=.txt

ag1.sinks.hdfssink.hdfs.useLocalTimeStamp=true

ag1.sinks.hdfssink.hdfs.rollInterval = 120

#Describe the kafkaSink

ag1.sinks.kafkasink.type = org.apache.flume.sink.kafka.KafkaSink

ag1.sinks.kafkasink.brokerList = nn01.itversity.com:6667

#Describe the File Channel

ag1.channels.hdfschannel.type = FILE

#Describe the memory Channel

ag1.channels.kafkachannel.type = memory

ag1.channels.kafkachannel.transactionCapacity=100

ag1.channels.kafkachannel.capacity=1000

# Bind the Components

ag1. sources.execsources.channels=kafkachannel hdfschannel

ag1.sinks.hdfssink.channel = hdfschannel

ag1.sinks.kafkasink.channel = kafkachannel

**KAFKA API**

**Produce API**

Custom Producer

**Step 1:** We have to specify the Producer config Properties

To Create a Kafka Producer we need to define the 3 mandatory properties

1. Bootstrap server - we are telling the address of the Kafka Broker
2. Key serializer – Specifying the Key Serialization
3. Value serializer - Specifying the Value Serialization

//Configure the Producer

Properties configProperties = new Properties(); configProperties.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG,"localhost:9092"); configProperties.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG,"org.apache.kafka.common.serialization.ByteArraySerializer"); configProperties.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG,"org.apache.kafka.common.serialization.StringSerializer");

**Step 2 :** - Creating the Producer Object

We just need to create new instance of the class **Kafka Producer.**

Producer<String, String> producer = new kafkaProducer<>(configProperties)

Val producer = new KafkaProducer[String,String](props)

**Step 3 :** Create the Producer Record

String topicName = "AsynchronousProducerTopic";

String key = "Key1";

String value = "Value-1";

ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);

Step 4 :- Send the Message

producer.send(record)

3 ways to send the message

1. Fire and forget
2. We Fire the Message and Forget it we don’t care whether it receives or not.
3. This Method has high Throughput, sometimes data might get lost.
4. Producer.send(data)
5. Synchronous send
   1. We send the Message1 and we wait for the ACK for the message1 we have sent. Once we receive it we sent the message2.
   2. In this Method the wait time is more based on the network traffic
   3. RecordMetadata metadata = producer.send(record).get();
6. Asynchronous send
   1. We send the Message without wait for the ACK of message1 we are sending the message2.
   2. In this method we are achieving the high throughput
   3. We can sent up to a particular no of message without the ACK based on the property “”

producer.send(record, new MyProducerCallback());

class MyProducerCallback implements Callback{

@Override

public void onCompletion(RecordMetadata recordMetadata, Exception e) {

if (e != null)

System.out.println("AsynchronousProducer failed with an exception");

else

System.out.println("AsynchronousProducer call Success:");

}

}

Custom Consumers API

Consumer Groups: - used to read and process the data in parallel.

In order to distribute the Consumer Load, we have more no of Machine which belongs to the same Consumer group

Increase the number of consumers will distribute the load, Consumer 1 doesn’t have any idea about consumer 2 or consumer 3

1 consumer can read the message from multiple partitions

How to read the data Parallel in a single Application (Topic)

We create multiple consumers in a single consumer group. Kafka allows one Consumer to read the data from a Single Partition. Consumers don’t share the Partition with other consumer. NO of Partitions in the Topic is the higher limit of the Consumer. if we have 4 partition and we have 5 consumers then 5th consumer won’t read anything and Kafka wont report it.

How Consumer enters / exist the group?

Let’s say we add the new Consumer to the Consumer Group whom will assign Partitions to the New Consumer? What will happen when one Consumer fails?

Group co coordinator will control all these, one of the Kafka Broker will be elected as group Coordinator.

When consumer wants to join the Group it sends the request to the Group Coordinator, The first consumer joins the group becomes the leader, next consumers become the members of the group

Coordinator will manage the list of group members, Every Time list is modified based on the joining and existing of consumer. Based on this Info it(Coordinator) will trigger the rebalance activity. Since we have new members and we have to assign the partition to the new members.

Leader is responsible for rebalance activity

Leader will take the list of current members and assign the partitions and send back to the group Coordinator. Then Coordinator communicates back to the member.

During the rebalance activity, none of the consumer members can’t read the data from Kafka.

**Creating the Kafka Consumer**

1. Create the Property Object and assign the below important property
   1. Bootstrap.servers
   2. Key.deserializer
   3. Value.deserializer
   4. Group.id => Specify the Consumer group Name
2. Create Kafka Consumer Object and subscribe the Topic Name
   1. KafkaConsumer<String,String> consumer = new KafkaConsumer<>(props)
   2. Consumer.subscribe(Arrays.asList(topicName))
3. Poll the record , poll method handles all the heartbeat rebalancing connect to the Group Coordinator - join the group
   1. ConsumerRecord<String,String> records = consumer.poll(100)

**Offset Management**

Kafka has two set of Offsets

* Current Offset
  + When we send the poll method Kafka will send the first 20 messages, and the offset moves to the 21 message. This integer number to maintain the current position of the Consumer. So the Consumer won’t get the same record again because of this offset.
* Committed Offset
  + This offset is the Position that is already processed by the consumer
  + Up to this position data has been committed to the Consumer.
  + If we read the data up to 20 messages and send the New Request to Kafka so now committed offset is at position 20 and Current offset is at position 30

Partition rebalancing:-

Consumers send the heart beat to the group coordinator (one of the machine from the Kafka cluster). If the group coordinator doesn’t receive any heartbeat for a long time from a consumer, then it will consider as inactive and its session is timed out and rebalancing is triggered.

How to Commit an Offset:-

* Auto Commit
  + it’s the easiest method we can control it by using the below property
    - Enable.auto.commit => By default it is true
    - Auto.commit.intervals.ms => By default it is 5 sec
* Manual Commit
  + Commit sync
  + Commit Async

How to Commit a Particular Offset?

How to know that a rebalance is triggered?

Rebalance Listener

**Schema Evaluation**

* Define an Avro Schema for your message record
* Generate the source code for your avro schema
* Create a producer and use kafkaAvroSerializer
* Create A consumer and use KafkaAvroDeserializer

Custom Scala Code

[**ConsumerExample.scala**](https://gist.github.com/fancellu/f78e11b1808db2727d76#file-consumerexample-scala)

|  |  |
| --- | --- |
|  | import java.util |
|  |  |
|  | import org.apache.kafka.clients.consumer.KafkaConsumer |
|  |  |
|  | import scala.collection.JavaConverters.\_ |
|  |  |
|  | object ConsumerExample extends App { |
|  |  |
|  | import java.util.Properties |
|  |  |
|  | val TOPIC="test" |
|  |  |
|  | val props = new Properties() |
|  | props.put("bootstrap.servers", "localhost:9092") |
|  |  |
|  | props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer") |
|  | props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer") |
|  | props.put("group.id", "something") |
|  |  |
|  | val consumer = new KafkaConsumer[String, String](props) |
|  |  |
|  | consumer.subscribe(util.Collections.singletonList(TOPIC)) |
|  |  |
|  | while(true){ |
|  | val records=consumer.poll(100) |
|  | for (record<-records.asScala){ |
|  | println(record) |
|  | } |
|  | } |
|  | } |

[**ProducerExample.scala**](https://gist.github.com/fancellu/f78e11b1808db2727d76#file-producerexample-scala)

|  |  |
| --- | --- |
|  | object ProducerExample extends App { |
|  |  |
|  | import java.util.Properties |
|  |  |
|  | import org.apache.kafka.clients.producer.\_ |
|  |  |
|  | val props = new Properties() |
|  | props.put("bootstrap.servers", "localhost:9092") |
|  |  |
|  | props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer") |
|  | props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer") |
|  |  |
|  | val producer = new KafkaProducer[String, String](props) |
|  |  |
|  | val TOPIC="test" |
|  |  |
|  | for(i<- 1 to 50){ |
|  | val record = new ProducerRecord(TOPIC, "key", s"hello $i") |
|  | producer.send(record) |
|  | } |
|  |  |
|  | val record = new ProducerRecord(TOPIC, "key", "the end "+new java.util.Date) |
|  | producer.send(record) |
|  |  |
|  | producer.close() |
|  | } |

Every Topic has a Partition, Every Partition has a Message, Every Message has a Offset Key and Value.

Consumer Produces a Message to the Kafka to special \_consumer\_offsets topics, with the committed offset for each partition.

If the committed offset is smaller than the offset of the last message and in case of any failure it will start from the committed offset and re read the data.

If we enable auto commit, poll method will commit the highest offset value.

Kafka AVRO SCHEMA REGISTRY

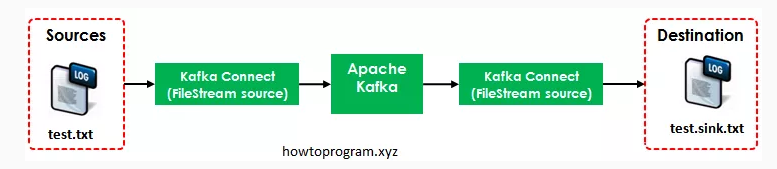
AVRO :- language neutral data serialization framework

AVRO PRODUCER

**Apache Kafka Connect**

As per the Kafka 0.9 , ***Apache Kafka Connect  has been introduced, which simplifies the connection between Kafka and other systems***

The example is used to demo how to use Kafka Connect to stream data from source which is  file **test.txt** to destination which is also a file, **test.sink.txt.**



Configure mysql connector jar in following location  
/opt/confluent/share/java/kafka-connect-jdbc/mysql-connector-java-5.1.38-bin.jar

Create quickstart-mysql.properties in /opt/confluent/etc/kafka-connect-jdbc/  
with following content

name=test-mysql-jdbc-autoincrement

connector.class=io.confluent.connect.jdbc.JdbcSourceConnector

tasks.max=1

**connection**.url=jdbc:mysql://127.0.0.1/bigdata?user=hduser&amp;password=\*\*\*\*\*\*

mode=incrementing

incrementing.**column**.name=id

topic.prefix=test-mysql-jdbc-