Case Study: Telecom Pipeline Domain: Telecom

There are two large obstacles in collecting metadata from a network as large as India's Big Telecom operator:

a. ***Transporting the sheer volume of data and***
b. ***Processing it before the data no longer accurately reflects the state of the network.***

Fortunately, combining Apache Flume and Apache Kafka using the Kafka pattern provides a means to move data into the Hadoop cluster and readily scale the pipeline to address both transient and persistent spikes in data volume.

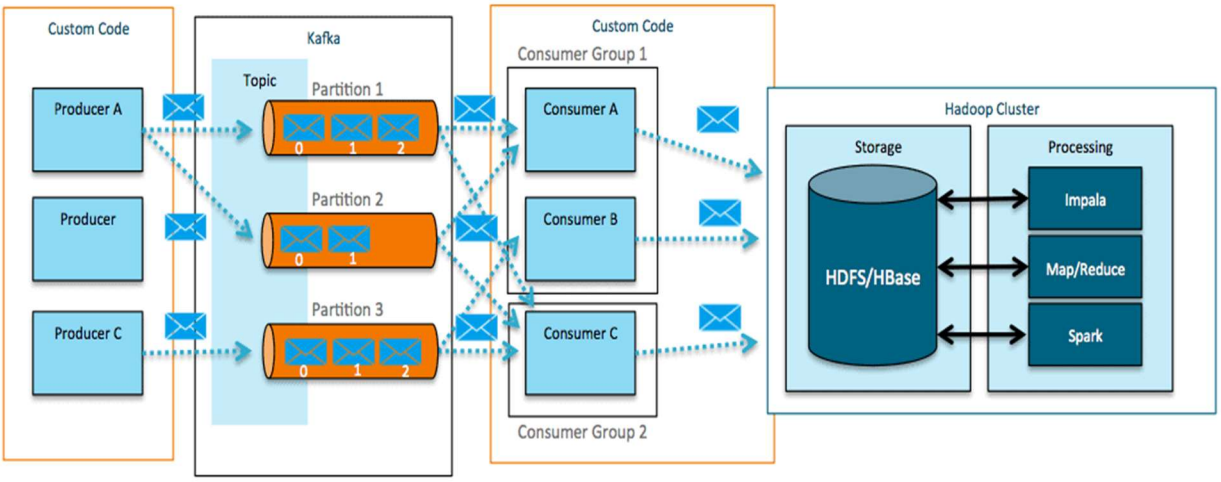
Company is planning to deploy Flume and Kafka across the network in a geographically distributed architecture that achieves scale and resilience, having been tuned from around 10,000 events per second on initial deployment to 1,000,000 events per second using a three-node Kafka cluster.

Tasks:

You are part of the Telecom Operator's R&D team, which is required to perform a quick POC on the Kafka Flume pipeline to persist data to HDFS and analyze the data through spark streaming.

Solution :

"Flafka : Apache Flume Meets Apache Kafka for Event Processing"



The above is the diagram showing how the Kafka + Custom Producer/Consumer can take the Data from Data Source to Data Destination.

So , here comes the role of Flume.

Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of data from ***many different sources to a centralized data store.***

Flume-Kafka integration offers the following functionality that Kafka, absent custom coding, does not.

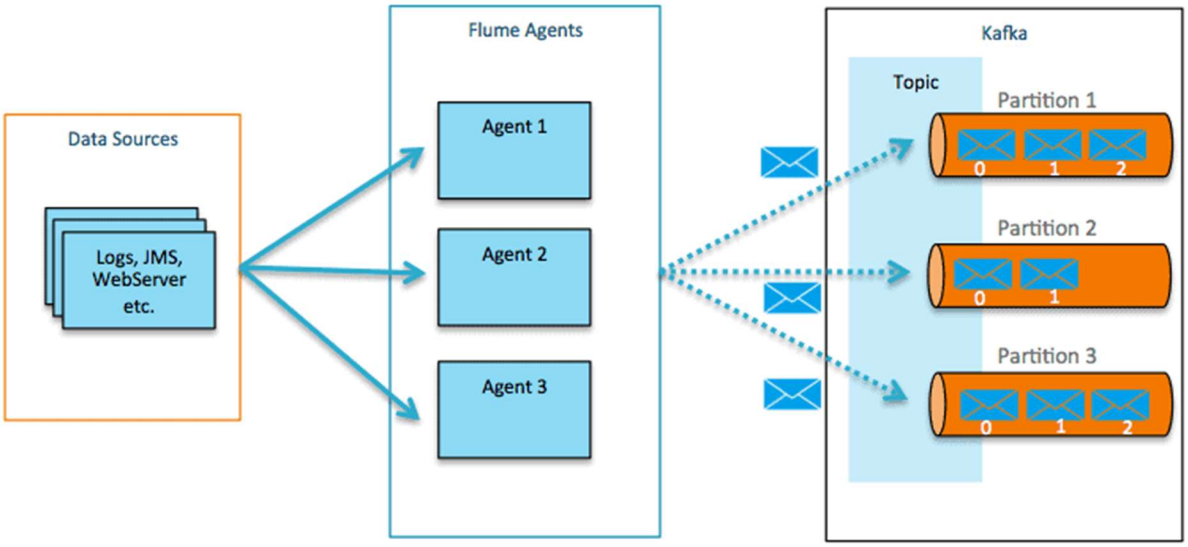
Producers – Use Flume sources to write to Kafka

Consumers – Write to Flume sinks reading from Kafka

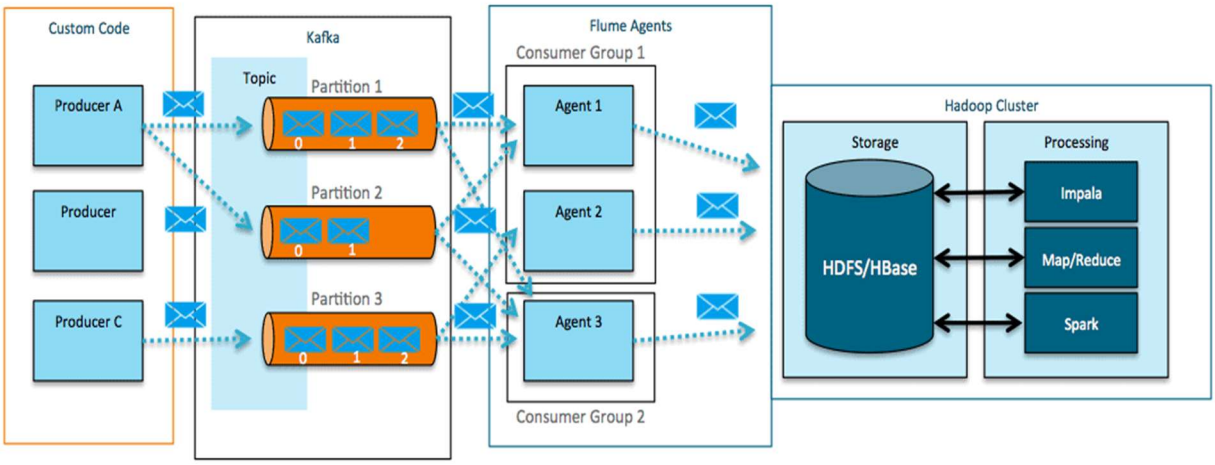
A combination of the above --- In-flight transformations and processing.

Best practice of using the Flume with Kafka.

With Flume as a Producer



With Flume as a Consumer:



In conclusion

As a best practice to integrate Kafka with Flume for Streaming heavy velocity data, Flafka provides more flexibility for the data pipeline and can achieve distributed ingestion pipeline that, with careful tuning, can ingest more than 1 million events per second

In the solution of our case study :

Kafka and Flume integration is needed wherein the stream of data is put into the Kafka topic with high speed and then Flume act as a Consumer who writes to Sink (HDFS in our case) . Here the Flume acts as Consumer and stores in HDFS. So no custom code for integration. And our objectives are achieved.

a. ***Transporting the sheer volume of data and (Large Volume Data Transfer)***
b. ***Processing it before the data no longer accurately reflects the state of the network. (Fast / Low latency in the streaming Data)***

Kafka set up:

1. **Creating the topic**

kafka-topics --create --zookeeper ip-20-0-21-161.ec2.internal:2181 --replication-factor 2 --partitions 3 --topic flume.txn

Note that Topic has been created , Replication factor of 2 for high availability of Data and then 3 partitions of the same topic which can be in different brokers so that three consumers can be run in parallel (in our case the 3 agents can run in parallel to extract the data from 3 partitions)

Flume Set Up:

2. **Flume Configuration**

```
# Sources, channels, and sinks are defined per
# agent name, in this case flume1.
flume1.sources  = kafka-source-1
flume1.channels = hdfs-channel-1
flume1.sinks    = hdfs-sink-1

# For each source, channel, and sink, set
# standard properties.
flume1.sources.kafka-source-1.type = org.apache.flume.source.kafka.KafkaSource
flume1.sources.kafka-source-1.zookeeperConnect = ip-20-0-21-161.ec2.internal:2181
flume1.sources.kafka-source-1.topic = flume.txn
flume1.sources.kafka-source-1.batchSize = 100
flume1.sources.kafka-source-1.channels = hdfs-channel-1

flume1.channels.hdfs-channel-1.type   = memory
flume1.sinks.hdfs-sink-1.channel = hdfs-channel-1
```

flume1.sinks.hdfs-sink-1.type = hdfs

flume1.sinks.hdfs-sink-1.hdfs.writeFormat = Text

flume1.sinks.hdfs-sink-1.hdfs.fileType = DataStream

flume1.sinks.hdfs-sink-1.hdfs.filePrefix = test-events

flume1.sinks.hdfs-sink-1.hdfs.useLocalTimeStamp = true

flume1.sinks.hdfs-sink-1.hdfs.path = /user/edureka_960126/flume_kafka

flume1.sinks.hdfs-sink-1.hdfs.rollCount=100

flume1.sinks.hdfs-sink-1.hdfs.rollSize=0

Other properties are specific to each type of

source, channel, or sink. In this case, we

specify the capacity of the memory channel.

flume1.channels.hdfs-channel-1.capacity = 10000

flume1.channels.hdfs-channel-1.transactionCapacity = 1000

Command to run the flume agent:

./flume-ng agent --conf conf --conf-file /mnt/home/edureka_960126/flume_http.conf --name httpagent -Dflume.root.logger=INFO,console --plugins-path /mnt/home/edureka_960126

3. Generation of events to the Kafka Source through Kafka Console for testing purpose of our pipeline

kafka-console-producer --topic flume.txn --broker-list ip-20-0-31-210.ec2.internal:9092, ip-20-0-31-221.ec2.internal:9092, ip-20-0-31-221.ec2.internal:9092

Important Learnings :

Batch size can be declared in one of two ways:

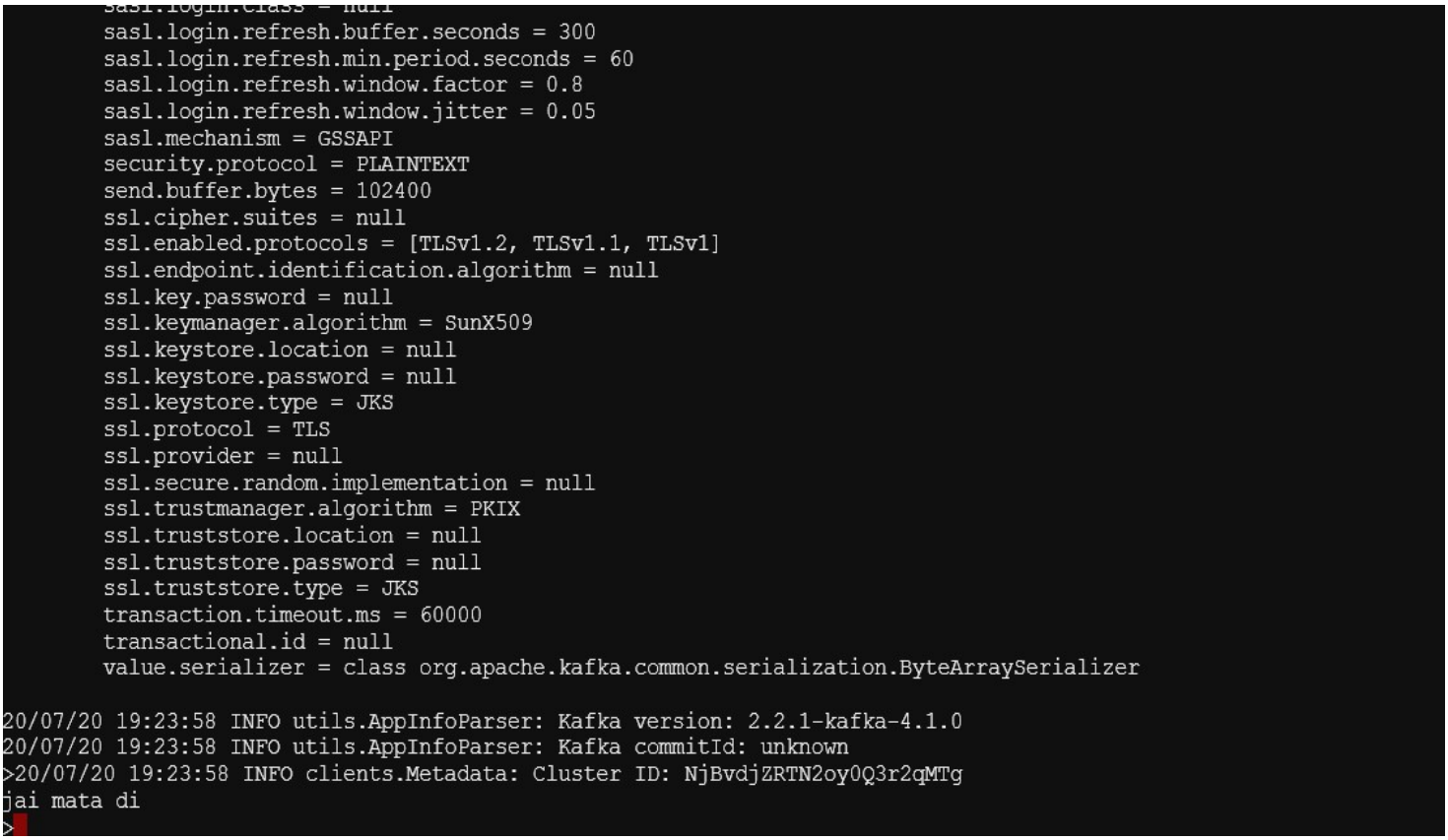
1. by specifying the size of the batch in terms of number of events (batchSize), or
2. as a number of milliseconds (batchDurationMillis) to wait while receiving events from Kafka.

In this manner, latency-based SLAs can be maintained for lower volume flows.

In our case simply we have used the batch size of 100 only but in case velocity is high we can increase the batch size and decrease the Batch duration time where the Flume consumer wait to pull data out of the Kafka.

Output Screen shots :

1. Message transfer/event generation



Note that here the kafka console producer is running and the message/event has been generated and is being transferred to HDFS as shown ahead in event file starting test-events as mentioned in the flume configuration

2. HDFS stored events

```
ip-20-0-41-62 login: edureka_960126
Password:
Last login: Mon Jul 20 19:23:27 on pts/17
[edureka_960126@ip-20-0-41-62 ~]$ hdfs dfs -ls /user/edureka_960126/flume_kafka/
Found 10 items
-rw-r--r--   3 edureka_960126 hadoop        140 2020-07-20 05:25 /user/edureka_960126/flume_kafka/event-.1595222683059
-rw-r--r--   3 edureka_960126 hadoop        128 2020-07-20 05:31 /user/edureka_960126/flume_kafka/event-.1595223038741
-rw-r--r--   3 edureka_960126 hadoop        144 2020-07-20 05:36 /user/edureka_960126/flume_kafka/event-.1595223386413
-rw-r--r--   3 edureka_960126 hadoop        144 2020-07-20 05:40 /user/edureka_960126/flume_kafka/event-.1595223620059
-rw-r--r--   3 edureka_960126 hadoop        144 2020-07-20 05:47 /user/edureka_960126/flume_kafka/event-.1595224000300
-rw-r--r--   3 edureka_960126 hadoop        164 2020-07-20 05:48 /user/edureka_960126/flume_kafka/event-.1595224085681
-rw-r--r--   3 edureka_960126 hadoop         30 2020-07-20 05:59 /user/edureka_960126/flume_kafka/event-.1595224727528
-rw-r--r--   3 edureka_960126 hadoop        140 2020-07-20 04:21 /user/edureka_960126/flume_kafka/events-.1595218843195
-rw-r--r--   3 edureka_960126 hadoop        128 2020-07-20 05:16 /user/edureka_960126/flume_kafka/events-.1595222172724
-rw-r--r--   3 edureka_960126 hadoop         12 2020-07-20 19:25 /user/edureka_960126/flume_kafka/test-events.1595273071552
[edureka_960126@ip-20-0-41-62 ~]$ hdfs dfs -cat /user/edureka_960126/flume_kafka/test-events.1595273071552
jai mata di
[edureka_960126@ip-20-0-41-62 ~]$
```