**STREAMING DATA ETL with KAFKA and SPARK**

**Setup gen\_logs**

One of the common use case for streaming pipelines is to understand visitor traffic. [gen\_logs](https://github.com/dgadiraju/gen_logs.git) is a simple Python based application which can be used to generate log messages as if some one is accessing an eCommerce website.

* Identify directory /opt/
* Copy gen\_logs directory to /opt – sudo mv -f gen\_logs /opt
* Run sudo chown -R `whoami` /opt/gen\_logs
* Update PATH in .profile or .bash\_profile or create soft links for the shell programs – export PATH=$PATH:/opt/gen\_logs
* Either restart shell or run .profile/.bash\_profile script
* Run start\_logs.sh to start generating web logs
* Run tail\_logs.sh to preview while logs are being generated (Hit ctrl-c to come out)
* Run stop\_logs.sh to stop generating web logs

## Setup Kafka Locally

Let us see how we can set up Kafka locally on Mac or PC or Linux. We can have multi-node cluster as well on one machine with different ports, however it is not good practice for production environment. We will see multi-node cluster setup on simulated environment based on virtual machines later

* For Windows make sure you have Ubuntu setup using subsystem.
* Let us review steps for setup process.
  + Download Kafka binaries (compressed tar ball)
  + Untar and uncompress tar ball (creates a directory)
  + Create soft link pointing to the directory created in last step. This will come handy in case of upgrades.
  + Kafka require Zookeeper to be up and running. We will understand the role of Zookeeper at later point in time.
  + Review the Zookeeper properties and start Zookeeper server.
  + Review Kafka properties and start Kafka server.
* Once you finalize the directory, steps are same irrespective of the platform we are using. We will be using /opt as the directory.
* Go to <https://kafka.apache.org/downloads>
* Choose latest version built with Scala 2.11 (1.0.0)
* Click on the link and go to mirror for kafka\_2.11-1.0.0.tgz. Run this command under /opt – sudo wget http://apache.mirrors.ionfish.org/kafka/1.0.0/kafka\_2.11-1.0.0.tgz (I have already downloaded on my PC)
* Untar the tar ball under /opt sudo tar xzf kafka\_2.11-1.0.0.tgz
* Create soft link with name /opt/kafka – ln -s /opt/kafka\_2.11-1.0.0 /opt/kafka
* Update PATH as part of .profile or .bash\_profile for user kafka and training export PATH=$PATH:/opt/kafka/bin
* Starting Kafka – Kafka requires
  + Zookeeper
  + Kafka Server
* Here are the commands you need to run (provided PATH is updated)
  + Make sure you are connected as kafka
  + Starting zookeeper-server: zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
  + Starting kafka-server: kafka-server-start.sh -daemon "/opt/kafka/config/server.properties"
  + Use -daemon to submit zookeeper and kafka servers in background

## Validating Kafka

* As part of setting up of Kafka, you will get bunch of shell scripts under bin. We have already seen few while starting zookeeper and kafka earlier
* There are several other commands
  + zookeeper-server-stop.sh
  + kafka-server-stop.sh
  + kafka-topics.sh
  + kafka-console-producer.sh
  + kafka-console-consumer.sh
  + There are bunch of other commands as well.
* In actual implementation we will not use these commands to manage topics or produce/consume messages from topic. However these commands come handy for quick validations and troubleshooting certain issues related to topics.
* We will manage topics, publish messages into topic as well as consume messages from topic programmatically, which means we will be using respective APIs in programming languages such as Java, Scala, Python etc.
* Each of these commands use different details to connect to the cluster. kafka-topics.sh uses zookeeper while other 2 uses kafka broker details.
* Create topic: kafka-topics.sh --zookeeper localhost:2181 --create --topic test --partitions 1 --replication-factor 1
* When we create topic, it will create directory under /tmp/kafka-logs (review log.dirs under server.properties) using the name of topic and partition index (e.g.: test-0)
* List topic: kafka-topics.sh --zookeeper localhost:2181 --list
* Publish Messages: kafka-console-producer.sh --broker-list localhost:9092 --topic test
* From other window Consume Messages: kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
* Deleting topic: kafka-topics.sh --zookeeper localhost:2181 --delete --topic test

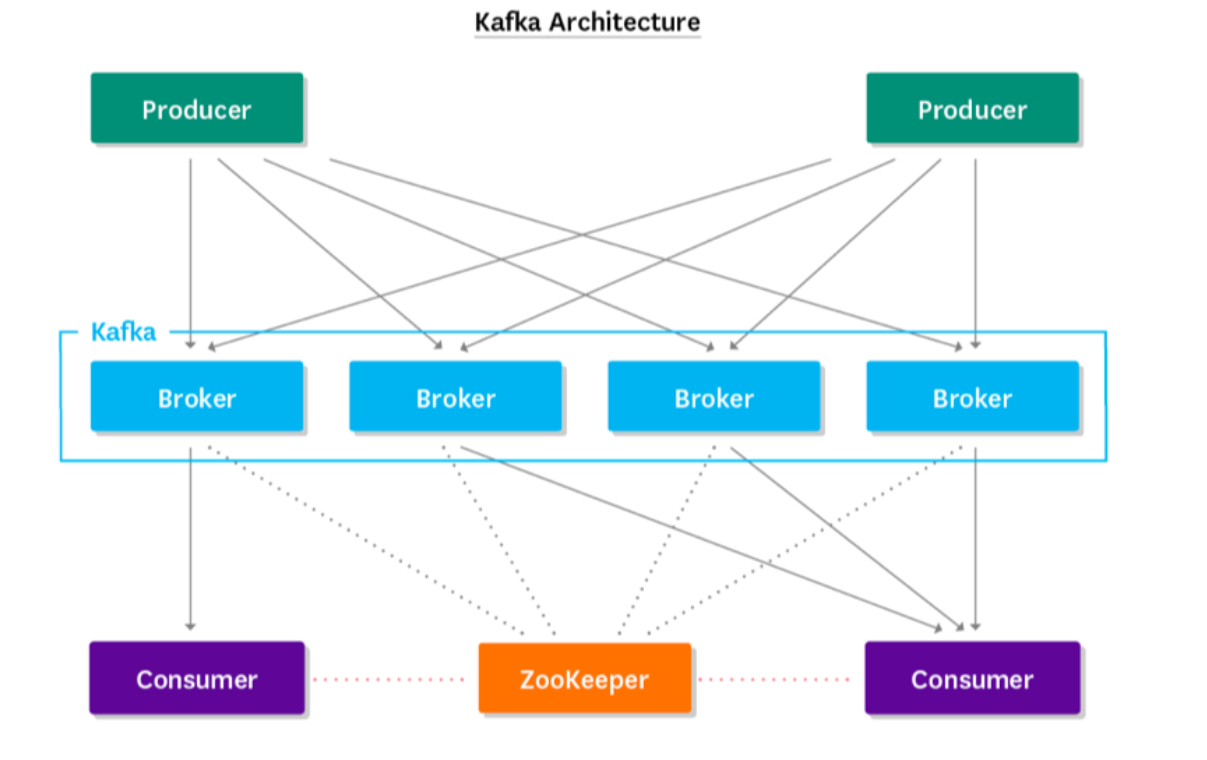
## Validate with gen\_logs traffic

Now let us validate using data generated by gen\_logs application streaming into the topic by using command line approach (available under bin directory of kafka).

* Make sure zookeeper and kafka broker are running (using telnet command)
* Make sure gen\_logs are generating log messages simulating visitor traffic
* We will pipe the output of tail\_logs.sh to kafka-console-producer.sh command and then use kafka-console-consumer.sh to consume message from the topic.
* Topic Details
  + Name: retail
  + Partitions: 3
  + Replication Factor: 1 (as kafka broker is only running on one node, we cannot have replication factor higher than 1)
* Create topic: kafka-topics.sh --zookeeper localhost:2181 --create --topic retail --partitions 3 --replication-factor 1
* List topic: kafka-topics.sh --list --zookeeper localhost:2181
* Publish Messages: tail\_logs.sh|kafka-console-producer.sh --broker-list localhost:9092 --topic retail
* From other window Consume Messages: kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic retail --from-beginning

## Kafka on multi-node cluster

* Topic is a distributed log file
* Producers will log messages to this log file and Consumers will consume messages from this log file

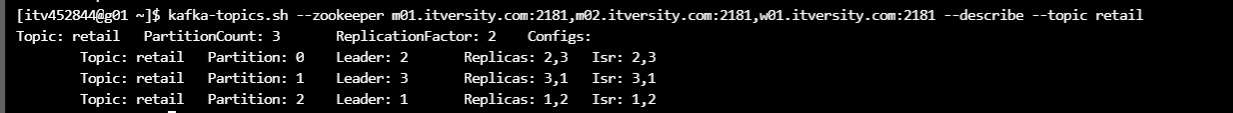


**Creating a topic :**

*kafka-topics.sh --zookeeper m01.itversity.com:2181,m02.itversity.com:2181,w01.itversity.com:2181 --create --topic retail --partitions 3 --replication-factor 2*

**Describing a topic:**

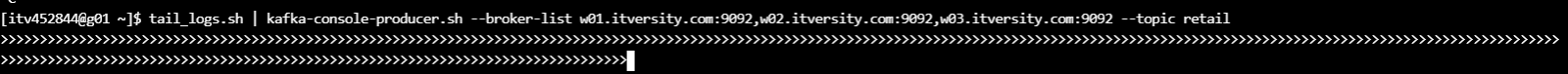
kafka-topics.sh --zookeeper m01.itversity.com:2181,m02.itversity.com:2181,w01.itversity.com:2181 --describe --topic retail



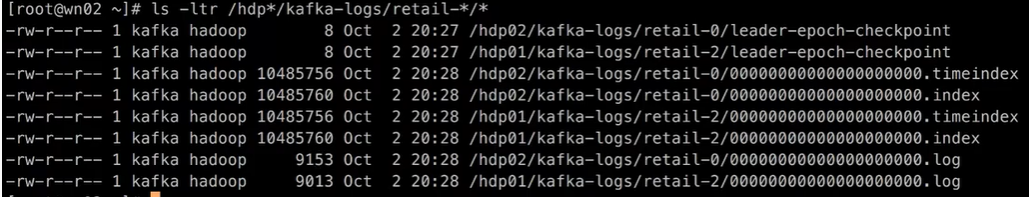
* When you create a topic in a multi node cluster , each and every broker nodes are having the copy of the metadata of this topic.
* So while giving --broker—list values in kafka-console-producer.sh , we don’t need to specify all the brokers. If you give any 1 or more brokers, it will read the metadata for that topic and understand the partitions and respective leaders of the partition and it will facilitate the client to produce the messages to respective partitions by connecting to leader broker

**Re-routing gen\_logs to kafka producer for creating topics:**

Command : *tail\_logs.sh | kafka-console-producer.sh --broker-list w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic retail*



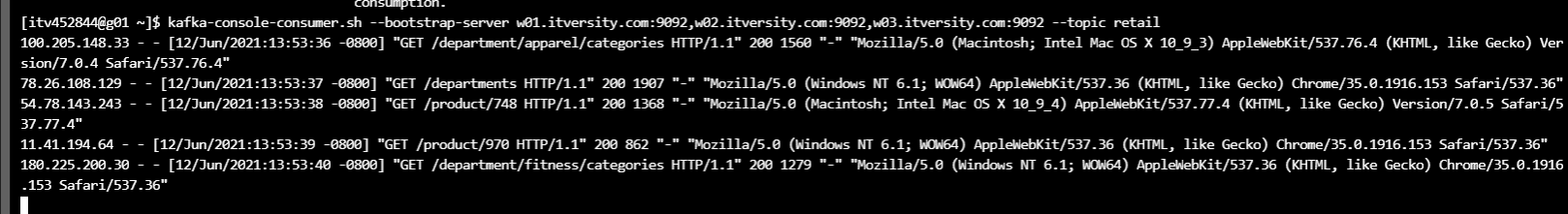
**Output in respective worker nodes will be like:**



**Accessing and viewing generated logs through Kafka consumer:**

**Command:**

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic retail*



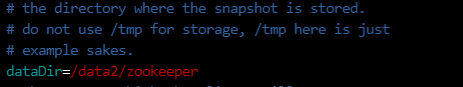
--from-beginning can be added to view messages from beginning

* **Topic**
  + Distributed log file (distribution is achieved via partitions)
  + Each partition is nothing but directory named after topic and appended by partition index
  + In a multinode cluster we use higher replication-factor so that we can have multiple copies of each of the partition for fault tolerance.
  + Each of these copies are managed by brokers. As there will be multiple brokers associated with each partition of a given topic, one of the broker will be designated as leader.
* **Producers** connect to one or more brokers and push messages to topics via leader.
* **Consumers** pull message from topic by polling topic at regular intervals. Each time consumer read messages it need to keep track of offset (can be done using multiple ways)
* **Role of Zookeeper:**
  + Create and manage topics (a topic is nothing but distributed log file with one or more partitions)
  + Zookeeper understand load on the Kafka brokers and take care of creation of partitions as part of nodes on which brokers are running.
  + Zookeeper will also assign leader for each of the partition.
  + Manage consumer offsets
* **Role of Gateway nodes**
  + Developers can troubleshoot issues from gateway nodes
  + Plugins which produce messages to Kafka topic or consume messages from Kafka topic can be deployed on gateway nodes.

## Understanding Zookeeper and Kafka Properties

**zookeeper.properties:**

dataDir 🡪 Path configured for this config parameter in zookeeper.properties hold the metadata related to broker / topic / consumer offsets



clientPort 🡪 The port at which client will connect to zookeeper



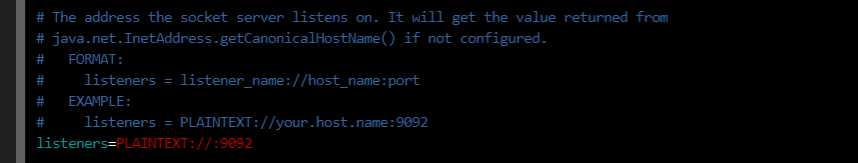
**server.properties:**

broker.id 🡪 It’s a unique integer for each broker. If you have 3 broker nodes in your kafka cluster,

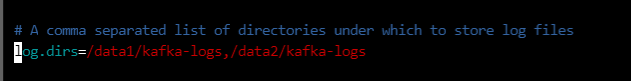
each broker will have this property defined with a unique id



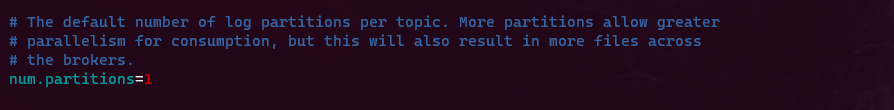
listeners 🡪 Producer and consumer connect to kafka brokers via this port



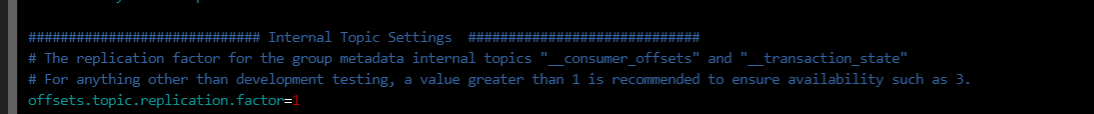
log.dirs 🡪 directories under which kafka topics will be created and stored



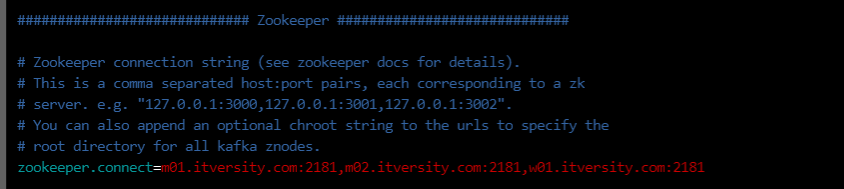
num.partitions 🡪 default number of partitions for a topic



offsets.topic.replication.factor 🡪 default replication factor for topic files (consumer offset)



zookeeper.connect 🡪zookeeper connection string



auto.create.topics.enable 🡪 setting true will auto create the topic if it does not exists

log.retention.hours 🡪 The minimum age of a log file to be eligible for deletion due to age

log.retention.bytes 🡪 A size-based retention policy for logs. Segments are pruned from the log unless the remaining. segments drop below log.retention.bytes. Functions independently of log.retention.hours

log.segment.bytes 🡪 The maximum size of a log segment file. When this size is reached a new log segment will be created.

log.retention.check.interval.ms 🡪 The interval at which log segments are checked to see if they can be deleted according to the retention policies

log.roll.hours 🡪 Setting this property will create a new log segment when the hours defined is reached even if log size has not reached log.segment.bytes value

* For local installation – you can go to $KAFKA\_HOME/config directory and visit zookeeper.propeties and server.properties to review some of the important properties.
* For multinode installation, you will have these files on all the nodes and hence reviewing them via command line is not reliable.
* Whatever management tools you have you should be able to review the properties through their interfaces. In our case we will review the cluster managed by Ambari
* You can go to zookeeper configs and kafka configs to review the properties
* Also let us understand some of the important properties with respect to zookeeper and kafka

## Apache Kafka – Overview

* Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. It was initially developed as an internal product at LinkedIn and was open-sourced and adopted by apache foundation.
* Named after author Franz Kafka
* Salient Features:
  + Highly Scalable (partitioning)
  + Fault Tolerant (replication factor)
  + Low Latency 🡪 messages are getting appended to log file. So achieving low latency
  + High Throughput 🡪 Since distributed in multiple nodes (basically multiple brokers)

## Kafka eco system

Heart of Kafka is topic a distributed and fault tolerant log file. However over a period of time Kafka is evolved into eco system of tools.

* Kafka Connect
* Kafka Streams and Kafka SQL
* Producer and Consumer APIs
* 3rd party plugins to integrate with Flume, logstash, Spark Streaming, Storm, Flink etc.

## 

## Kafka Use cases

As micro services have evolved Kafka become popular to integrate data between different micro services – asynchronous, real time as well as batch.

* **Activity Tracking:** Kafka was originally developed for tracking user activity on LinkedIn
* **Messaging:** Kafka is also used for messaging, where applications need to send notifications (such as emails) to users.
* **Metrics and logging:** Applications publish metrics on a regular basis to a Kafka topic, and those metrics can be consumed by systems for monitoring and alerting.
* **Commit log:** database changes can be published to Kafka and applications can easily monitor this stream to receive live updates as they happen. This changelog stream can also be used for replicating database updates to a remote system.
* **Stream processing:** Kafka can be integrated with stream frameworks such as Spark Streaming, Flink, Storm etc. Users are allowed to write applications to operate on Kafka messages, performing tasks such as counting metrics, transform data, etc.

## Glossary

**Topic:** A topic represent group of files and directories. When we create topic, it will create directories with topic name and partition index. These directories have bunch of files which will actually store the messages that are being produced.

**Publisher or Producer:** Publishers or producers are processes that publish data (push messages) to the log file associated with Kafka topic.

**Subscriber or Consumer:** Subscribers or consumers are processes that read from the log file associated with Kafka topic

#### Kafka Pub Sub Model

## 

**Partition:** Kafka topics are divided into a number of partitions, which contains messages in an unchangeable sequence. This allows for multiple consumers to read from a topic in parallel.

**Leader:**When we create Kafka topic with partitions and replication factor, each partition will have leader. Messages will be first written to the partition on broker which is designated as leader and then copied to rest of followers.

**Replication Factor:** Each partition can be cloned into multiple copies using replication factor. It will facilitate fault tolerance. With replication factor of n on m node cluster (where n <= m), cluster can survive the failure of n-1 nodes at any point in time.

**Broker:** A Kafka cluster consists of one or more servers (Kafka brokers), which are running Kafka. Producers query metadata of each of the topic and connect to leader of each partition to produce messages into Kafka topic. Consumers do the same while consuming messages from the topic.

**Offset:** The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition.

## Scenarios:

#### Single Partition topic – One Consumer

* One producer – Topic with Single partition – One consumer

1. Creating a topic with single partition

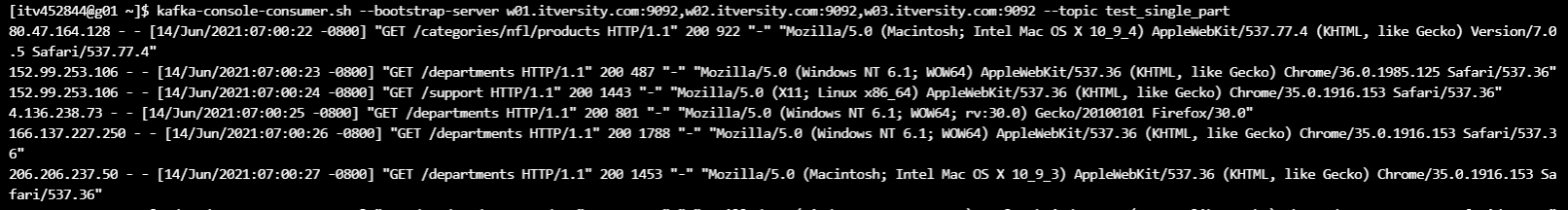
*kafka-topics.sh --zookeeper m01.itversity.com:2181,m02.itversity.com:2181,w01.itversity.com:2181 --create --topic test\_single\_part --partitions 1 --replication-factor 2*

* Publishing messages to above topic from gen\_logs

*tail\_logs.sh | kafka-console-producer.sh --broker-list w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_single\_part*

* Consuming messages using kafka-console-consumer.sh:

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_single\_part*



#### Single Partition Topic – Multiple Consumers

* If consumers are under same consumer group, then two consumers can not consume the messages from a single partitioned topic. If the consuming one is down, the other one in queue can start consuming the messages. Group ID decides the group of consumer

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_single\_part --group 1*

* If Consumers are under multiple groups , then two consumers can consume the messages from a single partitioned topic

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_single\_part --group 2*

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_single\_part --group 2*

#### Multiple Partition Topic – Single Consumer

* By default producer will write into all partitions in round robin fashion. But Producer API provide us capability to write into specific partition or based on custom map logic between data and partition. We will explore those when we build Kafka applications using supported programming languages such as Scala, Python etc.
* kafka-console-producer.sh can write to topic only in round robin fashion.

1. Creating a topic with multiple partition

*kafka-topics.sh --zookeeper m01.itversity.com:2181,m02.itversity.com:2181,w01.itversity.com:2181 --create --topic test\_multi\_part --partitions 3 --replication-factor 2*

1. Publishing messages to above topic from gen\_logs

*tail\_logs.sh | kafka-console-producer.sh --broker-list w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_multi\_part –group 1*

1. Consuming messages using kafka-console-consumer.sh:

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic test\_multi\_part group 1*

#### Multiple Partition Topic – Multiple Consumers

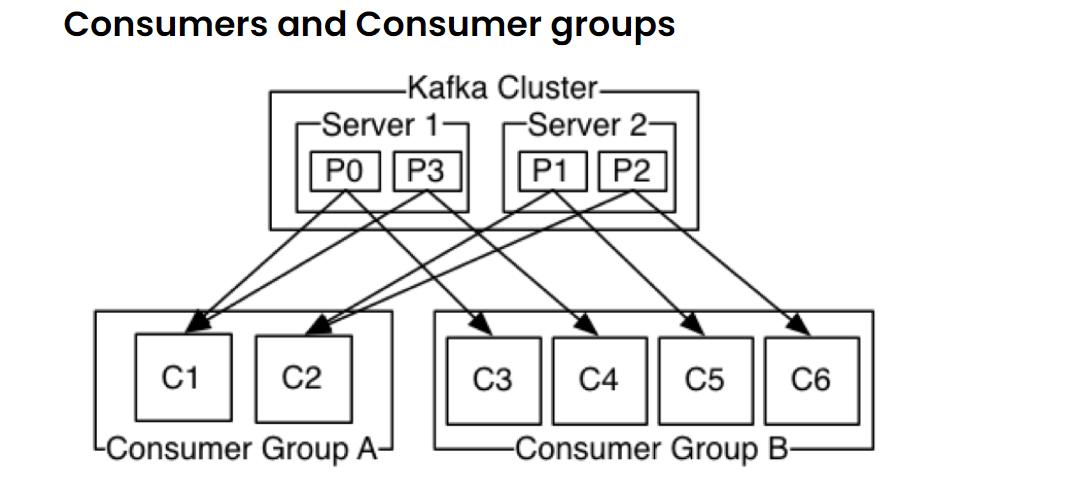
* Here at a time, still only one consumer from a consumer group can read a partition at given point of time. But since multiple partitions are there, mutually exclusive reads from multiple consumers is possible under one consumer group
* One producer – Topic with multiple partitions – Multiple consumers in one group
  + All consumers in the group will read data from partitions in tandem and in round robin fashion.
  + Each consumer will write the offset to a centralized location.
* One producer – Topic with multiple partitions – One consumer per partition in one group

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic retail\_multi\_part --partition 0*

*kafka-console-consumer.sh --bootstrap-server w01.itversity.com:9092,w02.itversity.com:9092,w03.itversity.com:9092 --topic retail\_multi\_part --partition 1*

* One producer – Topic with multiple partitions – One consumer for p0 and p3, one consumer for p1 and p2. This combination is not possible with kafka-console-consumer but we can take care of it using consumer APIs as part of our applications to consume the data.

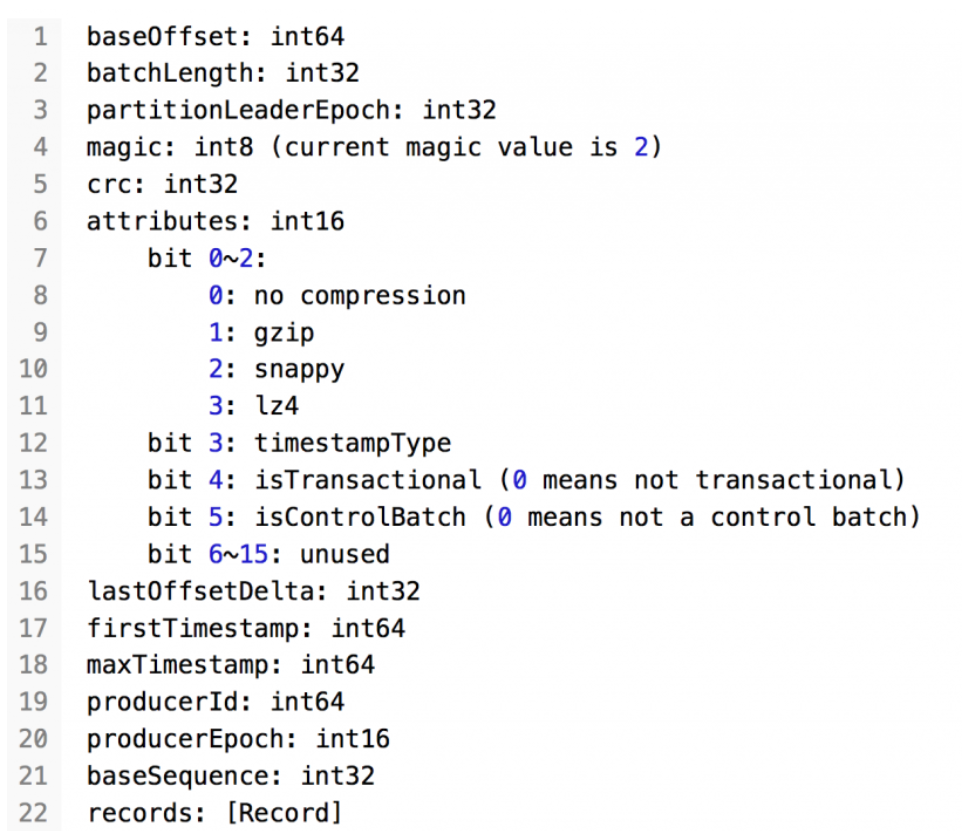
#### Consumers and Consumer groups



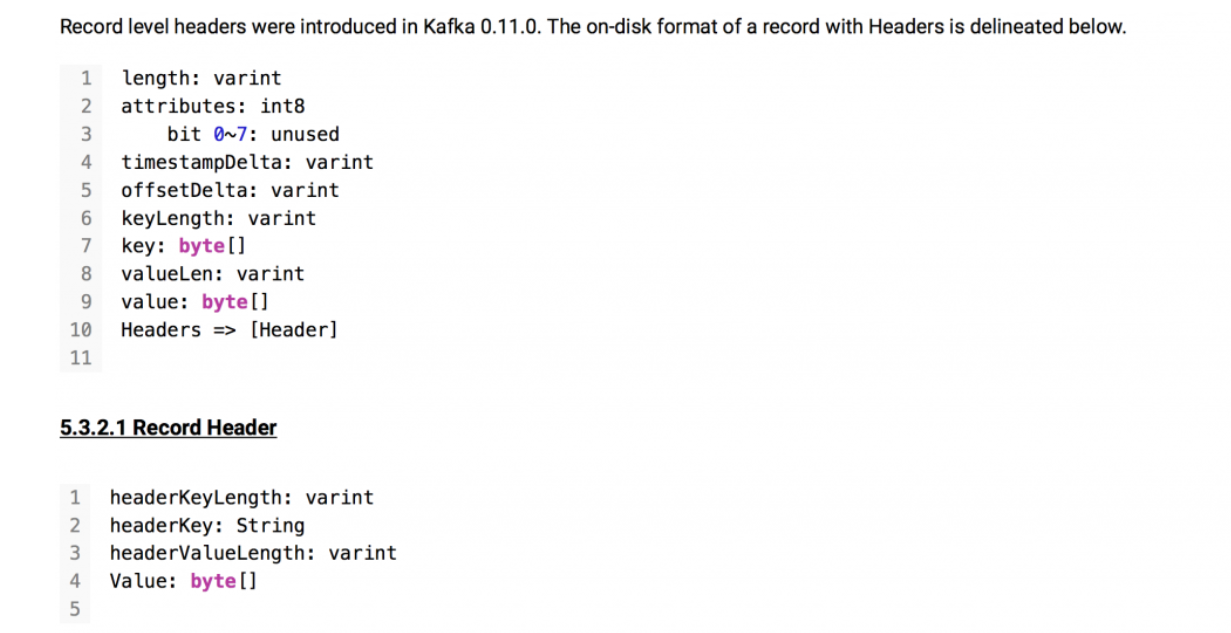
* Consumers label themselves with a consumer group name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group.
* This process of maintaining membership in the group is handled by Kafka dynamically. If new instances join the group they will take over some partitions from other members of the group; if an instance dies, its partitions will be distributed to the remaining instances.
* To consume data in parallel from topic which is partitioned, we need to have separate consumer for each partition. At any given point in time more than one consumer cannot consume data from same partition.

## Messages and Message Format

* Producer produces messages to Kafka topic in batches for performance reasons.
* All the messages are appended to the log files associated with partitions.
* We can have key for each message (using APIs). Keys are used in partitioning the data as well as log compaction if the logs have to be replayed.
* Partitions are allocated to the message either in round robin fashion (when there is no key) or based on hash mod value on key.
* Consumer periodically poll into Kafka topic partition using offset and read messages continuously from the topic.
* As messages are produced in batches, there will be some information for each batch as well as some information for each message with in batch written to log file (along with actual messages)
* Advantages of batching
  + As part of producing messages, we can send larger TCP packets leveraging network bandwidth. It is always better in performance compared to sending individual messages and waiting for acknowledgements.
  + Compression rates will be much higher with larger batch of messages than individual messages.
* When batching is used producer will take care of grouping messages for each partition and append to the log file associated with the partition by talking to the leader of respective partitions.
* Even while consuming data it is not advisable to consume individual messages and commit offset for each message. Performance will be poor.
* As we typically consume multiple messages in each iteration, our consumers should have necessary logic to deal with errors in such a way that same data is not duplicated in the target system.
* Record/Message Batch Format



* Record/Message Format



# Ingesting Web Server logs to Kafka

Kafka topic is intermediate data structure. We can get data into Kafka topic using different approaches.

* Custom applications using Producer API
* Kafka Connect
* Logstash
* Flume
* and more

Producer APIs can be used as part of the web applications to push log messages to Kafka topic directly. But in some cases it might not be possible to use Producer APIs (e.g.: mission critical legacy web applications) – we need to use tools like Logstash, Flume etc.

Kafka topic is intermediate data structure. We can get data into Kafka topic using different approaches.

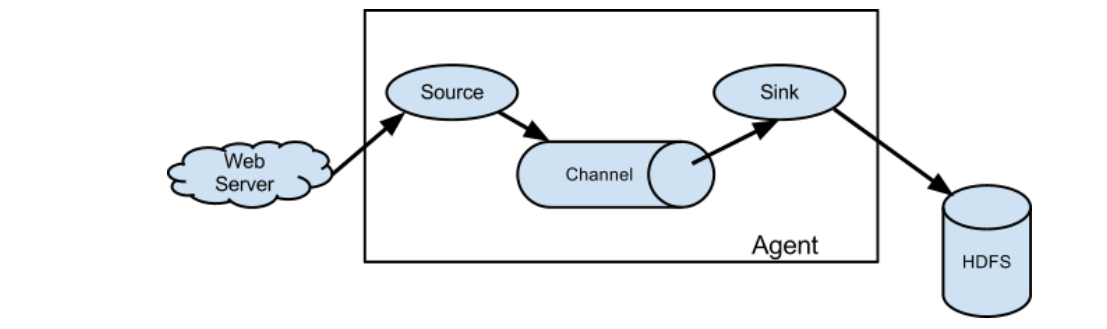
* Custom applications using Producer API
* Kafka Connect
* Logstash
* Flume
* and more

Producer APIs can be used as part of the web applications to push log messages to Kafka topic directly. But in some cases, it might not be possible to use Producer APIs (e.g.: mission critical legacy web applications) – we need to use tools like Logstash, Flume etc.

## Apache Flume

**user guide**:

https://flume.apache.org/FlumeUserGuide.html

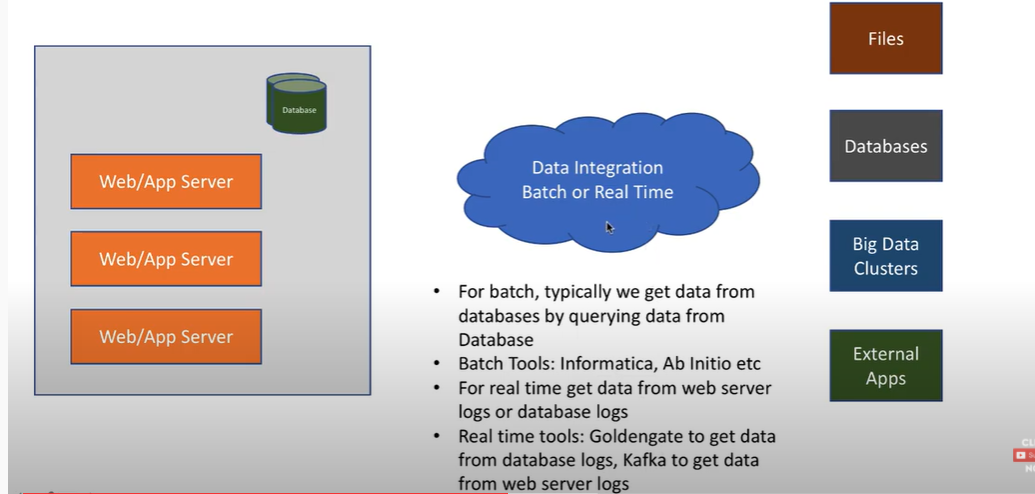


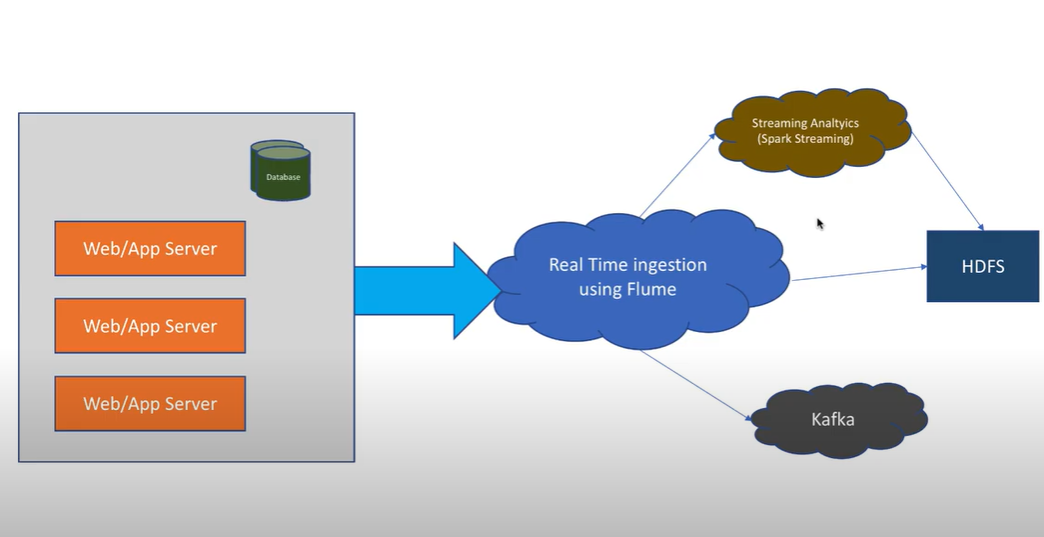
Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.

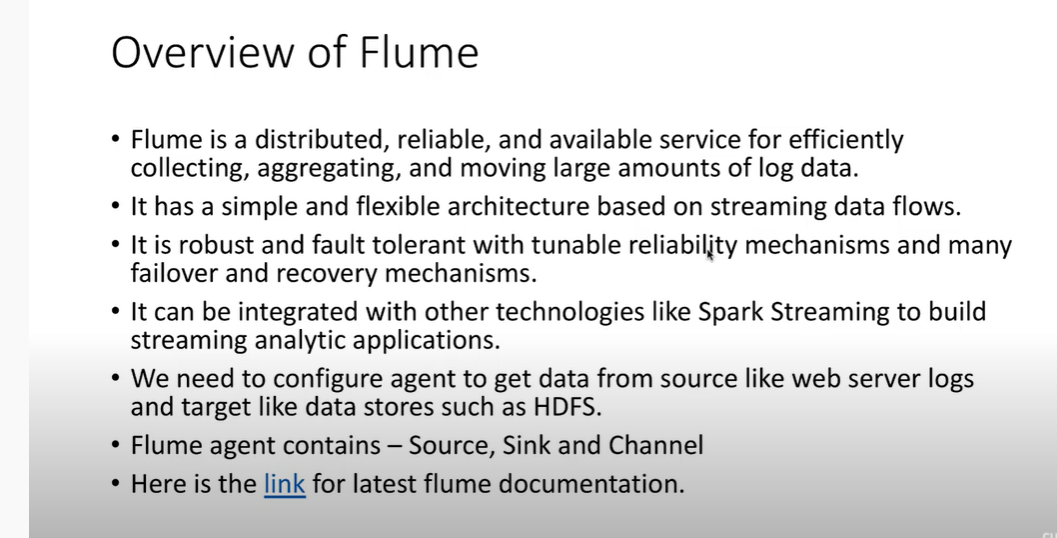
**Why Flume:**

* collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.





**Overview of Flume:**



A flume data flow three components for a single agent. And there can be multiple agents

**source:**

A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to

Flume in a format that is recognized by the target Flume source. For example, an Avro Flume source can be used to

receive Avro events from Avro clients or other Flume agents in the flow that send events from an Avro sink

* + 1. **netcat**: This is only used for exploratory purposes, not in real time. It is known as the TCP/IP Swiss army knife. From the netcat man page: netcat is a simple unix utility which reads and writes data across network connections,

using TCP or UDP protocol.

It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts

* + 1. **syslog** : This is used when an application is running in a container and logs need to be flushed out of container for getting consumed by flume.
    2. **Exec** : Web app application running on a server or a virtual machine and you can deploy flume agent on top of it. Since Your app and flume agent are on same sever, you can fire commands like ‘tail -f log\_file’ through conf file. Such scenarios, we need to define ‘exec’ as source type
    3. **Avro** source is mainly used in multi agent scenarios where one agent’s source is another agent’s sink. This deserializer is able to read an Avro container file, and it generates one event per Avro record in the file. Each event is annotated with a header that indicates the schema used. The body of the event is the binary Avro record data, not including the schema or the rest of the container file elements.
    4. **Thrift** source is mainly used in multi agent scenarios where one agent’s source is another agent’s sink. It is an interface definition language and binary communication protocol used for defining and creating services for numerous programming languages. It forms a remote procedure call (RPC) framework and was developed at Facebook for "scalable cross-language services development.

**channel:**

When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store

that keeps the event until it’s consumed by a Flume sink.

The file channel is one example – it is backed by the local filesystem

* Memory
* File
* Kafka

**sink:**

The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink)

or forwards it to the Flume source of the next Flume agent (next hop) in the flow.

* Logger
* HDFS
* Avro

The source and sink within the given agent run asynchronously with the events staged in the channel.

**Sample config file for an agent (agent.conf) :**

# Name the components on this agent

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Describe/configure the source

a1.sources.r1.type = netcat

a1.sources.r1.bind = 0.0.0.0

a1.sources.r1.port = 45455

# Describe the sink

a1.sinks.k1.type = logger

# Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

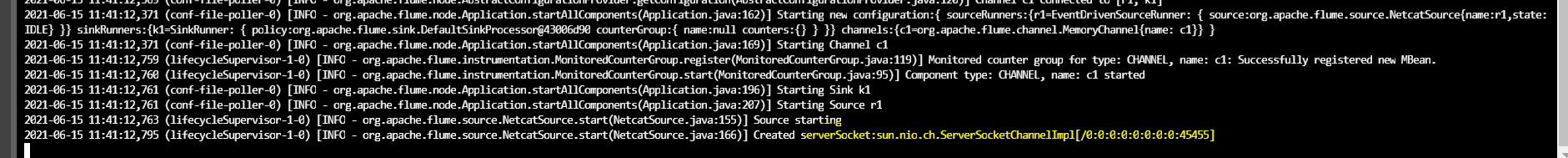
# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

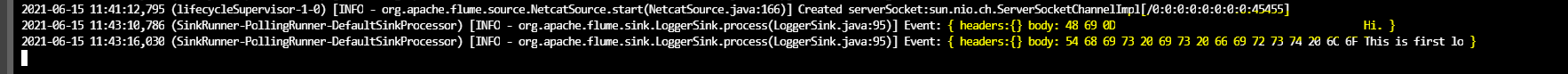
**Command to start a flume agent with above conf:**

*flume-ng agent --conf /opt/flume/conf --name ag1 --conf-file agent.conf*



**Listening to above port via telnet and logging message:**





**Simple Multi-Agent flow:**

