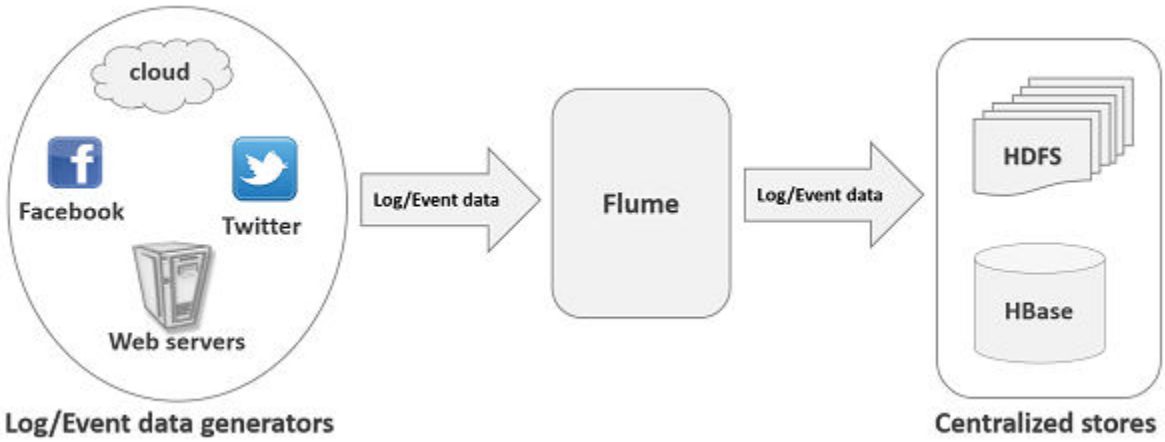
FLUME:

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.

Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



Advantages of Flume:

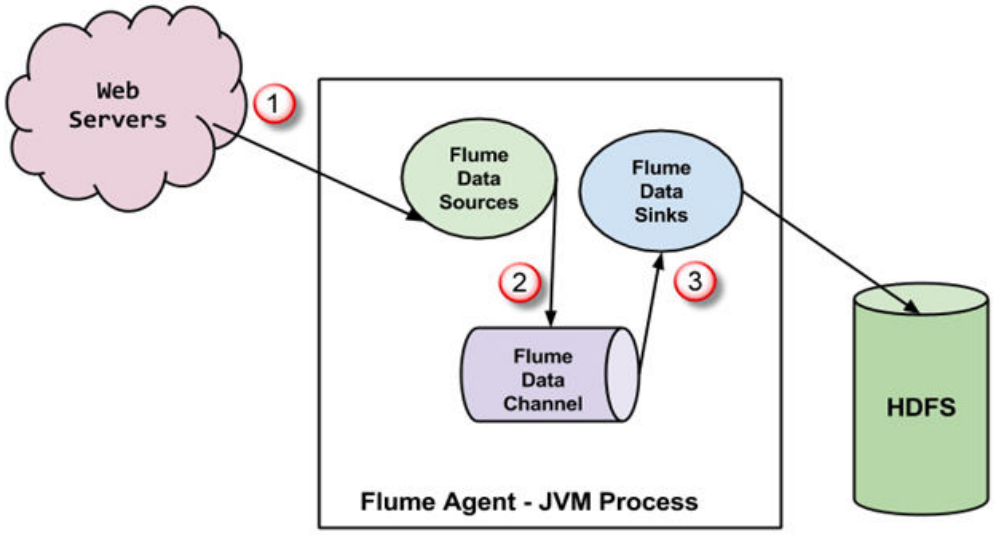
- Using Apache Flume we can store the data in to any of the centralized stores (HBase, HDFS).
- When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
- The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
- Flume is reliable, fault tolerant, scalable, manageable, and customizable.

Features of Flume:

- Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.
- Using Flume, we can get the data from multiple servers immediately into Hadoop.
- Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.
- Flume supports a large set of sources and destinations types.
- Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Flume can be scaled horizontally.

FLUME Architecture:

A **Flume agent** is a **JVM** process which has 3 components –**Flume Source, Flume Channel** and **Flume Sink**– through which events propagate after initiated at an external source.



Flume Architecture

- In the above diagram, the events generated by external source (WebServer) are consumed by Flume Data Source. The external source sends events to Flume source in a format that is recognized by the target source.
- Flume Source receives an event and stores it into one or more channels. The channel acts as a store which keeps the event until it is consumed by the flume sink. This channel may use a local file system in order to store these events.
- Flume sink removes the event from a channel and stores it into an external repository like e.g., HDFS. There could be multiple flume agents, in which case flume sink forwards the event to the flume source of next flume agent in the flow.

Flume Event:

- An event is the basic unit of the data transported inside Flume.
- It contains a payload of byte array that is to be transported from the source to the destination accompanied by optional headers.
- A typical Flume event would have the following structure

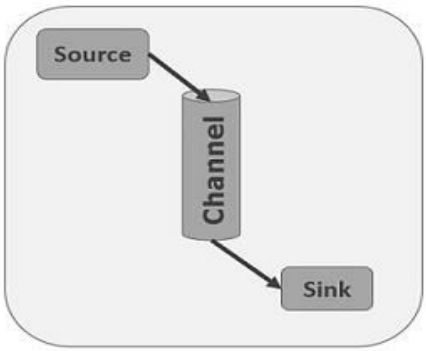
Flume event

Flume Agent:

- An agent is an independent daemon process (JVM) in Flume.
- It receives the data (events) from clients or other agents and forwards it to its next destination (sink or agent). Flume may have more than one agent.

Following diagram represents a Flume Agent:

As shown in the diagram a Flume Agent contains three main components namely, source, channel, and sink.



Flume Agent

Source:

- A source is the component of an Agent which receives data from the data generators and transfers it to one or more channels in the form of Flume events.
- Apache Flume supports several types of sources and each source receives events from a specified data generator.

Example: Avro source, Thrift source, twitter 1% source etc.

Channel:

- A channel is a transient store which receives the events from the source and buffers them till they are consumed by sinks. It acts as a bridge between the sources and the sinks.
- These channels are fully transactional and they can work with any number of sources and sinks.

Example: JDBC channel, File system channel, Memory channel, etc.

Sink:

- A sink stores the data into centralized stores like HBase and HDFS.
- It consumes the data (events) from the channels and delivers it to the destination.
- The destination of the sink might be another agent or the central stores.

Example: HDFS sink

Note: A flume agent can have multiple sources, sinks and channels. We have listed all the supported sources, sinks, channels in the Flume configuration chapter of this tutorial.

Multi-hop Flow:
Within Flume, there can be multiple agents and before reaching the final destination, an event may travel through more than one agent. This is known as multi-hop flow.

Fan-out Flow:
The dataflow from one source to multiple channels is known as fan-out flow. It is of two types:

- **Replicating:** The data flow where the data will be replicated in all the configured channels.
- **Multiplexing:** The data flow where the data will be sent to a selected channel which is mentioned in the header of the event.

Fan-in Flow:
The data flow in which the data will be transferred from many sources to one channel is known as fan-in flow.

Failure Handling:

- In Flume, for each event, two transactions take place: one at the sender and one at the receiver.
- The sender sends events to the receiver.
- Soon after receiving the data, the receiver commits its own transaction and sends a "received" signal to the sender.
- After receiving the signal, the sender commits its transaction. (Sender will not commit its transaction till it receives a signal from the receiver.)

In the Flume configuration file, we need to:

- Name the components of the current agent.
- Describe/Configure the source.
- Describe/Configure the sink.
- Describe/Configure the channel.
- Bind the source and the sink to the channel.

Naming the Components:
First of all, you need to name/list the components such as sources, sinks, and the channels of the agent, as shown below.

agent_name.sources = source_name
agent_name.sinks = sink_name
agent_name.channels = channel_name
Flume supports various sources, sinks, and channels. They are listed in the table given below.

Sources	Channels	Sinks
Avro Source	Memory Channel	HDFS Sink
Thrift Source	JDBC Channel	Hive Sink
Exec Source	Kafka Channel	Logger Sink
JMS Source	File Channel	Avro Sink
Spooling Directory Source	Spillable Memory Channel	Thrift Sink
Twitter 1% firehose Source	Pseudo Transaction Channel	IRC Sink
Kafka Source		File Roll Sink
NetCat Source		Null Sink
Sequence Generator Source		HBaseSink
Syslog Sources		AsyncHBaseSink
Syslog TCP Source		MorphlineSolrSink
Multiport Syslog TCP Source		ElasticSearchSink
Syslog UDP Source		Kite Dataset Sink
HTTP Source		Kafka Sink
Stress Source		
Legacy Sources		
Thrift Legacy Source		
Custom Source		
Scribe Source		

For example, if you are transferring Twitter data using twitter source through a memory channel to an HDFS sink, and the agent name id TwitterAgent, then

TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

After listing the components of the agent, you have to describe the source(s), sink(s), and channel(s) by providing values to their properties.

Describing the Source:
Each source will have a separate list of properties. The property named “type” is common to every source, and it is used to specify the type of the source we are using.

Along with the property “type”, it is needed to provide the values of all the required properties of a particular source to configure it, as shown below.

agent_name.sources. source_name.type = value
agent_name.sources. source_name.property2 = value
agent_name.sources. source_name.property3 = value

For example, if we consider the twitter source, following are the properties to which we must provide values to configure it.

TwitterAgent.sources.Twitter.type = Twitter (type name)
TwitterAgent.sources.Twitter.consumerKey =
TwitterAgent.sources.Twitter.consumerSecret =
TwitterAgent.sources.Twitter.accessToken =
TwitterAgent.sources.Twitter.accessTokenSecret =

Describing the Sink:
Just like the source, each sink will have a separate list of properties. The property named "type" is common to every sink, and it is used to specify the type of the sink we are using. Along with the property "type", it is needed to provide values to all the required properties of a particular sink to configure it, as shown below.

agent_name.sinks. sink_name.type = value
agent_name.sinks. sink_name.property2 = value
agent_name.sinks. sink_name.property3 = value

For example, if we consider HDFS sink, following are the properties to which we must provide values to configure it.

TwitterAgent.sinks.HDFS.type = hdfs (type name)
TwitterAgent.sinks.HDFS.hdfs.path = HDFS directory's Path to store the data

Describing the Channel:
Flume provides various channels to transfer data between sources and sinks. Therefore, along with the sources and the channels, it is needed to describe the channel used in the agent.
To describe each channel, you need to set the required properties, as shown below

agent_name.channels.channel_name.type = value
agent_name.channels.channel_name. property2 = value
agent_name.channels.channel_name. property3 = value

For example, if we consider memory channel, following are the properties to which we must provide values to configure it.

TwitterAgent.channels.MemChannel.type = memory (type name)

Binding the Source and the Sink to the Channel:
Since the channels connect the sources and sinks, it is required to bind both of them to the channel, as shown below.

agent_name.sources.source_name.channels = channel_name
agent_name.sinks.sink_name.channels = channel_name

The following example shows how to bind the sources and the sinks to a channel. Here, we consider twitter source, memory channel, and HDFS sink.

TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sinks.HDFS.channels = MemChannel

Starting a Flume Agent:
After configuration, we have to start the Flume agent. It is done as follows:
$ bin/flume-ng agent --conf ./conf/ -f conf/twitter.conf
Dflume.root.logger=DEBUG, console -n TwitterAgent

Where:

- **agent:** Command to start the Flume agent
- **--conf ,-c<conf>:** Use configuration file in the conf directory
- **-f<file>:** Specifies a config file path, if missing
- **--name, -n <name>:** Name of the twitter agent
- **-D property = value** Sets a Java system property value.