tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

## About the Tutorial

Flume is a standard, simple, robust, flexible, and extensible tool for data ingestion from various data producers (webservers) into Hadoop. In this tutorial, we will be using simple and illustrative example to explain the basics of Apache Flume and how to use it in practice.

## Audience

This tutorial is meant for all those professionals who would like to learn the process of transferring log and streaming data from various webservers to HDFS or HBase using Apache Flume.

## Prerequisites

To make the most of this tutorial, you should have a good understanding of the basics of Hadoop and HDFS commands.

## Copyright & Disclaimer

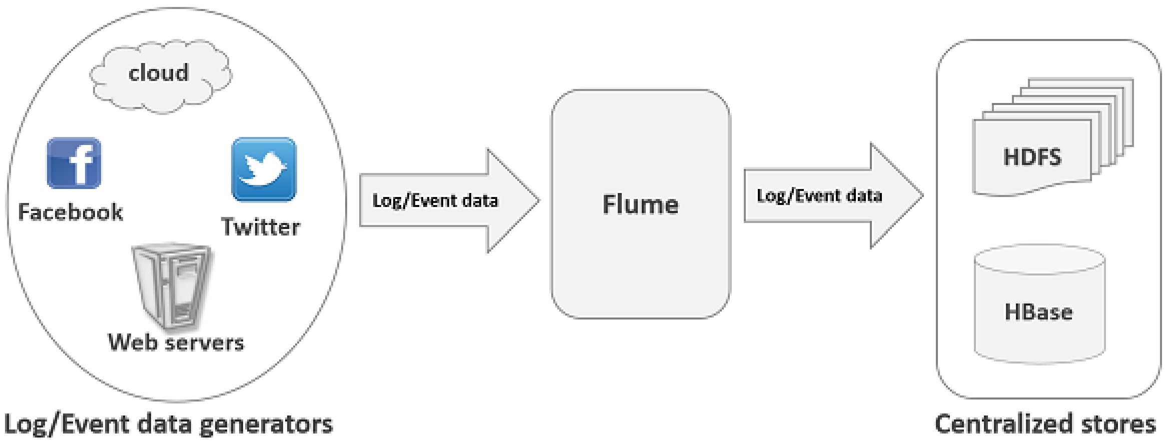
# Table of Contents

# 1. FLUME – INTRODUCTION

## What is 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.



## Applications of Flume

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

## Advantages of Flume

Here are the advantages of using 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.

- Flume provides the feature of **contextual routing**.

- 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

Some of the notable features of Flume are as follows:

- 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.

# 2. DATA TRANSFER IN HADOOP

**Big Data**, as we know, is a collection of large datasets that cannot be processed using traditional computing techniques. Big Data, when analyzed, gives valuable results. **Hadoop** is an open-source framework that allows to store and process Big Data in a distributed environment across clusters of computers using simple programming models.

## Streaming / Log Data

Generally, most of the data that is to be analyzed will be produced by various data sources like applications servers, social networking sites, cloud servers, and enterprise servers. This data will be in the form of **log files** and **events**.

**Log file:** In general, a log file is a **file** that lists events/actions that occur in an operating system. For example, web servers list every request made to the server in the log files.

On harvesting such log data, we can get information about:

- the application performance and locate various software and hardware failures.
- the user behavior and derive better business insights.

The traditional method of transferring data into the HDFS system is to use the **put** command. Let us see how to use the **put** command.

## HDFS put Command

The main challenge in handling the log data is in moving these logs produced by multiple servers to the Hadoop environment.

Hadoop **File System Shell** provides commands to insert data into Hadoop and read from it. You can insert data into Hadoop using the **put** command as shown below.

```
$ Hadoop fs –put /path of the required file  /path in HDFS where to save the
file
```

### Problem with put Command

We can use the **put** command of Hadoop to transfer data from these sources to HDFS. But, it suffers from the following drawbacks:

- Using **put** command, we can transfer **only one file at a time** while the data generators generate data at a much higher rate. Since the analysis made on older data is less accurate, we need to have a solution to transfer data in real time.
- If we use **put** command, the data is needed to be packaged and should be ready for the upload. Since the webservers generate data continuously, it is a very difficult task.

What we need here is a solutions that can overcome the drawbacks of **put** command and transfer the "streaming data" from data generators to centralized stores (especially HDFS) with less delay.

### Problem with HDFS

In HDFS, the file exists as a directory entry and the length of the file will be considered as zero till it is closed. For example, if a source is writing data into HDFS and the network was interrupted in the middle of the operation (without closing the file), then the data written in the file will be lost.

Therefore we need a reliable, configurable, and maintainable system to transfer the log data into HDFS.

**Note**: In POSIX file system, whenever we are accessing a file (say performing write operation), other programs can still read this file (at least the saved portion of the file). This is because the file exists on the disc before it is closed.

## Available Solutions

To send streaming data (log files, events etc..,) from various sources to HDFS, we have the following tools available at our disposal:

### Facebook's Scribe

Scribe is an immensely popular tool that is used to aggregate and stream log data. It is designed to scale to a very large number of nodes and be robust to network and node failures.

### Apache Kafka

Kafka has been developed by Apache Software Foundation. It is an open-source message broker. Using Kafka, we can handle feeds with high-throughput and low-latency.

### Apache Flume

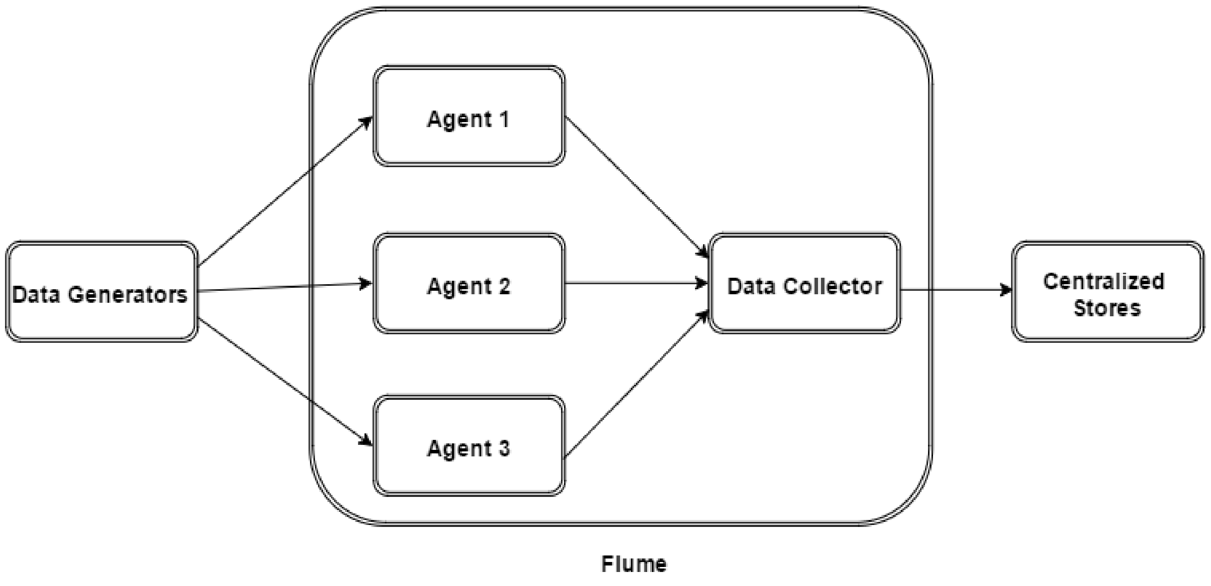
Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log data, events (etc...) from various webserves to a centralized data store.

It is a highly reliable, distributed, and configurable tool that is principally designed to transfer streaming data from various sources to HDFS.

In this tutorial, we will discuss in detail how to use Flume with some examples.

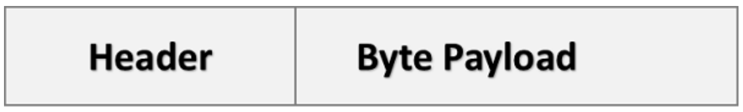
# 3. FLUME – ARCHITECTURE

The following illustration depicts the basic architecture of Flume. As shown in the illustration, **data generators** (such as Facebook, Twitter) generate data which gets collected by individual Flume **agents** running on them. Thereafter, a **data collector** (which is also an agent) collects the data from the agents which is aggregated and pushed into a centralized store such as HDFS or HBase.



## Flume Event

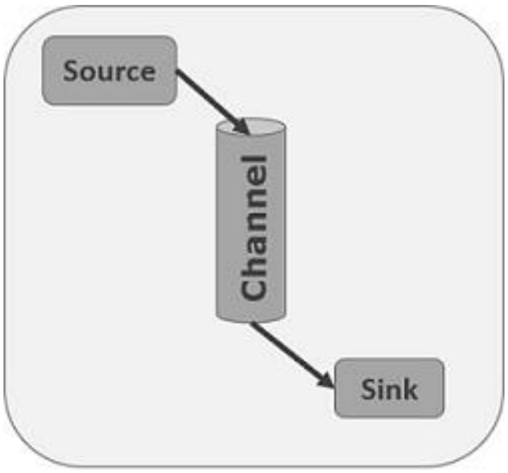
An **event** is the basic unit of the data transported inside **Flume**. It contains a payload of byte array that is to be transported from the source to the destination accompanied by optional headers. A typical Flume event would have the following structure:



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**



Flume Agent

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

## 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, HBase sink, Avro sink, Kafka sink, etc.

**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.

## Additional Components of Flume Agent

What we have discussed above are the primitive components of the agent. In addition to this, we have a few more components that play a vital role in transferring the events from the data generator to the centralized stores.

### Interceptors

Interceptors are used to alter/inspect flume events which are transferred between source and channel.

### Channel Selectors

These are used to determine which channel is to be opted to transfer the data in case of multiple channels. There are two types of channel selectors:

- **Default channel selectors**: These are also known as replicating channel selectors they replicates all the events in each channel.

- **Multiplexing channel selectors**: These decides the channel to send an event based on the address in the header of that event.

### Sink Processors

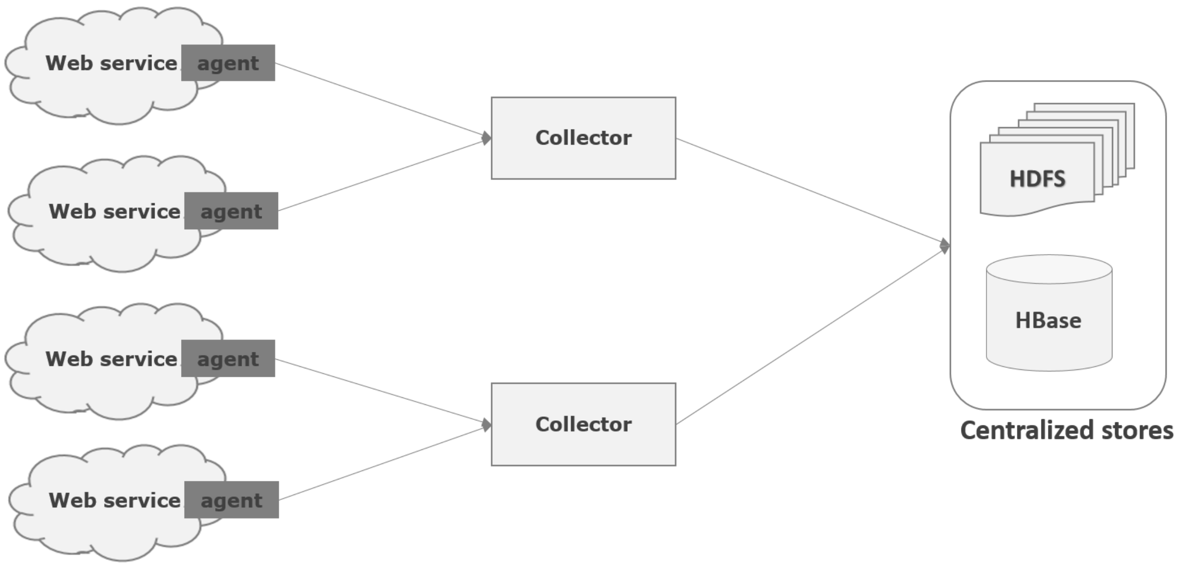
These are used to invoke a particular sink from the selected group of sinks. These are used to create failover paths for your sinks or load balance events across multiple sinks from a channel.

# 4. FLUME – DATA FLOW

Flume is a framework which is used to move log data into HDFS. Generally events and log data are generated by the log servers and these servers have Flume agents running on them. These agents receive the data from the data generators.

The data in these agents will be collected by an intermediate node known as **Collector**. Just like agents, there can be multiple collectors in Flume.

Finally, the data from all these collectors will be aggregated and pushed to a centralized store such as HBase or HDFS. The following diagram explains the data flow in Flume.



## 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.)

# 5. ENVIRONMENT

We already discussed the architecture of Flume in the previous chapter. In this chapter, let us see how to download and setup Apache Flume.

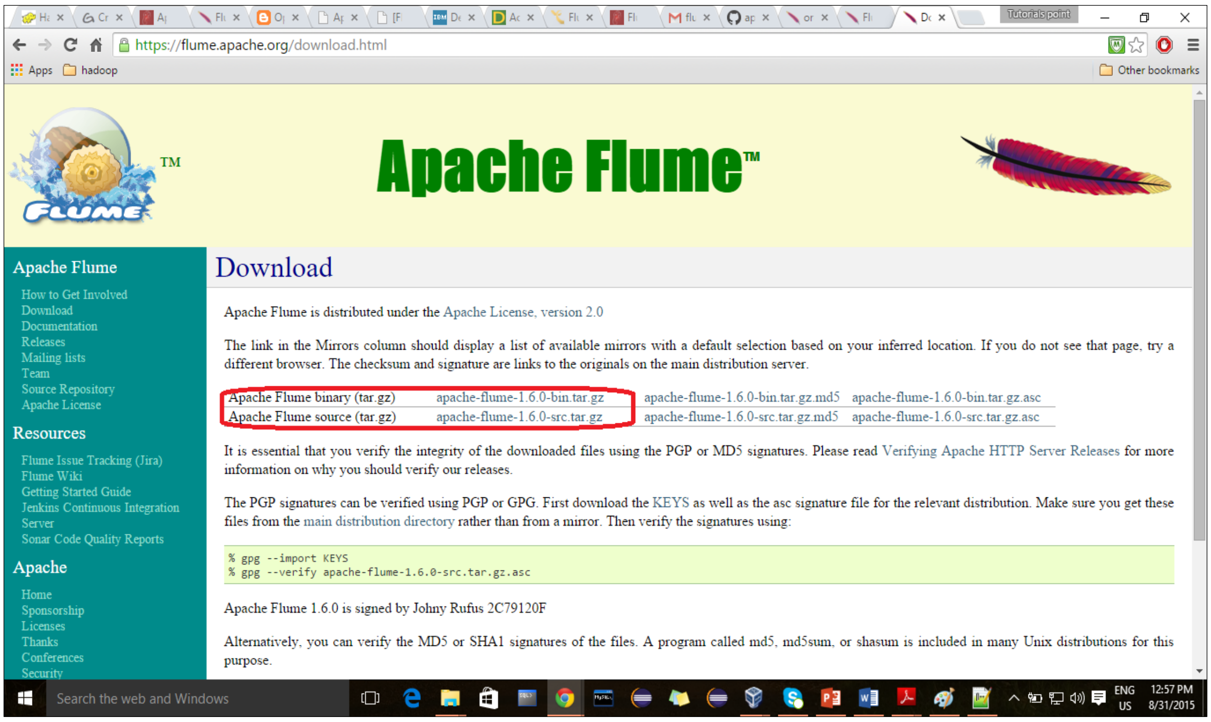
Before proceeding further, you need to have a Java environment in your system. So first of all, make sure you have Java installed in your system. For some examples in this tutorial, we have used Hadoop HDFS (as sink). Therefore, we would recommend that you go install Hadoop along with Java. To collect more information, follow the link: http://www.tutorialspoint.com/hadoop/hadoop_enviornment_setup.htm

## Installing Flume

First of all, download the latest version of Apache Flume software from the website https://flume.apache.org/.

### Step 1

Open the website. Click on the **download** link on the left-hand side of the home page. It will take you to the download page of Apache Flume.



Download

Apache Flume is distributed under the Apache License, version 2.0

The link in the Mirrors column should display a list of available mirrors with a default selection based on your inferred location. If you do not see that page, try a different browser. The checksum and signature are links to the originals on the main distribution server.

| | | | |
|---|---|---|---|
| Apache Flume binary (tar.gz) | apache-flume-1.6.0-bin.tar.gz | apache-flume-1.6.0-bin.tar.gz.md5 | apache-flume-1.6.0-bin.tar.gz.asc |
| Apache Flume source (tar.gz) | apache-flume-1.6.0-src.tar.gz | apache-flume-1.6.0-src.tar.gz.md5 | apache-flume-1.6.0-src.tar.gz.asc |

It is essential that you verify the integrity of the downloaded files using the PGP or MD5 signatures. Please read Verifying Apache HTTP Server Releases for more information on why you should verify our releases.

The PGP signatures can be verified using PGP or GPG. First download the KEYS as well as the asc signature file for the relevant distribution. Make sure you get these files from the main distribution directory rather than from a mirror. Then verify the signatures using:

```
% gpg --import KEYS
% gpg --verify apache-flume-1.6.0-src.tar.gz.asc
```

Apache Flume 1.6.0 is signed by Johny Rufus 2C79120F

Alternatively, you can verify the MD5 or SHA1 signatures of the files. A program called md5, md5sum, or shasum is included in many Unix distributions for this purpose.

### Step 2

In the Download page, you can see the links for binary and source files of Apache Flume. Click on the link apache-flume-1.6.0-bin.tar.gz

You will be redirected to a list of mirrors where you can start your download by clicking any of these mirrors. In the same way, you can download the source code of Apache Flume by clicking on apache-flume-1.6.0-src.tar.gz.

### Step 3

Create a directory with the name Flume in the same directory where the installation directories of **Hadoop**, **HBase**, and other software were installed (if you have already installed any) as shown below.

```
$ mkdir Flume
```

### Step 4

Extract the downloaded tar files as shown below.

```
$ cd Downloads/
$ tar zxvf apache-flume-1.6.0-bin.tar.gz
$ tar zxvf apache-flume-1.6.0-src.tar.gz
```

### Step 5

Move the content of apache**-flume-1.6.0-bin.tar** file to the **Flume** directory created earlier as shown below. (Assume we have created the Flume directory in the local user named Hadoop.)

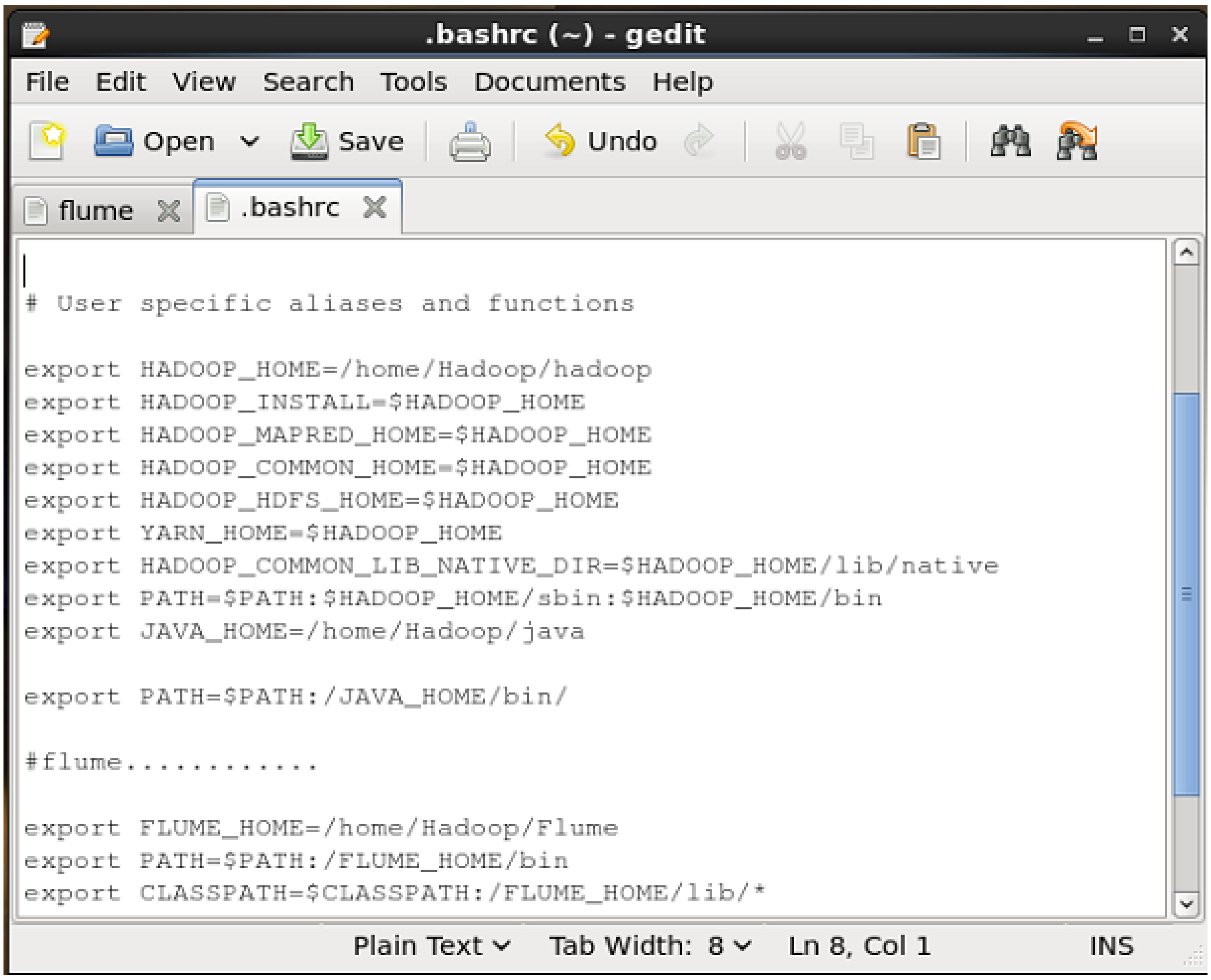
```
$ mv apache-flume-1.6.0-bin.tar/* /home/Hadoop/Flume/
```

## Configuring Flume

To configure Flume, we have to modify three files namely, **flume-env.sh**, **flume-conf.properties,** and **bash.rc**.

### Setting the Path / Classpath

In the **.bashrc** file, set the home folder, the path, and the classpath for Flume as shown below.

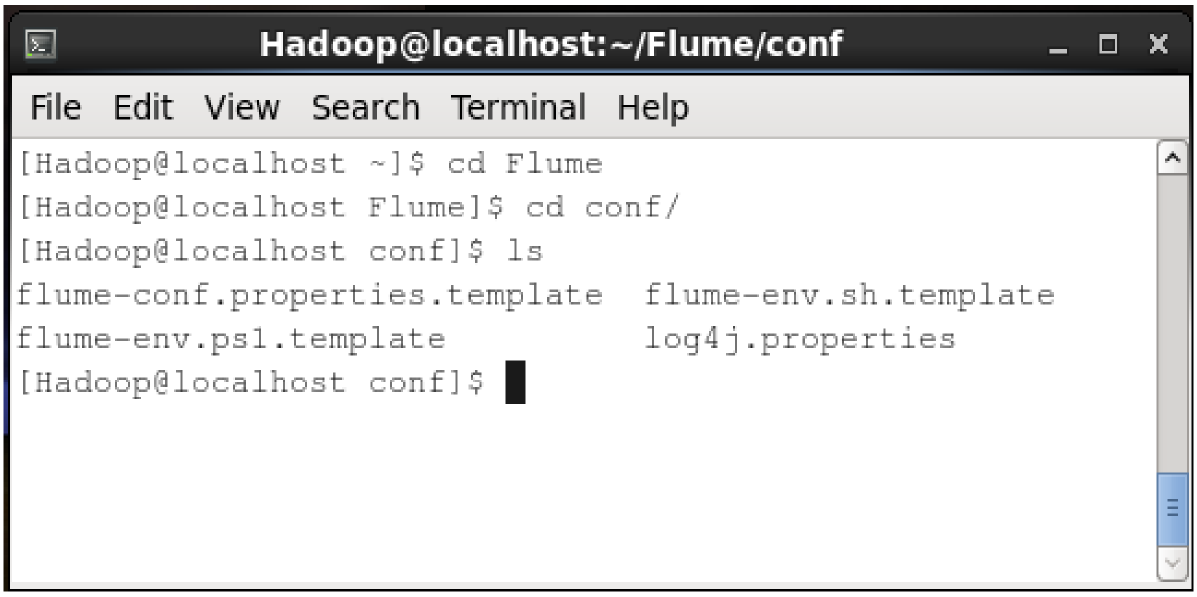


```
# User specific aliases and functions

export HADOOP_HOME=/home/Hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export JAVA_HOME=/home/Hadoop/java

export PATH=$PATH:/JAVA_HOME/bin/

#flume............

export FLUME_HOME=/home/Hadoop/Flume
export PATH=$PATH:/FLUME_HOME/bin
export CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*
```

## conf Folder

If you open the **conf** folder of Apache Flume, you will have the following four files:

- flume-conf.properties.template,
- flume-env.sh.template,
- flume-env.ps1.template, and
- log4j.properties.


Hadoop@localhost:~/Flume/conf
File Edit View Search Terminal Help

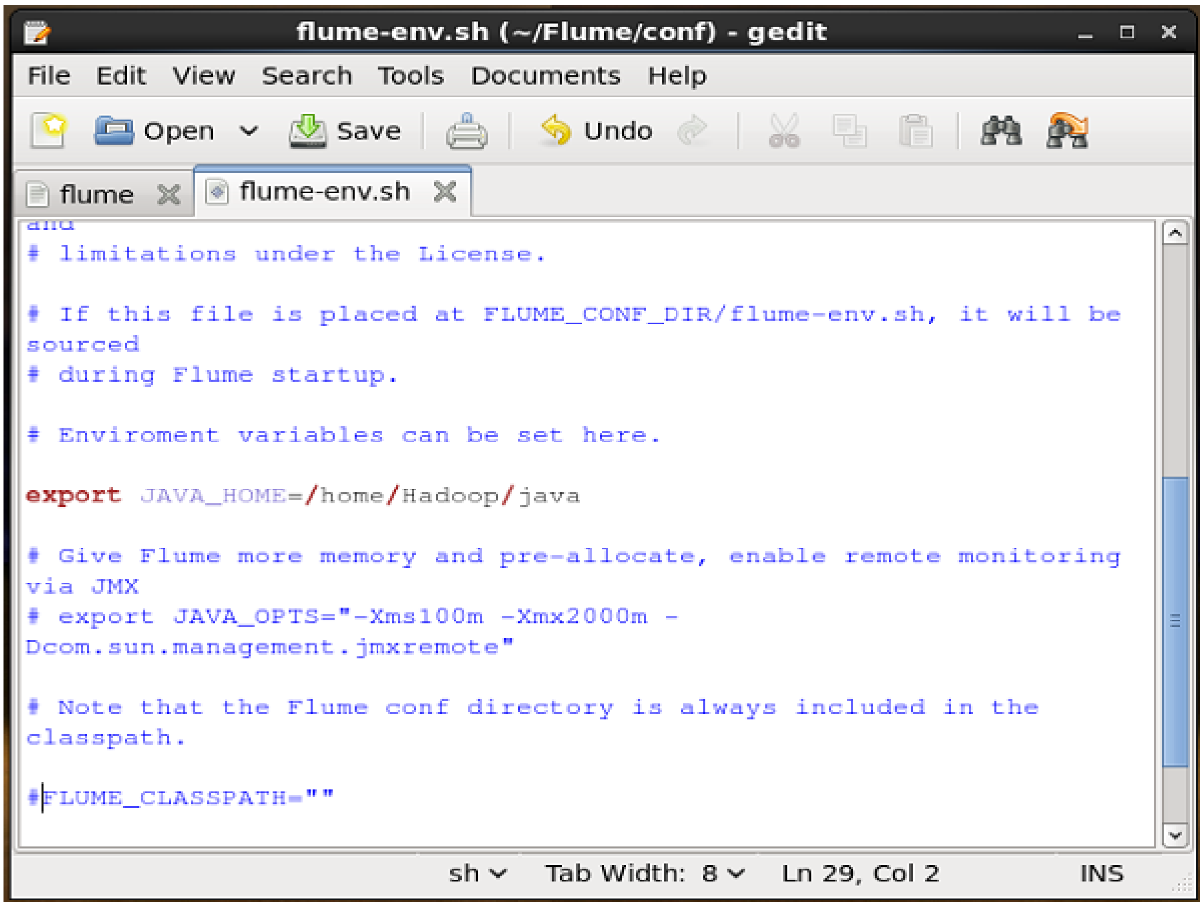
```
[Hadoop@localhost ~]$ cd Flume
[Hadoop@localhost Flume]$ cd conf/
[Hadoop@localhost conf]$ ls
flume-conf.properties.template  flume-env.sh.template
flume-env.ps1.template          log4j.properties
[Hadoop@localhost conf]$
```

Now rename

- **flume-conf.properties.template** file as **flume-conf.properties** and
- **flume-env.sh.template** as **flume-env.sh**

## flume-env.sh

Open **flume-env.sh** file and set the **JAVA_Home** to the folder where Java was installed in your system.


flume-env.sh (~/Flume/conf) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
flume
flume-env.sh
sh
Tab Width: 8
Ln 29, Col 2
INS

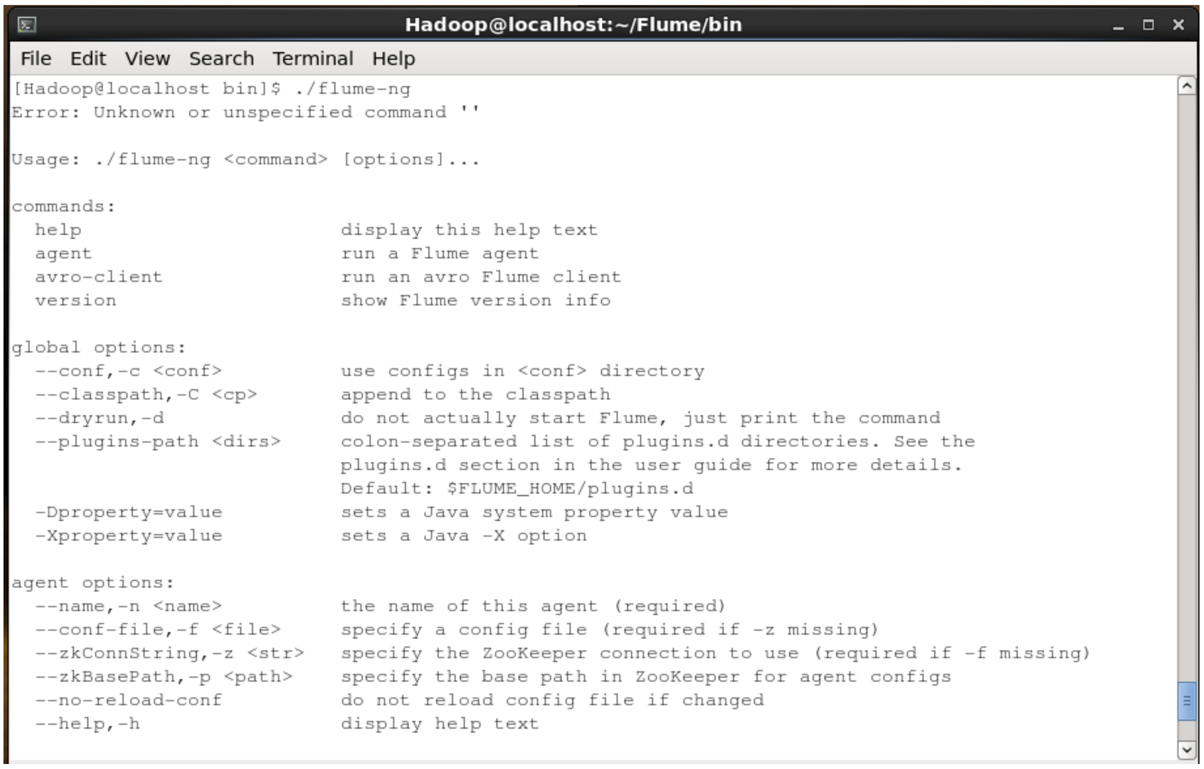
```
and
# limitations under the License.

# If this file is placed at FLUME_CONF_DIR/flume-env.sh, it will be
sourced
# during Flume startup.

# Enviroment variables can be set here.

export JAVA_HOME=/home/Hadoop/java

# Give Flume more memory and pre-allocate, enable remote monitoring
via JMX
# export JAVA_OPTS="-Xms100m -Xmx2000m -
Dcom.sun.management.jmxremote"

# Note that the Flume conf directory is always included in the
classpath.

#FLUME_CLASSPATH=""
```

## Verifying the Installation

Verify the installation of Apache Flume by browsing through the **bin** folder and typing the following command.

```
$ ./flume-ng
```

If you have successfully installed Flume, you will get a help prompt of Flume as shown below.


Hadoop@localhost:~/Flume/bin
File Edit View Search Terminal Help

```
[Hadoop@localhost bin]$ ./flume-ng
Error: Unknown or unspecified command ''

Usage: ./flume-ng <command> [options]...

commands:
  help                        display this help text
  agent                       run a Flume agent
  avro-client                 run an avro Flume client
  version                     show Flume version info

global options:
  --conf,-c <conf>            use configs in <conf> directory
  --classpath,-C <cp>         append to the classpath
  --dryrun,-d                 do not actually start Flume, just print the command
  --plugins-path <dirs>       colon-separated list of plugins.d directories. See the
                              plugins.d section in the user guide for more details.
                              Default: $FLUME_HOME/plugins.d
  -Dproperty=value            sets a Java system property value
  -Xproperty=value            sets a Java -X option

agent options:
  --name,-n <name>            the name of this agent (required)
  --conf-file,-f <file>       specify a config file (required if -z missing)
  --zkConnString,-z <str>     specify the ZooKeeper connection to use (required if -f missing)
  --zkBasePath,-p <path>      specify the base path in ZooKeeper for agent configs
  --no-reload-conf            do not reload config file if changed
  --help,-h                   display help text
```

# 6. FLUME – CONFIGURATION

After installing Flume, we need to configure it using the configuration file which is a Java property file having **key-value pairs**. We need to pass values to the keys in the file.

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.

Usually we can have multiple agents in Flume. We can differentiate each agent by using a unique name. And using this name, we have to configure each agent.

## 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<br>• Thrift Source<br>• Exec Source<br>• JMS Source<br>• Spooling Directory Source<br>• Twitter 1% firehose Source<br>• Kafka Source<br>• NetCat Source<br>• Sequence Generator Source<br>• Syslog Sources<br>• Syslog TCP Source | • Memory Channel<br>• JDBC Channel<br>• Kafka Channel<br>• File Channel<br>• Spillable Memory Channel<br>• Pseudo Transaction Channel | • HDFS Sink<br>• Hive Sink<br>• Logger Sink<br>• Avro Sink<br>• Thrift Sink<br>• IRC Sink<br>• File Roll Sink<br>• Null Sink<br>• HBaseSink<br>• AsyncHBaseSink<br>• MorphlineSolrSink<br>• ElasticSearchSink |

| | | |
|---|---|---|
| • Multiport Syslog TCP Source<br>• Syslog UDP Source<br>• HTTP Source<br>• Stress Source<br>• Legacy Sources<br>• Thrift Legacy Source<br>• Custom Source<br>• Scribe Source | | • Kite Dataset Sink<br>• Kafka Sink |

You can use any of them. 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:

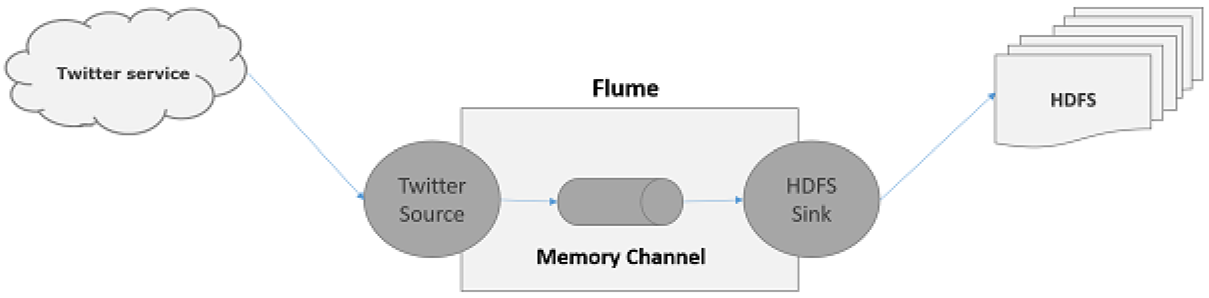
- **agent**: Command to start the Flume agent
- **--conf ,-c<conf>:** Use configuration file in the conf directory
- **-f<file>:** Specifies a config file path, if missing
- **--name, -n <name> :** Name of the twitter agent
- **-D property =value:** Sets a Java system property value.

# 7. FETCHING TWITTER DATA

Using Flume, we can fetch data from various services and transport it to centralized stores (HDFS and HBase). This chapter explains how to fetch data from Twitter service and store it in HDFS using Apache Flume.

As discussed in Flume Architecture, a webserver generates log data and this data is collected by an agent in Flume. The channel buffers this data to a sink, which finally pushes it to centralized stores.

In the example provided in this chapter, we will create an application and get the tweets from it using the experimental twitter source provided by Apache Flume. We will use the memory channel to buffer these tweets and HDFS sink to push these tweets into the HDFS.



To fetch Twitter data, we will have to follow the steps given below:

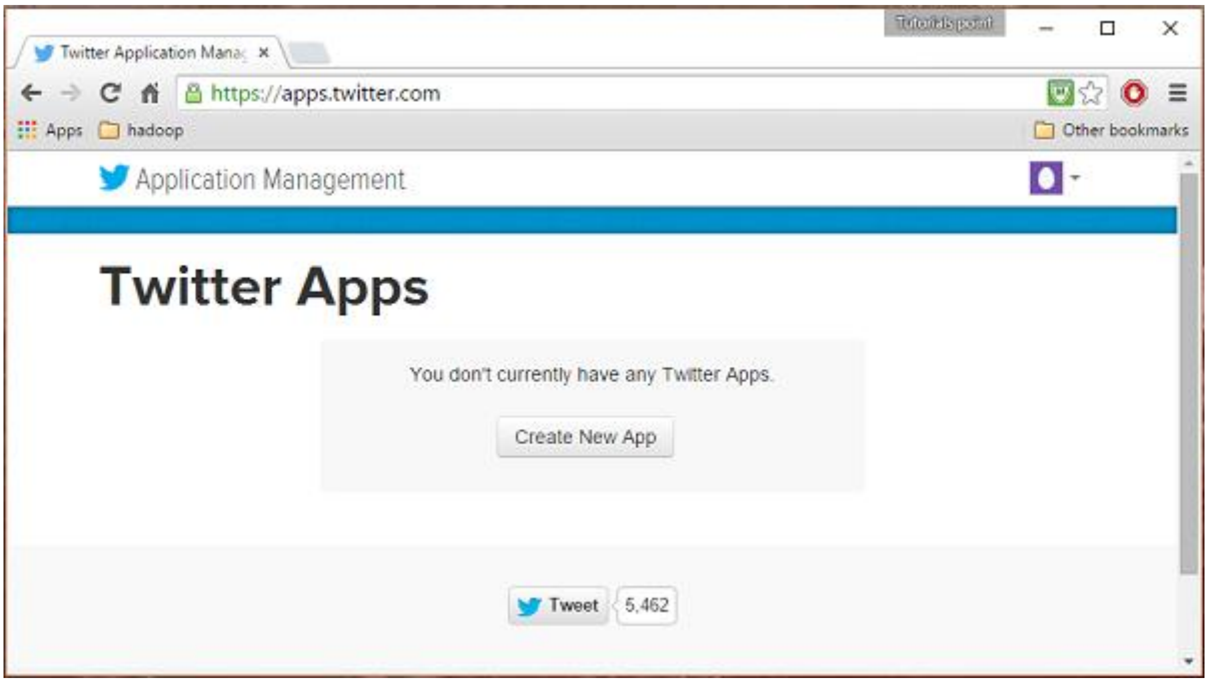
- Create a twitter Application
- Install / Start HDFS
- Configure Flume

## Creating a Twitter Application

In order to get the tweets from Twitter, it is needed to create a Twitter application. Follow the steps given below to create a Twitter application.

### Step 1

To create a Twitter application, click on the following link https://apps.twitter.com/. Sign in to your Twitter account. You will have a Twitter Application Management window where you can create, delete, and manage Twitter Apps.

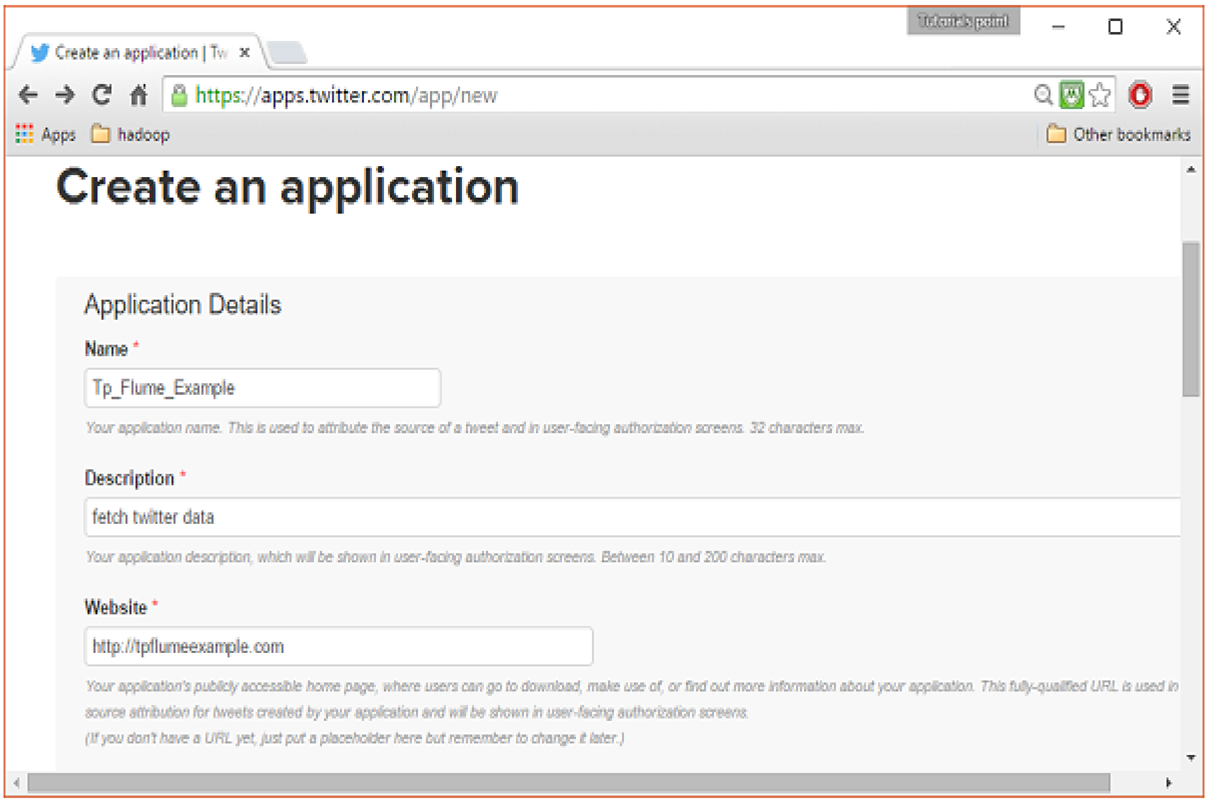


Twitter Apps

You don't currently have any Twitter Apps.

## Step 2

Click on the **Create New App** button. You will be redirected to a window where you will get an application form in which you have to fill in your details in order to create the App. While filling the website address, give the complete URL pattern, for example, **http://example.com**.



Create an application

Application Details

Name *

Tp_Flume_Example

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

fetch twitter data

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

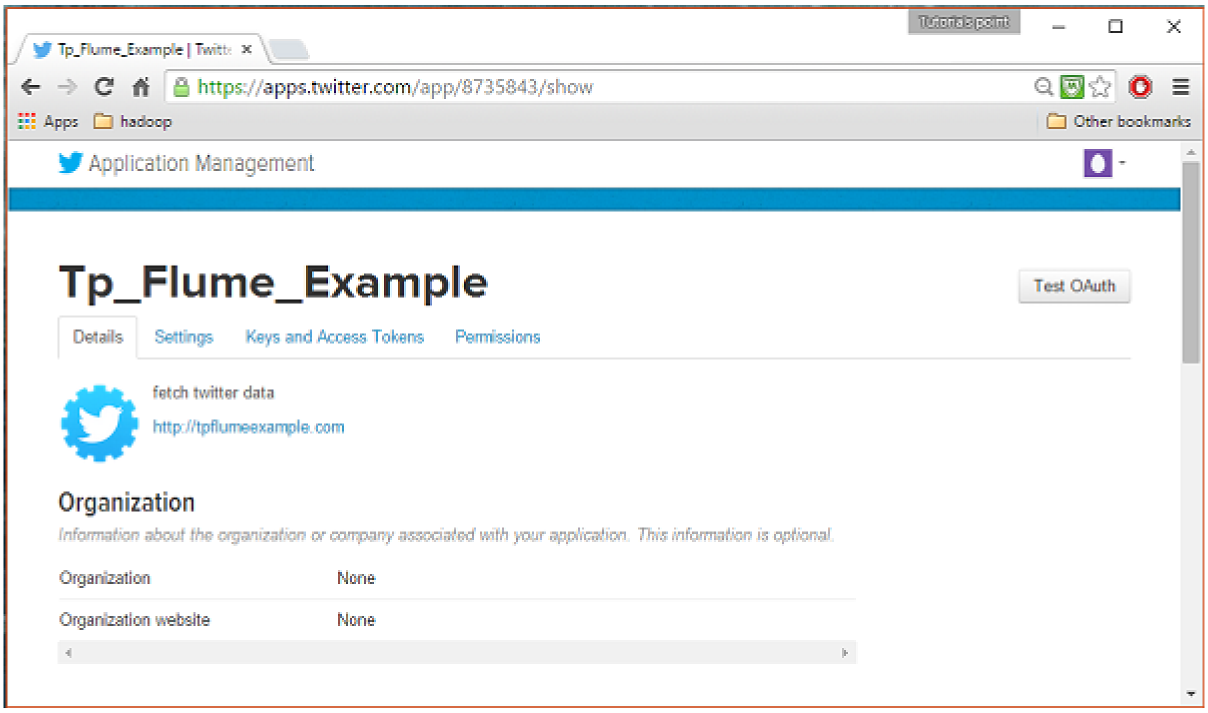
Website *

http://tpflumeexample.com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

## Step 3

Fill in the details, accept the **Developer Agreement** when finished, click on the **Create your Twitter application button** which is at the bottom of the page. If everything goes fine, an App will be created with the given details as shown below.



Tp_Flume_Example

fetch twitter data

http://tpflumeexample.com

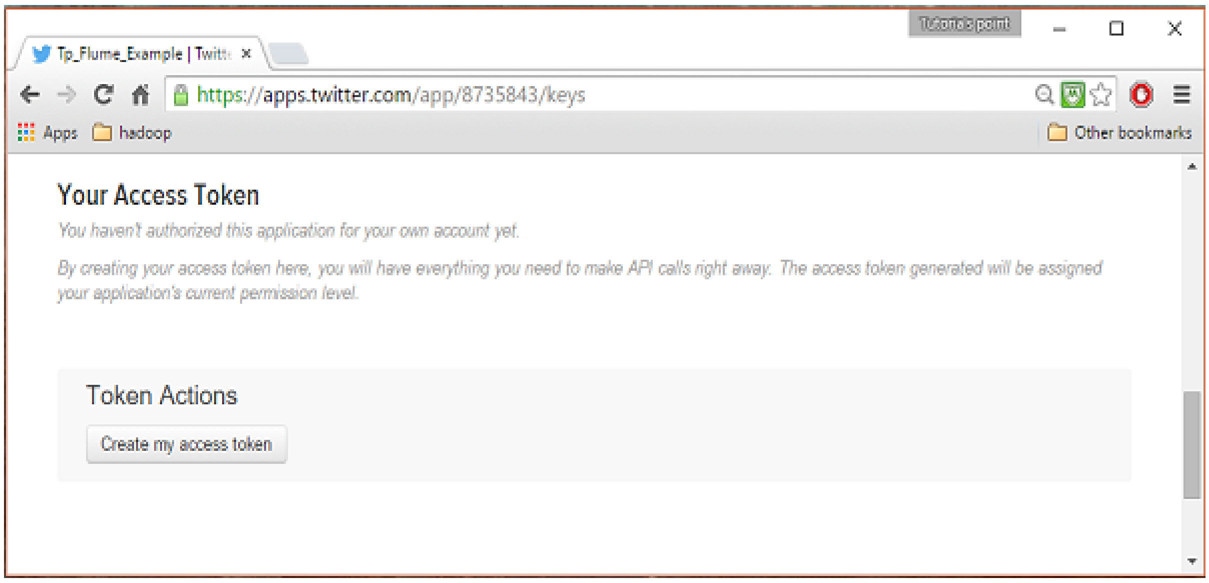
Organization

*Information about the organization or company associated with your application. This information is optional.*

| | |
|---|---|
| Organization | None |
| Organization website | None |

## Step 4

Under **keys and Access Tokens** tab at the bottom of the page, you can observe a button named **Create my access token.** Click on it to generate the access token.



Your Access Token

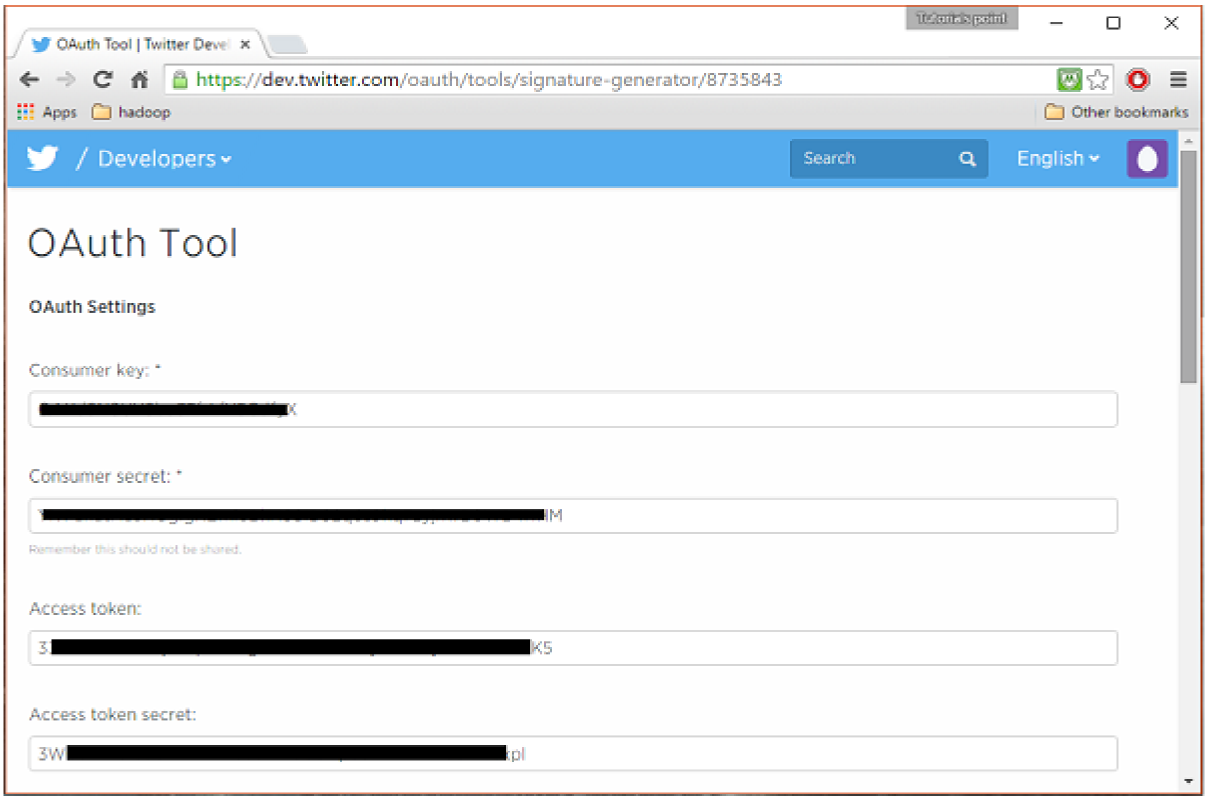
*You haven't authorized this application for your own account yet.*

*By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.*

Token Actions

## Step 5

Finally, click on the **Test OAuth** button which is on the right side top of the page. This will lead to a page which displays your **Consumer key**, **Consumer secret**, **Access token**, and **Access token secret**. Copy these details. These are useful to configure the agent in Flume.


OAuth Tool | Twitter Devel
https://dev.twitter.com/oauth/tools/signature-generator/8735843
Apps
hadoop
Other bookmarks
Developers
Search
English

OAuth Tool

OAuth Settings

Consumer key: *

[redacted]X

Consumer secret: *

[redacted]M

Remember this should not be shared.

Access token:

3[redacted]K5

Access token secret:

3W[redacted]kpl

## Starting HDFS

Since we are storing the data in HDFS, we need to install / verify Hadoop. Start Hadoop and create a folder in it to store Flume data. Follow the steps given below before configuring Flume.

### Step 1: Install / Verify Hadoop

Install Hadoop. If Hadoop is already installed in your system, verify the installation using Hadoop version command, as shown below.

```
$ hadoop version
```

If your system contains Hadoop, and if you have set the path variable, then you will get the following output:

```
Hadoop 2.6.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r
e3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled by jenkins on 2014-11-13T21:10Z
Compiled with protoc 2.5.0
```

```
From source with checksum 18e43357c8f927c0695f1e9522859d6a
This command was run using /home/Hadoop/hadoop/share/hadoop/common/hadoop-
common-2.6.0.jar
```

### Step 2: Starting Hadoop

Browse through the **sbin** directory of Hadoop and start yarn and Hadoop dfs (distributed file system) as shown below.

```
cd /$Hadoop_Home/sbin/
$ start-dfs.sh
localhost: starting namenode, logging to /home/Hadoop/hadoop/logs/hadoop-
Hadoop-namenode-localhost.localdomain.out
localhost: starting datanode, logging to /home/Hadoop/hadoop/logs/hadoop-
Hadoop-datanode-localhost.localdomain.out
Starting secondary namenodes [0.0.0.0]
starting secondarynamenode, logging to /home/Hadoop/hadoop/logs/hadoop-Hadoop-
secondarynamenode-localhost.localdomain.out


$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/Hadoop/hadoop/logs/yarn-Hadoop-
resourcemanager-localhost.localdomain.out
localhost: starting nodemanager, logging to /home/Hadoop/hadoop/logs/yarn-
Hadoop-nodemanager-localhost.localdomain.out
```

### Step 3: Create a Directory in HDFS

In Hadoop DFS, you can create directories using the command **mkdir**. Browse through it and create a directory with the name **twitter_data** in the required path as shown below.

```
$cd /$Hadoop_Home/bin/
$ hdfs dfs -mkdir hdfs://localhost:9000/user/Hadoop/twitter_data
```

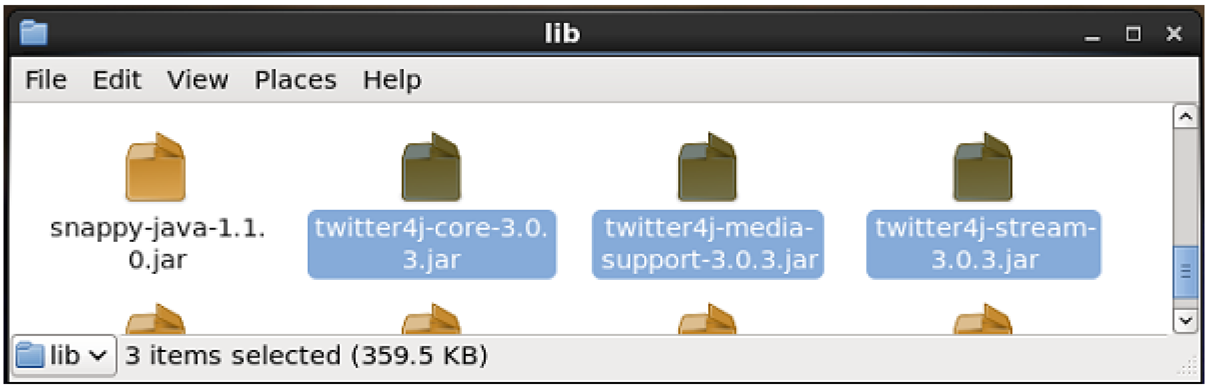
## Configuring Flume

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder. The example given in this chapter uses an experimental source provided by Apache Flume named **Twitter 1% Firehose** Memory channel and HDFS sink.

### Twitter 1% Firehose Source

This source is highly experimental. It connects to the 1% sample Twitter Firehose using streaming API and continuously downloads tweets, converts them to Avro format, and sends Avro events to a downstream Flume sink.

We will get this source by default along with the installation of Flume. The **jar** files corresponding to this source can be located in the **lib** folder as shown below.


lib
File Edit View Places Help
snappy-java-1.1.0.jar
twitter4j-core-3.0.3.jar
twitter4j-media-support-3.0.3.jar
twitter4j-stream-3.0.3.jar
lib
3 items selected (359.5 KB)

## Setting the classpath

Set the **classpath** variable to the **lib** folder of Flume in **Flume-env.sh** file as shown below.

```
export CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*
```

This source needs the details such as **Consumer key**, **Consumer secret**, **Access token**, and **Access token secret** of a Twitter application. While configuring this source, you have to provide values to the following properties:

- **Channels**
- **Source type : org.apache.flume.source.twitter.TwitterSource**
- **consumerKey :** The OAuth consumer key
- **consumerSecret :** OAuth consumer secret
- **accessToken :** OAuth access token
- **accessTokenSecret :** OAuth token secret
- **maxBatchSize :** Maximum no of twitter messages that should be in a twitter batch. The default value is 1000 (optional).
- **maxBatchDurationMillis :** Maximum number of milliseconds to wait before closing a batch. The default value is 1000 (optional).

## Channel

We are using the memory channel. To configure the memory channel, you *must* provide value to the type of the channel.

- **type**: It holds the type of the channel. In our example, the type is **MemChannel**.
- **Capacity**: It is the maximum number of events stored in the channel. Its default value is 100 (optional).

- **TransactionCapacity :** It is the maximum number of events the channel accepts or sends. Its default value is 100 (optional).

## HDFS Sink

This sink writes data into the HDFS. To configure this sink, you *must* provide the following details.

- **Channel**
- **type :** hdfs
- **hdfs.path :** the path of the directory in HDFS where data is to be stored.

And we can provide some optional values based on the scenario. Given below are the optional properties of the HDFS sink that we are configuring in our application.

- **fileType :** This is the required file format of our HDFS file. **SequenceFile**, **DataStream** and **CompressedStream** are the three types available with this stream. In our example, we are using the **DataStream**.

- **writeFormat :** Could be either text or writable.

- **batchSize :** It is the number of events written to a file before it is flushed into the HDFS. Its default value is 100.

- **rollsize :** It is the file size to trigger a roll. It default value is 100.

- **rollCount :** It is the number of events written into the file before it is rolled. Its default value is 10.

## Example – Configuration File

Given below is an example of the configuration file. Copy this content and save as **twitter .conf** in the conf folder of Flume.

```
# Naming the components on the current agent.

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


# Describing/Configuring the source
TwitterAgent.sources.Twitter.type =
org.apache.flume.source.twitter.TwitterSource
TwitterAgent.sources.Twitter.consumerKey = Your OAuth consumer key
TwitterAgent.sources.Twitter.consumerSecret = Your OAuth consumer secret
TwitterAgent.sources.Twitter.accessToken = Your OAuth consumer key access token
TwitterAgent.sources.Twitter.accessTokenSecret = Your OAuth consumer key access
token secret

TwitterAgent.sources.Twitter.keywords = tutorials point,java, bigdata,
mapreduce, mahout, hbase, nosql
```

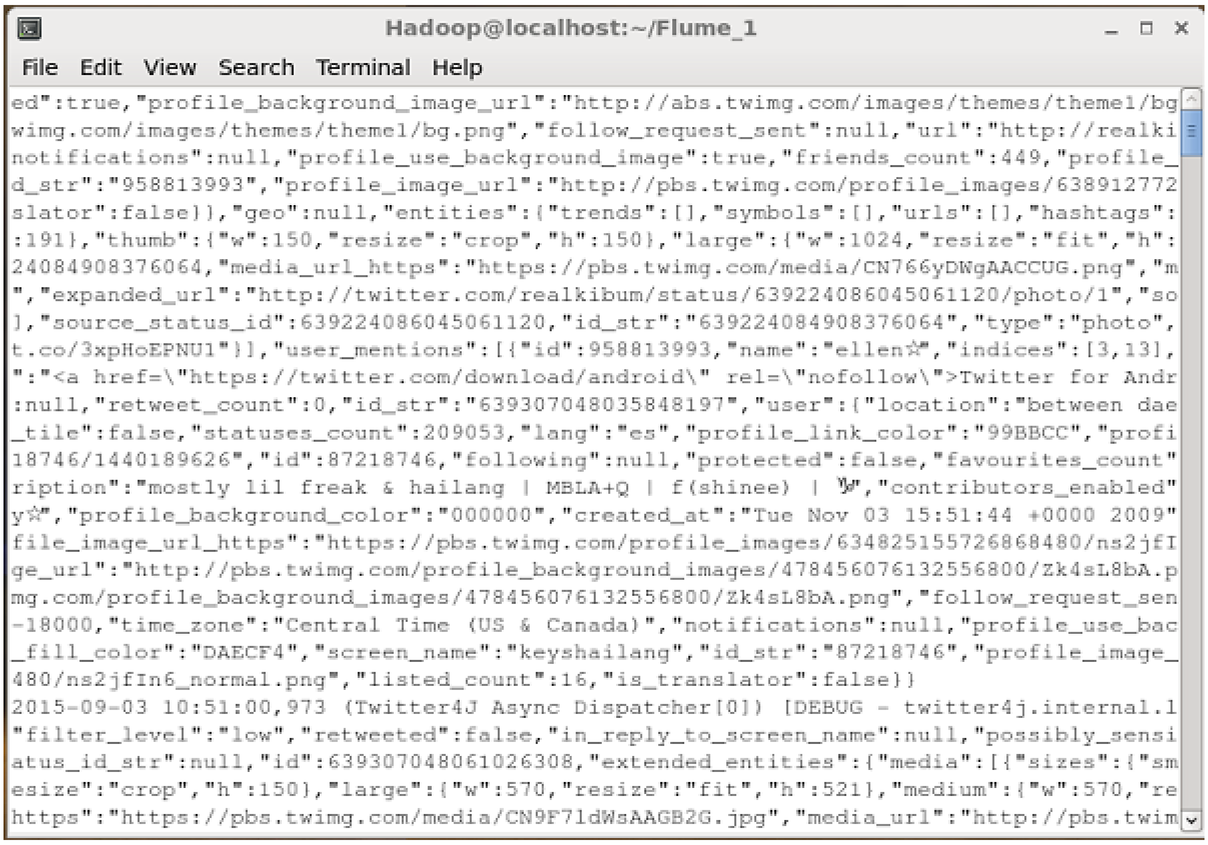
```
# Describing/Configuring the sink

TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path =
hdfs://localhost:9000/user/Hadoop/twitter_data/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000


# Describing/Configuring the channel
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100


# Binding the source and sink to the channel

TwitterAgent.sources.Twitter.channels = MemChannel

TwitterAgent.sinks.HDFS.channel = MemChannel
```

## Execution

Browse through the Flume home directory and execute the application as shown below.

```
$ cd $FLUME_HOME

$ bin/flume-ng agent --conf ./conf/ -f conf/twitter.conf -

Dflume.root.logger=DEBUG,console -n TwitterAgent
```

If everything goes fine, the streaming of tweets into HDFS will start. Given below is the snapshot of the command prompt window while fetching tweets.


Hadoop@localhost:~/Flume_1
File Edit View Search Terminal Help

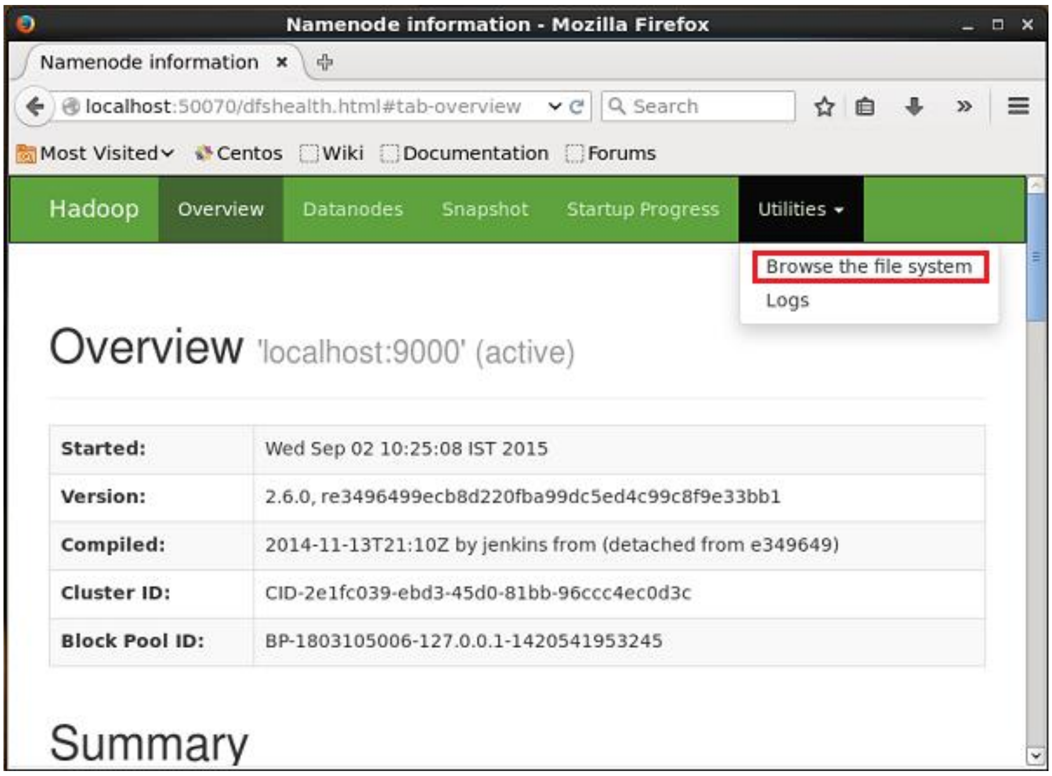
```
ed":true,"profile_background_image_url":"http://abs.twimg.com/images/themes/theme1/bg
wimg.com/images/themes/theme1/bg.png","follow_request_sent":null,"url":"http://realki
notifications":null,"profile_use_background_image":true,"friends_count":449,"profile_
d_str":"958813993","profile_image_url":"http://pbs.twimg.com/profile_images/638912772
slator":false}},"geo":null,"entities":{"trends":[],"symbols":[],"urls":[],"hashtags":
:191},"thumb":{"w":150,"resize":"crop","h":150},"large":{"w":1024,"resize":"fit","h":
24084908376064,"media_url_https":"https://pbs.twimg.com/media/CN766yDWgAACCUG.png","m
","expanded_url":"http://twitter.com/realkibum/status/639224086045061120/photo/1","so
],"source_status_id":639224086045061120,"id_str":"639224084908376064","type":"photo",
t.co/3xpHoEPNU1"}],"user_mentions":[{"id":958813993,"name":"ellen☆","indices":[3,13],
":"<a href=\"https://twitter.com/download/android\" rel=\"nofollow\">Twitter for Andr
:null,"retweet_count":0,"id_str":"639307048035848197","user":{"location":"between dae
_tile":false,"statuses_count":209053,"lang":"es","profile_link_color":"99BBCC","profi
18746/1440189626","id":87218746,"following":null,"protected":false,"favourites_count"
ription":"mostly lil freak & hailang | MBLA+Q | f(shinee) | ꦿ","contributors_enabled"
y☆","profile_background_color":"000000","created_at":"Tue Nov 03 15:51:44 +0000 2009"
file_image_url_https":"https://pbs.twimg.com/profile_images/634825155726868480/ns2jfI
ge_url":"http://pbs.twimg.com/profile_background_images/478456076132556800/Zk4sL8bA.p
mg.com/profile_background_images/478456076132556800/Zk4sL8bA.png","follow_request_sen
-18000,"time_zone":"Central Time (US & Canada)","notifications":null,"profile_use_bac
_fill_color":"DAECF4","screen_name":"keyshailang","id_str":"87218746","profile_image_
480/ns2jfIn6_normal.png","listed_count":16,"is_translator":false}}
2015-09-03 10:51:00,973 (Twitter4J Async Dispatcher[0]) [DEBUG - twitter4j.internal.l
"filter_level":"low","retweeted":false,"in_reply_to_screen_name":null,"possibly_sensi
atus_id_str":null,"id":639307048061026308,"extended_entities":{"media":[{"sizes":{"sm
esize":"crop","h":150},"large":{"w":570,"resize":"fit","h":521},"medium":{"w":570,"re
https":"https://pbs.twimg.com/media/CN9F71dWsAAGB2G.jpg","media_url":"http://pbs.twim
```

## Verifying HDFS

You can access the Hadoop Administration Web UI using the URL given below.

```
http://localhost:50070/
```

Click on the dropdown named **Utilities** on the right-hand side of the page. You can see two options as shown in the snapshot given below.

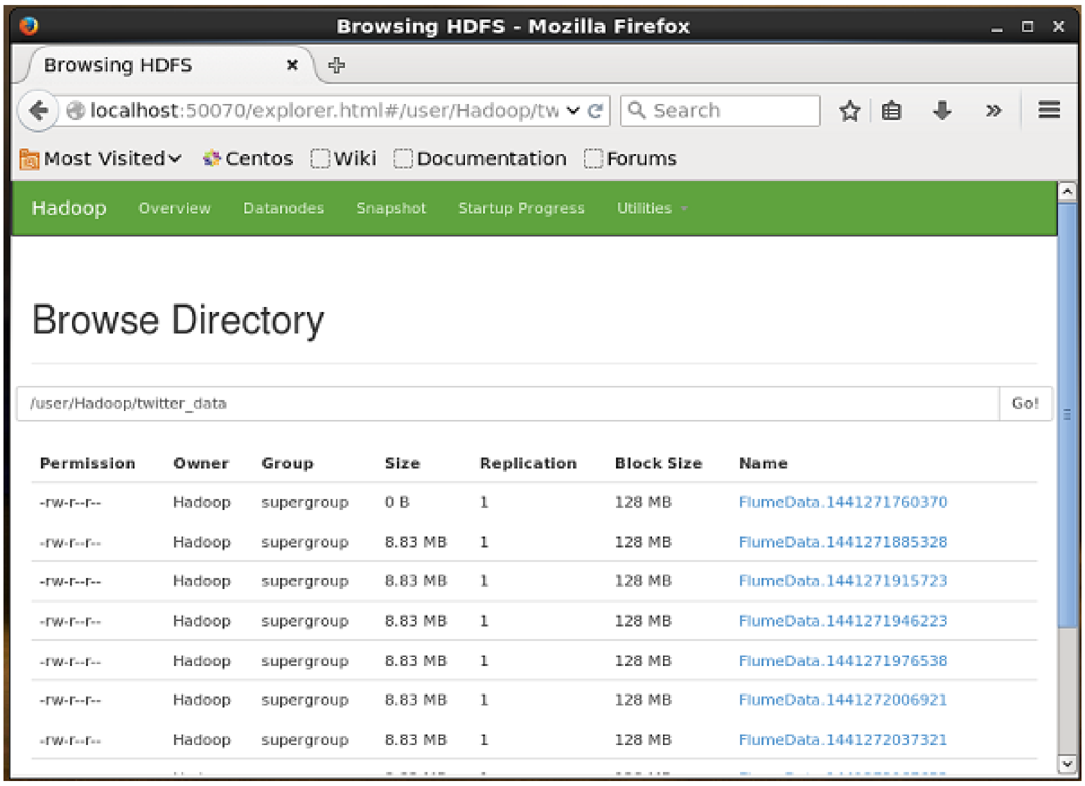


Overview 'localhost:9000' (active)

| | |
|---|---|
| **Started:** | Wed Sep 02 10:25:08 IST 2015 |
| **Version:** | 2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1 |
| **Compiled:** | 2014-11-13T21:10Z by jenkins from (detached from e349649) |
| **Cluster ID:** | CID-2e1fc039-ebd3-45d0-81bb-96ccc4ec0d3c |
| **Block Pool ID:** | BP-1803105006-127.0.0.1-1420541953245 |

Summary

Click on **Browse the file system** and enter the path of the HDFS directory where you have stored the tweets. In our example, the path will be **/user/Hadoop/twitter_data/.** Then, you can see the list of twitter log files stored in HDFS as given below.



Browse Directory

/user/Hadoop/twitter_data

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | Hadoop | supergroup | 0 B | 1 | 128 MB | FlumeData.1441271760370 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441271885328 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441271915723 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441271946223 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441271976538 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441272006921 |
| -rw-r--r-- | Hadoop | supergroup | 8.83 MB | 1 | 128 MB | FlumeData.1441272037321 |

# 8. SEQUENCE GENERATOR SOURCE

In the previous chapter, we have seen how to fetch data from twitter source to HDFS. This chapter explains how to fetch data from **Sequence generator**.

## Prerequisites

To run the example provided in this chapter, you need to install **HDFS** along with **Flume.** Therefore, verify Hadoop installation and start the HDFS before proceeding further. (Refer the previous chapter to learn how to start the HDFS).

## Configuring Flume

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder. The example given in this chapter uses a **sequence generator source**, a **memory channel**, and an **HDFS sink**.

## Sequence Generator Source

It is the source that generates the events continuously. It maintains a counter that starts from 0 and increments by 1. It is used for testing purpose. While configuring this source, you must provide values to the following properties:

- **Channels**
- **Source type :** seq

## Channel

We are using the **memory** channel. To configure the memory channel, you *must* provide a value to the type of the channel. Given below are the list of properties that you need to supply while configuring the memory channel:

- **type**: It holds the type of the channel. In our example the type is MemChannel.

- **Capacity**: It is the maximum number of events stored in the channel. Its default value is 100. (optional)

- **TransactionCapacity :** It is the maximum number of events the channel accepts or sends. Its default is 100. (optional).

## HDFS Sink

This sink writes data into the HDFS. To configure this sink, you *must* provide the following details.

- **Channel**
- **type :** hdfs
- **hdfs.path :** the path of the directory in HDFS where data is to be stored.

And we can provide some optional values based on the scenario. Given below are the optional properties of the HDFS sink that we are configuring in our application.

- **fileType :** This is the required file format of our HDFS file. **SequenceFile**, **DataStream** and **CompressedStream** are the three types available with this stream. In our example, we are using the **DataStream**.

- **writeFormat :** Could be either text or writable.

- **batchSize :** It is the number of events written to a file before it is flushed into the HDFS. Its default value is 100.

- **rollsize :** It is the file size to trigger a roll. It default value is 100.

- **rollCount :** It is the number of events written into the file before it is rolled. Its default value is 10.

## Example – Configuration File

Given below is an example of the configuration file. Copy this content and save as **seq_gen .conf** in the conf folder of Flume.

```
# Naming the components on the current agent

SeqGenAgent.sources = SeqSource
SeqGenAgent.channels = MemChannel
SeqGenAgent.sinks = HDFS


# Describing/Configuring the source
SeqGenAgent.sources.SeqSource.type = seq


# Describing/Configuring the sink
SeqGenAgent.sinks.HDFS.type = hdfs
SeqGenAgent.sinks.HDFS.hdfs.path =
hdfs://localhost:9000/user/Hadoop/seqgen_data/
SeqGenAgent.sinks.HDFS.hdfs.filePrefix = log
SeqGenAgent.sinks.HDFS.hdfs.rollInterval = 0
SeqGenAgent.sinks.HDFS.hdfs.rollCount = 10000
SeqGenAgent.sinks.HDFS.hdfs.fileType = DataStream


# Describing/Configuring the channel

SeqGenAgent.channels.MemChannel.type = memory
SeqGenAgent.channels.MemChannel.capacity = 1000
SeqGenAgent.channels.MemChannel.transactionCapacity = 100



# Binding the source and sink to the channel
SeqGenAgent.sources.SeqSource.channels = MemChannel
SeqGenAgent.sinks.HDFS.channel = MemChannel
```

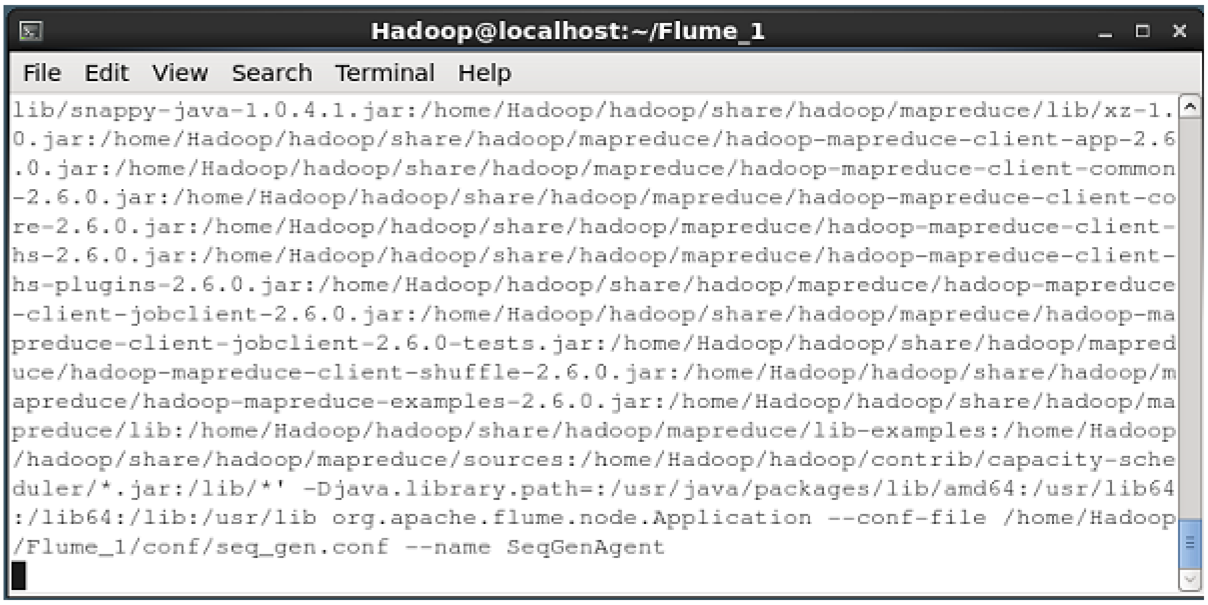
## Execution

Browse through the Flume home directory and execute the application as shown below.

```
$ cd $FLUME_HOME
$./bin/flume-ng agent --conf $FLUME_CONF --conf-file $FLUME_CONF/seq_gen.conf -
-name SeqGenAgent
```

If everything goes fine, the source starts generating sequence numbers which will be pushed into the HDFS in the form of log files.

Given below is a snapshot of the command prompt window fetching the data generated by the sequence generator into the HDFS.


Hadoop@localhost:~/Flume_1
File Edit View Search Terminal Help

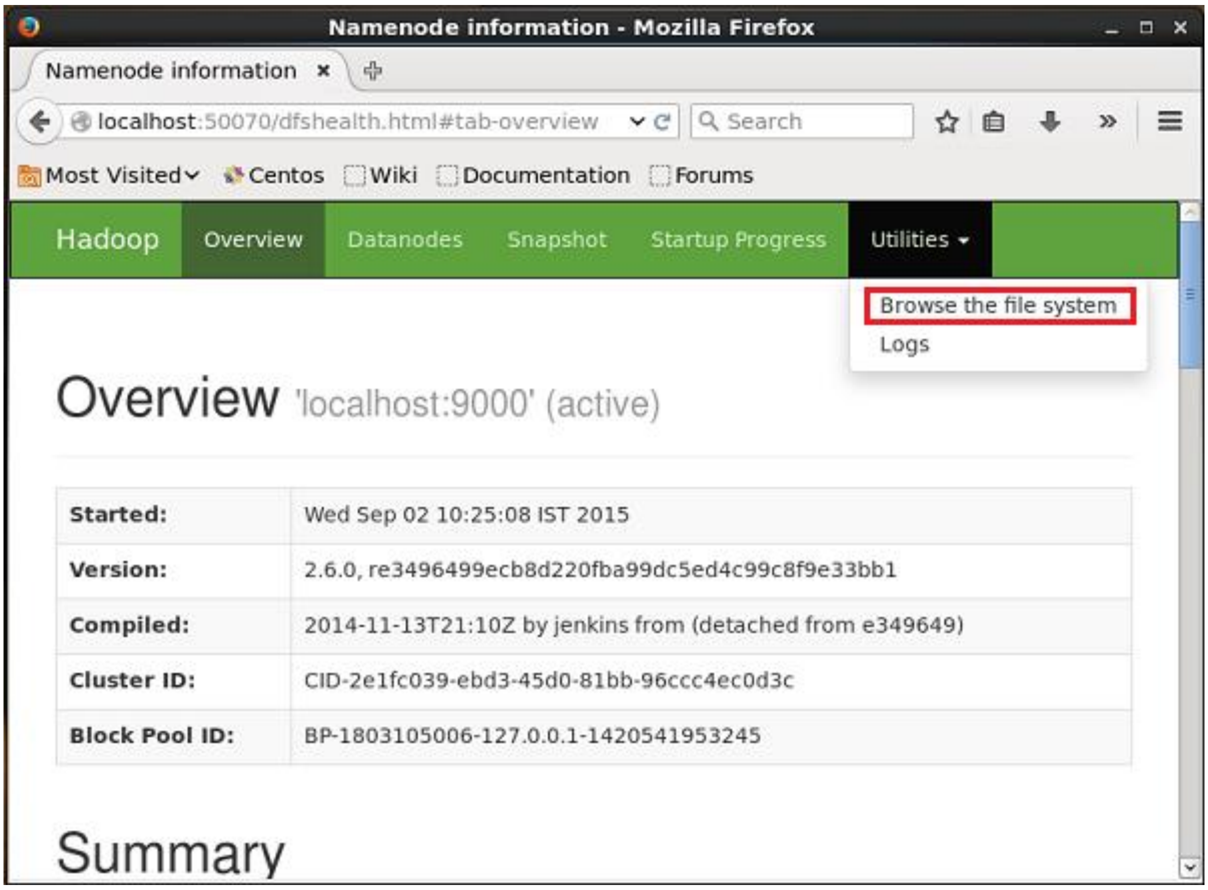
```
lib/snappy-java-1.0.4.1.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/lib/xz-1.
0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-app-2.6
.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-common
-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-co
re-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-
hs-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-
hs-plugins-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce
-client-jobclient-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/mapreduce/hadoop-ma
preduce-client-jobclient-2.6.0-tests.jar:/home/Hadoop/hadoop/share/hadoop/mapred
uce/hadoop-mapreduce-client-shuffle-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/m
apreduce/hadoop-mapreduce-examples-2.6.0.jar:/home/Hadoop/hadoop/share/hadoop/ma
preduce/lib:/home/Hadoop/hadoop/share/hadoop/mapreduce/lib-examples:/home/Hadoop
/hadoop/share/hadoop/mapreduce/sources:/home/Hadoop/hadoop/contrib/capacity-sche
duler/*.jar:/lib/*' -Djava.library.path=:/usr/java/packages/lib/amd64:/usr/lib64
:/lib64:/lib:/usr/lib org.apache.flume.node.Application --conf-file /home/Hadoop
/Flume_1/conf/seq_gen.conf --name SeqGenAgent
```

## Verifying the HDFS

You can access the Hadoop Administration Web UI using the following URL:

```
http://localhost:50070/
```

Click on the dropdown named **Utilities** on the right-hand side of the page. You can see two options as shown in the diagram given below.



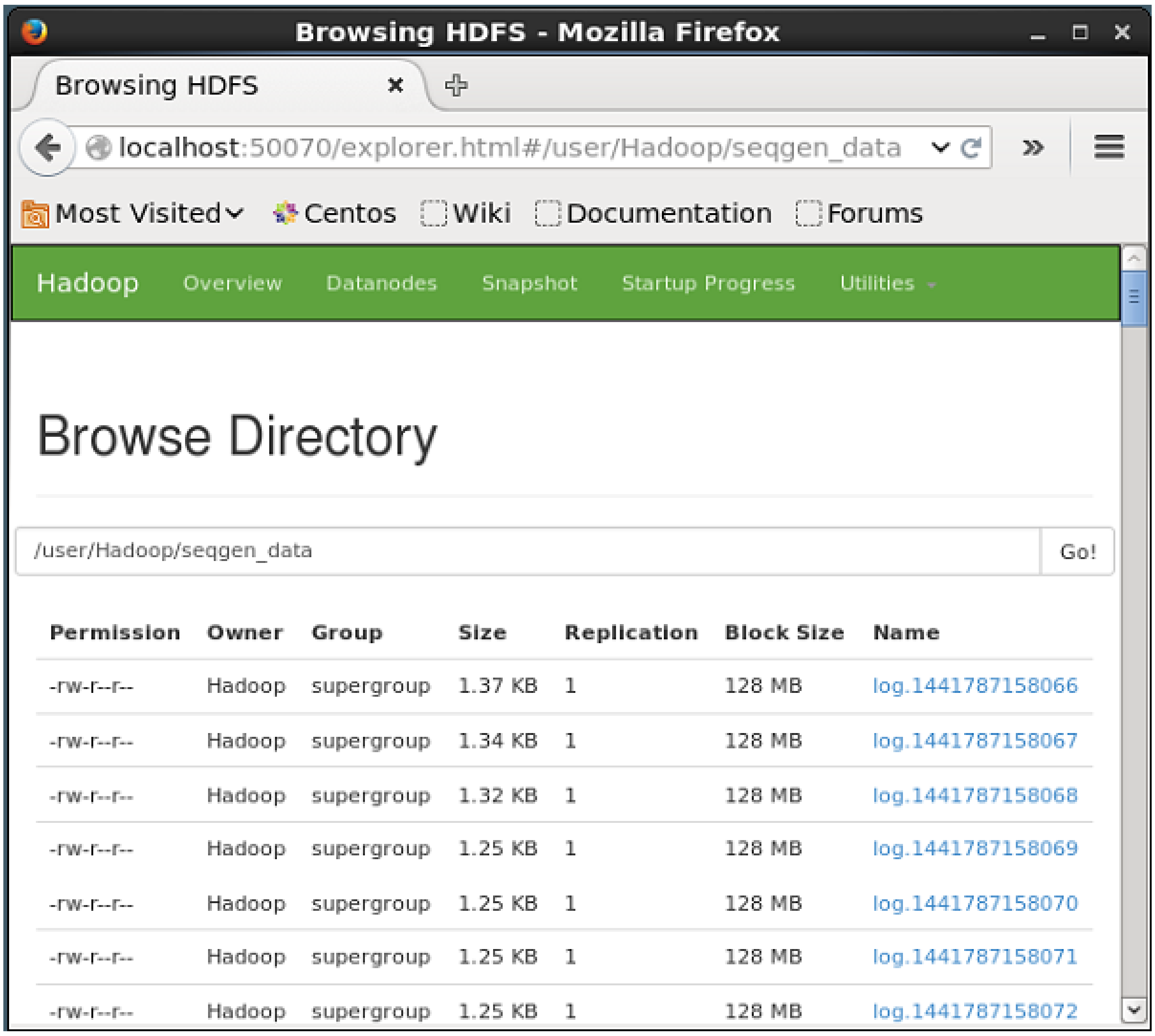
Overview 'localhost:9000' (active)

| | |
|---|---|
| **Started:** | Wed Sep 02 10:25:08 IST 2015 |
| **Version:** | 2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1 |
| **Compiled:** | 2014-11-13T21:10Z by jenkins from (detached from e349649) |
| **Cluster ID:** | CID-2e1fc039-ebd3-45d0-81bb-96ccc4ec0d3c |
| **Block Pool ID:** | BP-1803105006-127.0.0.1-1420541953245 |

Summary

Click on **Browse the file system** and enter the path of the HDFS directory where you have stored the data generated by the sequence generator.

In our example, the path will be **/user/Hadoop/ seqgen_data /**. Then, you can see the list of log files generated by the sequence generator, stored in the HDFS as given below.



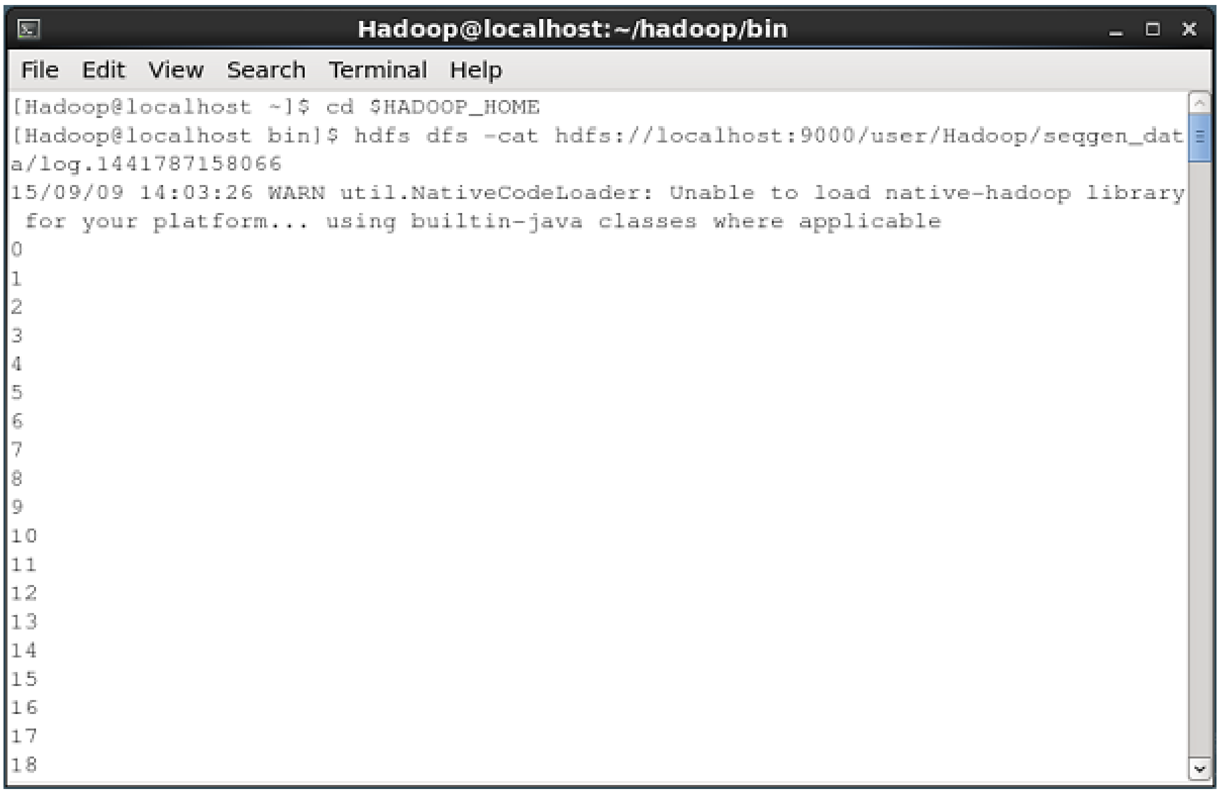
Browse Directory

/user/Hadoop/seqgen_data

| Permission | Owner | Group | Size | Replication | Block Size | Name |
|---|---|---|---|---|---|---|
| -rw-r--r-- | Hadoop | supergroup | 1.37 KB | 1 | 128 MB | log.1441787158066 |
| -rw-r--r-- | Hadoop | supergroup | 1.34 KB | 1 | 128 MB | log.1441787158067 |
| -rw-r--r-- | Hadoop | supergroup | 1.32 KB | 1 | 128 MB | log.1441787158068 |
| -rw-r--r-- | Hadoop | supergroup | 1.25 KB | 1 | 128 MB | log.1441787158069 |
| -rw-r--r-- | Hadoop | supergroup | 1.25 KB | 1 | 128 MB | log.1441787158070 |
| -rw-r--r-- | Hadoop | supergroup | 1.25 KB | 1 | 128 MB | log.1441787158071 |
| -rw-r--r-- | Hadoop | supergroup | 1.25 KB | 1 | 128 MB | log.1441787158072 |

## Verifying the Contents of the File

All these log files contain numbers in sequential format. You can verify the contents of these file in the file system using the **cat** command as shown below.


Hadoop@localhost:~/hadoop/bin
File Edit View Search Terminal Help

```
[Hadoop@localhost ~]$ cd $HADOOP_HOME
[Hadoop@localhost bin]$ hdfs dfs -cat hdfs://localhost:9000/user/Hadoop/seqgen_dat
a/log.1441787158066
15/09/09 14:03:26 WARN util.NativeCodeLoader: Unable to load native-hadoop library
 for your platform... using builtin-java classes where applicable
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

# 9. NETCAT SOURCE

This chapter takes an example to explain how you can generate events and subsequently log them into the console. For this, we are using the **NetCat** source and the **logger** sink.

## Prerequisites

To run the example provided in this chapter, you need to install **Flume.**

## Configuring Flume

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder. The example given in this chapter uses a **NetCat Source, Memory channel**, and a **logger sink.**

## NetCat Source

While configuring the NetCat source, we have to specify a port while configuring the source. Now the source (NetCat source) listens to the given port and receives each line we entered in that port as an individual event and transfers it to the sink through the specified channel.

While configuring this source, you have to provide values to the following properties:

- **channels**
- **Source type :** netcat
- **bind :** Host name or IP address to bind.
- **port :** Port number to which we want the source to listen.

## Channel

We are using the **memory** channel. To configure the memory channel, you *must* provide a value to the type of the channel. Given below are the list of properties that you need to supply while configuring the memory channel:

- **type**: It holds the type of the channel. In our example, the type is **MemChannel**.
- **Capacity**: It is the maximum number of events stored in the channel. Its default value is 100. (optional)
- **TransactionCapacity :** It is the maximum number of events the channel accepts or sends. Its default value is 100. (optional).

## Logger Sink

This sink logs all the events passed to it. Generally, it is used for testing or debugging purpose. To configure this sink, you must provide the following details.

- **Channel**

- **type :** logger

## Example Configuration File

Given below is an example of the configuration file. Copy this content and save as **netcat .conf** in the conf folder of Flume.

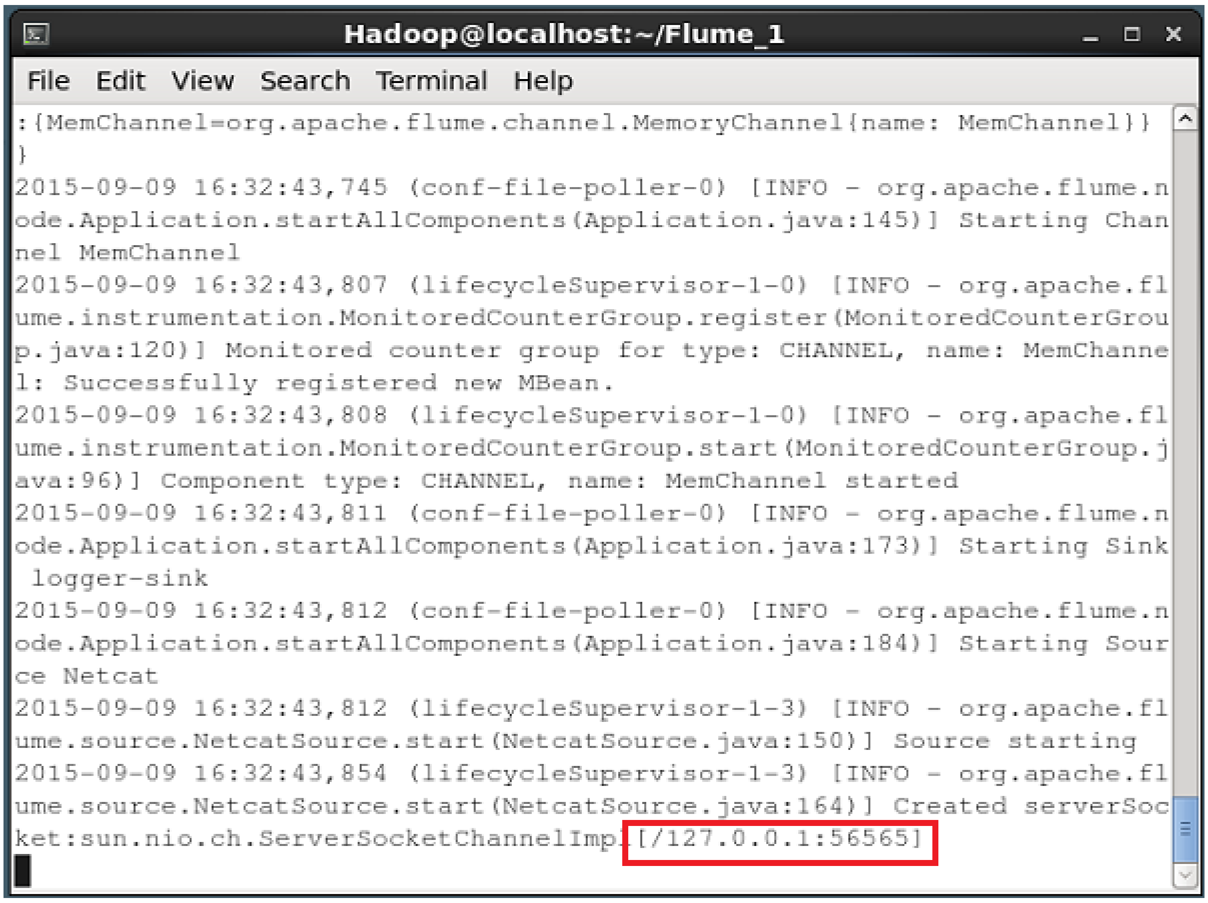
```
# Naming the components on the current agent
NetcatAgent.sources = Netcat
NetcatAgent.channels = MemChannel
NetcatAgent.sinks = LoggerSink

# Describing/Configuring the source
NetcatAgent.sources.Netcat.type = netcat
NetcatAgent.sources.Netcat.bind = localhost
NetcatAgent.sources.Netcat.port = 56565

# Describing/Configuring the sink
NetcatAgent.sinks.LoggerSink.type = logger

# Describing/Configuring the channel
NetcatAgent.channels.MemChannel.type = memory
NetcatAgent.channels.MemChannel.capacity = 1000
NetcatAgent.channels.MemChannel.transactionCapacity = 100

# Bind the source and sink to the channel
NetcatAgent.sources.Netcat.channels = MemChannel
NetcatAgent.sinks. LoggerSink.channel = MemChannel
```

## Execution

Browse through the Flume home directory and execute the application as shown below.

```
$ cd $FLUME_HOME
$ ./bin/flume-ng agent --conf $FLUME_CONF --conf-file $FLUME_CONF/netcat.conf -
-name NetcatAgent -Dflume.root.logger=INFO,console
```

If everything goes fine, the source starts listening to the given port. In this case, it is **56565**. Given below is the snapshot of the command prompt window of a NetCat source which has started and listening to the port 56565.


Hadoop@localhost:~/Flume_1
File Edit View Search Terminal Help

```
:{MemChannel=org.apache.flume.channel.MemoryChannel{name: MemChannel}}
}
2015-09-09 16:32:43,745 (conf-file-poller-0) [INFO - org.apache.flume.n
ode.Application.startAllComponents(Application.java:145)] Starting Chan
nel MemChannel
2015-09-09 16:32:43,807 (lifecycleSupervisor-1-0) [INFO - org.apache.fl
ume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGrou
p.java:120)] Monitored counter group for type: CHANNEL, name: MemChanne
l: Successfully registered new MBean.
2015-09-09 16:32:43,808 (lifecycleSupervisor-1-0) [INFO - org.apache.fl
ume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.j
ava:96)] Component type: CHANNEL, name: MemChannel started
2015-09-09 16:32:43,811 (conf-file-poller-0) [INFO - org.apache.flume.n
ode.Application.startAllComponents(Application.java:173)] Starting Sink
 logger-sink
2015-09-09 16:32:43,812 (conf-file-poller-0) [INFO - org.apache.flume.n
ode.Application.startAllComponents(Application.java:184)] Starting Sour
ce Netcat
2015-09-09 16:32:43,812 (lifecycleSupervisor-1-3) [INFO - org.apache.fl
ume.source.NetcatSource.start(NetcatSource.java:150)] Source starting
2015-09-09 16:32:43,854 (lifecycleSupervisor-1-3) [INFO - org.apache.fl
ume.source.NetcatSource.start(NetcatSource.java:164)] Created serverSoc
ket:sun.nio.ch.ServerSocketChannelImpl[/127.0.0.1:56565]
```

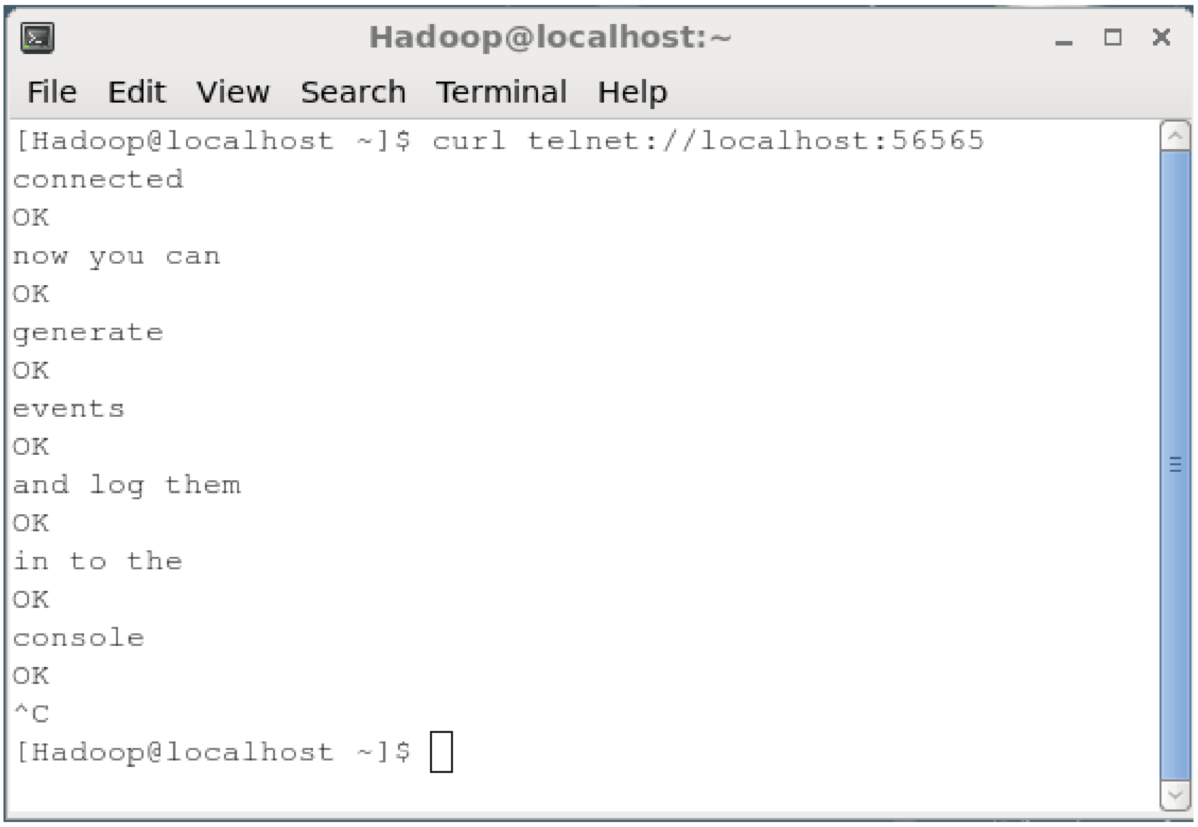
## Passing Data to the Source

To pass data to NetCat source, you have to open the port given in the configuration file. Open a separate terminal and connect to the source (56565) using the **curl** command. When the connection is successful, you will get a message "**connected**" as shown below.

```
$ curl telnet://localhost:56565

connected
```

Now you can enter your data line by line (after each line, you have to press Enter). The NetCat source receives each line as an individual event and you will get a received message "**OK**".

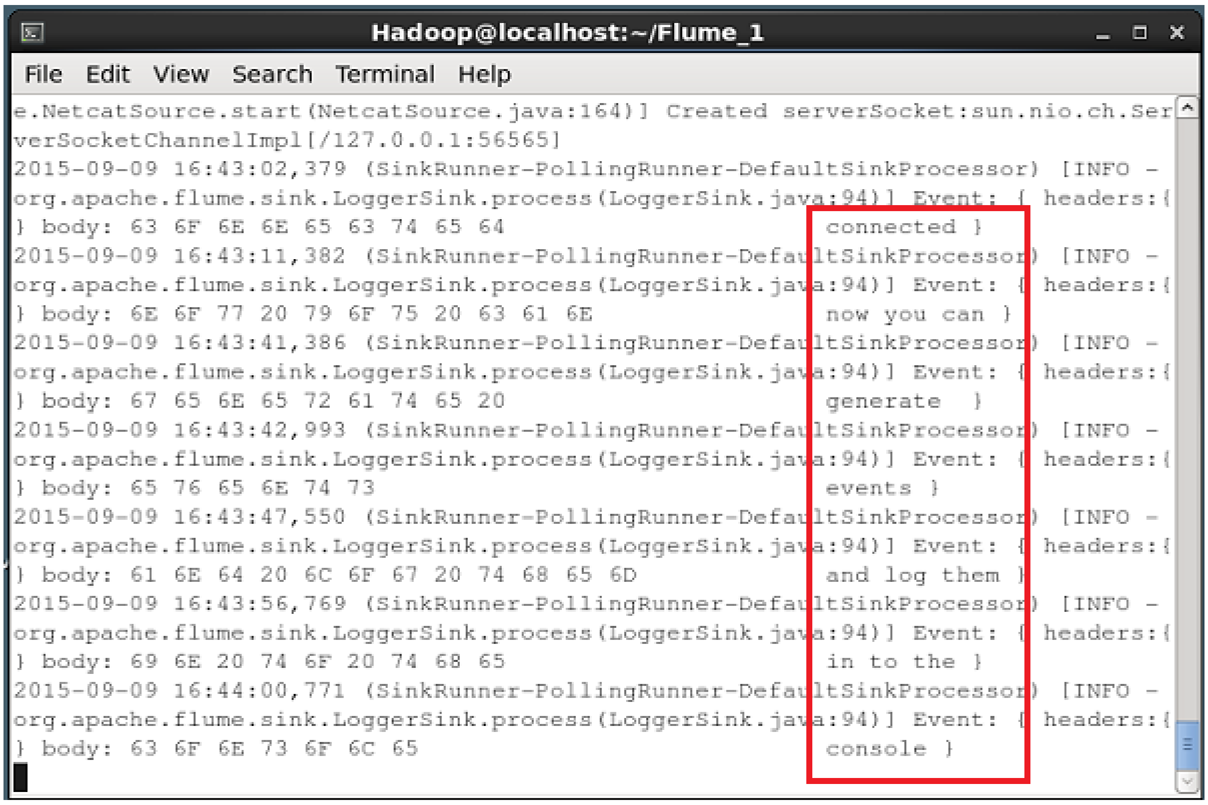
Whenever you are done with passing data, you can exit the console by pressing (**Ctrl+C**). Given below is the snapshot of the console where we have connected to the source using the **curl** command.



```
[Hadoop@localhost ~]$ curl telnet://localhost:56565
connected
OK
now you can
OK
generate
OK
events
OK
and log them
OK
in to the
OK
console
OK
^C
[Hadoop@localhost ~]$
```

Each line that is entered in the above console will be received as an individual event by the source. Since we have used the **Logger** sink, these events will be logged on to the console (source console) through the specified channel (memory channel in this case).

The following snapshot shows the NetCat console where the events are logged.


Hadoop@localhost:~/Flume_1
File Edit View Search Terminal Help

```
e.NetcatSource.start(NetcatSource.java:164)] Created serverSocket:sun.nio.ch.Ser
verSocketChannelImpl[/127.0.0.1:56565]
2015-09-09 16:43:02,379 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 63 6F 6E 6E 65 63 74 65 64                connected }
2015-09-09 16:43:11,382 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 6E 6F 77 20 79 6F 75 20 63 61 6E          now you can }
2015-09-09 16:43:41,386 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 67 65 6E 65 72 61 74 65 20                generate  }
2015-09-09 16:43:42,993 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 65 76 65 6E 74 73                         events }
2015-09-09 16:43:47,550 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 61 6E 64 20 6C 6F 67 20 74 68 65 6D       and log them }
2015-09-09 16:43:56,769 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 69 6E 20 74 6F 20 74 68 65                in to the }
2015-09-09 16:44:00,771 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{
} body: 63 6F 6E 73 6F 6C 65                      console }
```