UNIVERSITÀ DEGLI STUDI DI MESSINA
Dipartimento di Ingegneria

Corso di Laurea Magistrale in

Engineering and Computer Science (LM-18

LM-32)

Subject: Big Data

Mastering data ingestion with Apache Flume

Student:
Abdirashova Aruzhan

Docente:
Prof. Antonio Celesti

A.A. 2023

Table of Contents:

1. Introduction

My report will focus on Apache Flume, a tool assisting organizations in streaming sizable log files from diverse sources to distributed data storage, such as Hadoop HDFS. It highlights Apache Flume's features and capabilities, illustrating its role in enhancing the efficient processing of data for reporting and analytical purposes in applications.

2. Apache Flume Overview

2.1 Introduction to Apache Flume

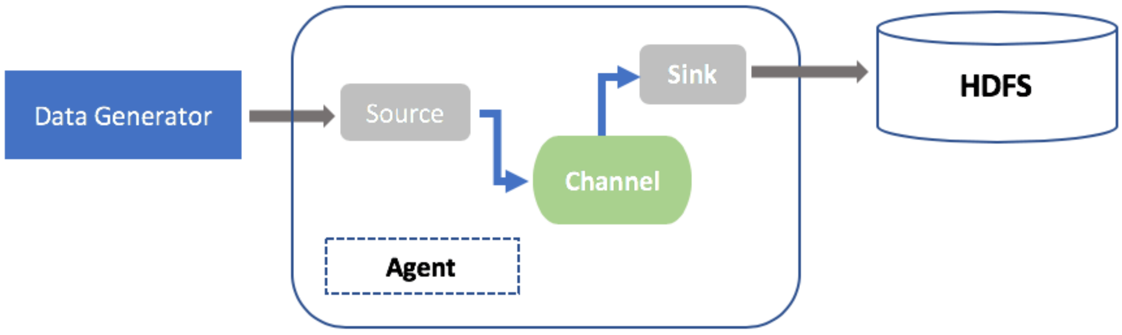
Apache Flume stands out as an effective, distributed, reliable, and fault-tolerant data-ingestion tool. It excels in streaming substantial volumes of log files from diverse sources, such as web servers, into storage solutions like the Hadoop Distributed File System (HDFS) or distributed databases like HBase on HDFS. Moreover, Flume extends its capabilities to destinations like Elasticsearch, achieving near-real-time speeds. Beyond log data, Flume adeptly handles streaming event data from web sources like Twitter, Facebook, and Kafka Brokers.

2.2 The History of Apache Flume

Apache Flume was created by Cloudera Data Platform. They created it in order to efficiently and dependably stream substantial quantities of web server log files into Hadoop, for comprehensive analysis in a distributed environment. Originally designed for log data, Apache Flume was later enhanced to manage event data as well.

2.3 Architecture and Components

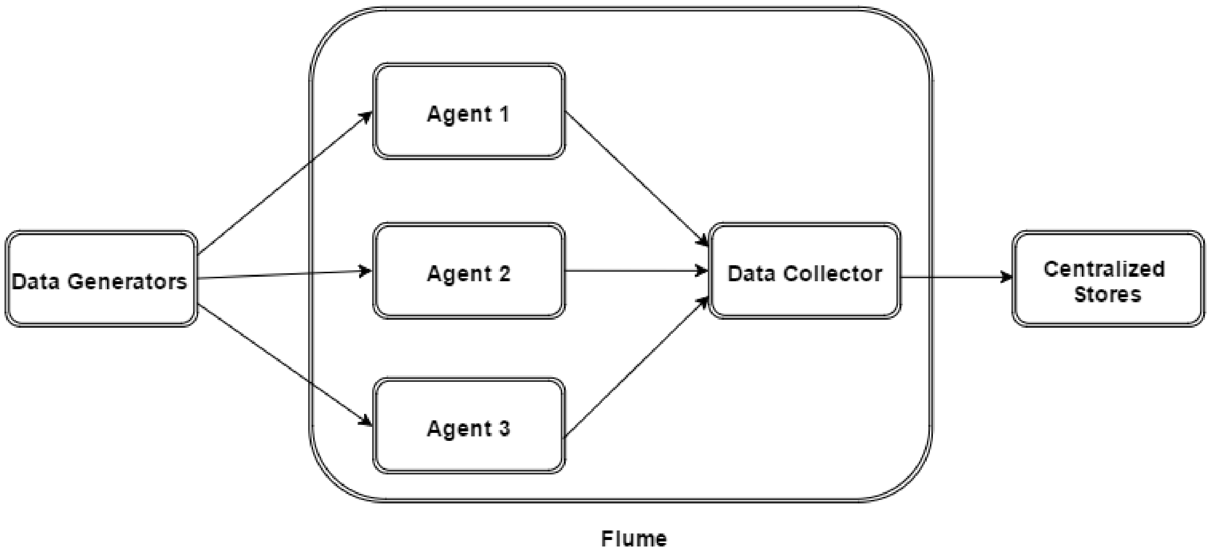
Apache Flume's data streaming architecture comprises three key elements: data-generating sources, the Flume agent, and the destination (or target). The Flume agent is composed of the source, channel, and sink. The source collects log files from various sources, such as web servers and Twitter, and transfers them to the channel. The sink component of Flume ensures that the received data is synchronized with the chosen destination, which could be HDFS, an HBase database on HDFS, or an analytics tool like Spark.



Pic-1. Basic architecture of Flume for an HDFS sink

In Apache Flume, the source, channel, and sink constitute the Flume agent. Scaling for large data volumes involves configuring multiple Flume agents to gather data from various sources, enabling parallel streaming to multiple destinations.

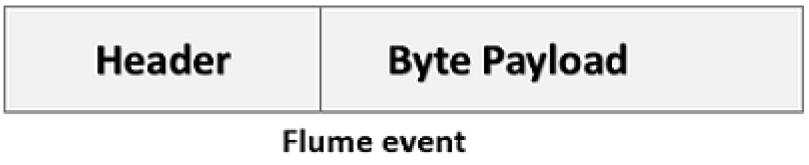
The architecture of Flume that follows, illustrates data generators like Facebook and Twitter producing data collected by individual Flume agents. Subsequently, a data collector, also functioning as an agent, gathers the data from these agents, aggregates it, and then pushes it into a centralized store, such as HDFS or HBase.



Pic-2. The architecture of Flume

Flume event

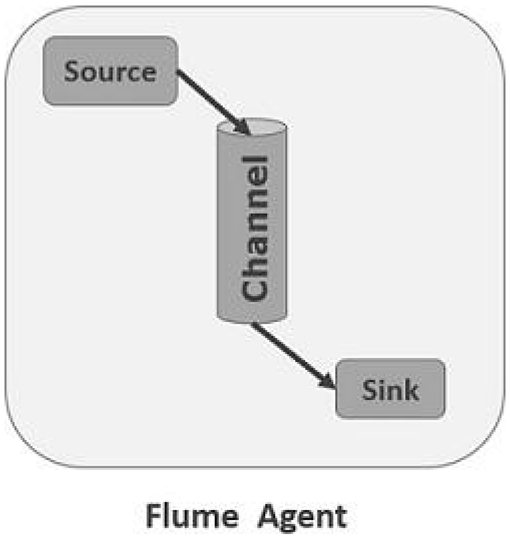
In Flume, an event serves as the fundamental unit for transporting data, encompassing a payload in the form of a byte array intended for transit from the source to the destination. Additionally, it may include optional headers.



Pic-3. Flume event

Flume agent

In Flume, an agent is a standalone daemon process (JVM) responsible for receiving data (events) from clients or other agents and forwarding it to its designated next destination, which can be a sink or another agent. Flume can utilize multiple agents concurrently. The following diagram represents a Flume Agent



Pic-4. Flume agent

As it is shown in the diagram, a flume agent contains three main components : the source, the channel and the sink.

Source

In Flume, a source is the agent component responsible for receiving data from data generators and transmitting it to one or more channels as Flume events. Apache Flume offers various source types, each designed to receive events from specific data generators. Examples include Avro source, Thrift source, and the Twitter 1% source.

Channel

A channel is a temporary storage area that receives events from a source and buffers them until they are consumed by the sink. It acts as a bridge between the source and the receiver. These channels are completely flexible and can be used on any surface and sink. Examples include JDBC pipes, file system pipes, memory pipes, etc.

Sink

Recipients store data in centralized storage such as HBase and HDFS. For example, receiving data from a channel and sending it to a destination. The intended buyer may be another agent or a central store. Example - HDFS pool memo. A Flume agent can have multiple sources, sinks, and channels.

Additional components of flume agent

I mentioned above the importance of names. We also have equipment that plays an important role in transmitting events from data generator to a centralized stores:

1. Interceptors

Interceptors are used to change/monitor sink conditions transmitted between the source and the channel.

2. Channel Selectors

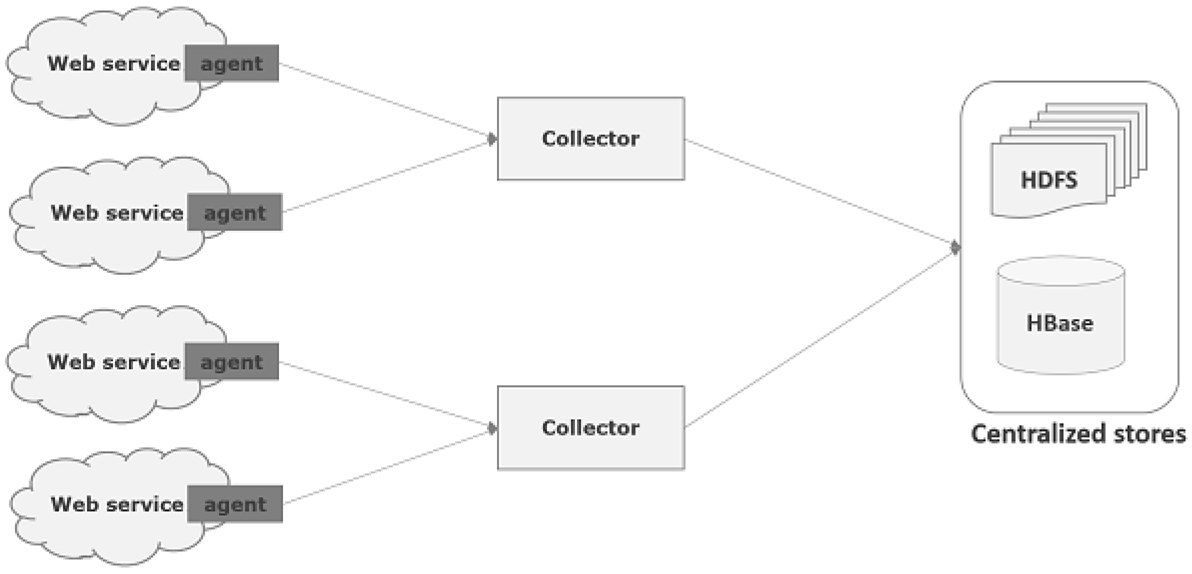
When there are multiple channels, it is used to determine which channel to select for data transmission. There are two types of channel switches: Default channel selector. This is also known as replicating channel selector. Copies all events from each channel. Multiplexing channel selector. Determines which channel to send the event to based on the channel selector in the event header.

3. Sink Processors

Used to call a specific sink in the selected sink group. This is used to create a failover path for a sink or to offload events equally to multiple sinks on a single channel.

2.3.1 Data flow in Apache Flume

Flume is a framework for moving data to HDFS. Typically, events and logs are generated by log servers running Flume agents. These intermediaries receive information from information producers. Data on these workers will be collected by midline called collectors. Just like agents, Flume can have multiple collectors. Finally, the data from all these collectors will be collected and transferred to a storage location such as HBase or HDFS. The diagram below explains the data flow in Flume.



Pic-5. Data flow in Flume

There are several types of data flow in Flume:

1. Multi-hop flow

There can be numerous agents in Flume, and a task can go through many workflows as agents before reaching its final destination. This is called multi-hop flow.

2. Fan-out flow

The flow of information from one source to another is called flow. It comes in two forms: Replication - a data stream in which data will be copied to all systems. Multiplexing - Data will be sent to the selected channel specified in the header of the data stream.

3. Fan-in flow

The data stream in which data is sent from multiple sources to a single channel is called fan-in flow.

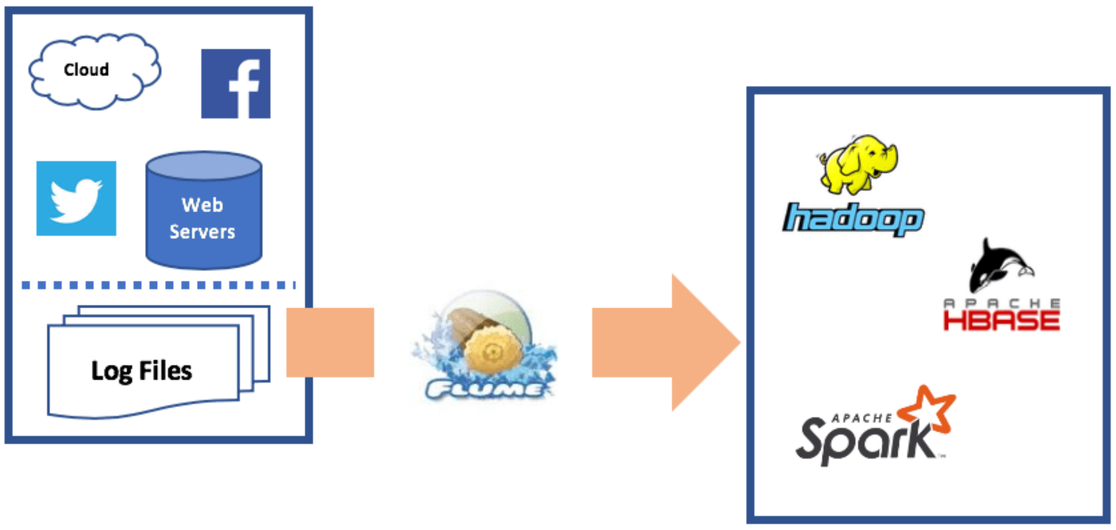
Failure handling

In Flume, two actions occur for each event: one at the sender and one at the receiver. The sender sends the event to the receiver. The receiver sends the transaction shortly after receiving the data and sends the "received" signal to the sender. After receiving the signal, the transmitter sends the changes. (The sender does not perform the conversion until it receives the signal from the receiver.)

The process of transferring data from Apache Flume must be prepared and designed to ensure that the information is transmitted correctly. To stream data from a web server to HDFS, the Flume configuration file must contain information about where the data is received and where it is forwarded. This information is very easy to provide; Flume's source component uses the data engine from the source or data creator and passes it to the data transfer agent. During this process, the information to be transferred is stored in the memory, which then reaches its destination and sinks.

2.4 Why Apache Flume?

Organizations that run multiple websites on multiple servers and hosts generate large amounts of data every day. These records will contain information about events and activities necessary for review and analysis. Their sizes can reach terabytes or even petabytes, and analyzing them requires significant development work and infrastructure costs.
Flume is a popular choice when creating file imports to a database due to its simplicity, flexibility, and functionality, discussed below.



Pic-6. Flume’s Features and Capabilities

Apache Flume extracts logs from various sources and sends them to Hadoop, Apache Hbase and Apache Spark.

Flume provides raw data by extracting data from various sources and sends them to Hadoop data systems. From there, log files can be used by analytics tools like Spark or Kafka. Flume can connect to multiple plugins to ensure log files are delivered to the correct location.

2.5 Integrating Flume with Distributed Databases and Tools

In addition to its ability to stream data from multiple sources to multiple sources, Flume can also be integrated with a variety of tools and products. It can extract data from almost any source, including web servers, log files, csv files created from RDBMS databases, and events. Similarly, Flume can export data to sources such as HDFS, HBase, and Hive.
Flume can be integrated with other data streaming systems such as Kafka and Spark.

2.6 The Limitations of Apache Flume

Apache Flume has some limitations. For starters, when streaming data from multiple sources to multiple sources the architecture can become complex and difficult to manage and control.

Also, Flume's data stream is not 100% real-time. If you need more data streams, you can use other methods like Kafka.
Although Flume can publish duplicate files to the source, identifying duplicate files can be difficult. This match will vary depending on the type of destination to which the data is sent.

Some key features of Flume are: Flume efficiently extracts data from various websites to central storage (HDFS, HBase). Using Flume, we can instantly ingest data from multiple servers into Hadoop. In addition to the dataset, Flume is used to retrieve large amounts of data on activity generated by social networking sites such as Facebook and Twitter, as well as e-commerce sites such as Amazon and Flipkart. Flume supports multiple sites and destinations. Flume supports multi-hop streams, input and output streams, content routing, and more. The channel can be measured horizontally.

3. An Overview of HDFS

HDFS stands for Hadoop Distributed File System. HDFS is a tool developed by Apache to store and process large amounts of data without distributed platform issues. Many databases use Hadoop to scalably accelerate large data sets using multiple computers on the network. Facebook, Yahoo, and LinkedIn are a few companies that rely on Hadoop for profile management.

3.1 Data Transfer In Hadoop

As we all know, big data is one of the big data that cannot be processed using modern calculation methods. Big data can produce great results after analysis. Hadoop is an open source system that allows large data to be stored and processed across computers in a distributed environment using a simple processing model.

Generally speaking, most of the data to be analyzed will be generated from various data sources such as application servers, social networking sites, cloud servers and enterprise servers. This information will be in the list of information and events.

Let me give basic information about log files. Log files are files that record events/transactions occurring in the system. For example, a web server records every request that comes to the server in a log file.
When we collect these logs, we can get information about the operation of the application and find various software and hardware malfunctions.
The general method of transferring data to the HDFS system is to use the put command.

3.2 HDFS put Command

The main challenge in log processing is moving these logs generated from multiple servers to the Hadoop environment.
Hadoop File System Shell Provides commands for adding files to and reading files from Hadoop. You can add data to Hadoop using the given 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 Hadoop commands to transfer data from this source to HDFS. But it has the following disadvantages:
Using the command we can modify the data only once, while the data generator generates the data with higher value. Since analysis of old data is not very accurate, we need a solution that can replace the data on the fly.
If we use the Upload command, the file must be packaged and ready to upload. This is a very difficult task because web servers are constantly generating data. What is needed is a solution that overcomes the shortcomings of the command and transfers the "data stream" from the data generator to central storage (specifically HDFS) with low latency.

3.3 Problem with HDFS

In HDFS, files are included in object names and the length of the file is fixed at zero until closed. For example, if storage for HDFS and network is interrupted during operation (data is not turned off), data written to the database will be lost. So we need a reliable, tunable and controllable method to transfer data to HDFS.
In POSIX documentation, when we access the data (e.g. writes), other services can still read the data (at least the stored information). This is because the data exists on disk before being saved.

3.4 Available Solutions

We can use the following tools to import streaming data (recorded files, events, etc.) into HDFS from various sources:

1. Facebook's Scribe

Scribe is a very popular tool for collecting and sending written documents. It is designed to scale to large numbers of nodes and is resilient to network and node failures.

2. Apache Kafka

Kafka was developed by the Apache Software Foundation. It is an open source messaging service. Using Kafka, we can handle high throughput and low latency broadcasts.

3. Apache Flume

Apache Flume is a tool/service/database used to collect, collect and transform large amounts of streaming data (e.g. logs, events, etc.) from various web servers to a central repository.
It is a reliable and reliable tool that is basically designed to transfer data stream from various sources to HDFS.

4. Best practices of Data Ingestion with Flume and HDFS

When it comes to data ingestion with Flume and HDFS, there are several best practices that can help ensure efficient and reliable data transfer. Here are some key recommendations:

1. Use a reliable and scalable Flume agent configuration: Design your Flume agent configuration to handle the expected data volume and velocity. Consider using multiple sources, channels, and sinks to distribute the load and provide fault tolerance.

2. Configure reliable and fault-tolerant sinks: Choose appropriate sinks that can handle the data ingestion requirements. For HDFS ingestion, use the HDFS sink to write data directly to HDFS. Configure the sink to use a reliable file naming strategy and handle file rotation efficiently.

3. Enable compression and serialization: To optimize data transfer and storage, enable compression and serialization in Flume. This can significantly reduce the size of the data being transferred and stored in HDFS.

4. Set appropriate batch sizes and buffer limits: Configure the batch size and buffer limits in Flume to optimize the data transfer. Adjust these parameters based on the data volume, network bandwidth, and HDFS cluster capacity.

5. Monitor and tune Flume performance: Regularly monitor the performance of your Flume agents and tune them as needed. Use Flume monitoring tools and metrics to identify bottlenecks and optimize the configuration.

6. Secure data transfer: If your data requires secure transfer, configure Flume to use SSL/TLS encryption. This ensures that the data is encrypted during transit.

7. Handle data format and schema changes: If the data format or schema changes over time, ensure that your Flume configuration can handle these changes. Use appropriate serializers and deserializers to handle different data formats.

8. Implement data validation and error handling: Validate the data being ingested to ensure its integrity and quality. Implement error handling mechanisms to handle data ingestion failures and retries.

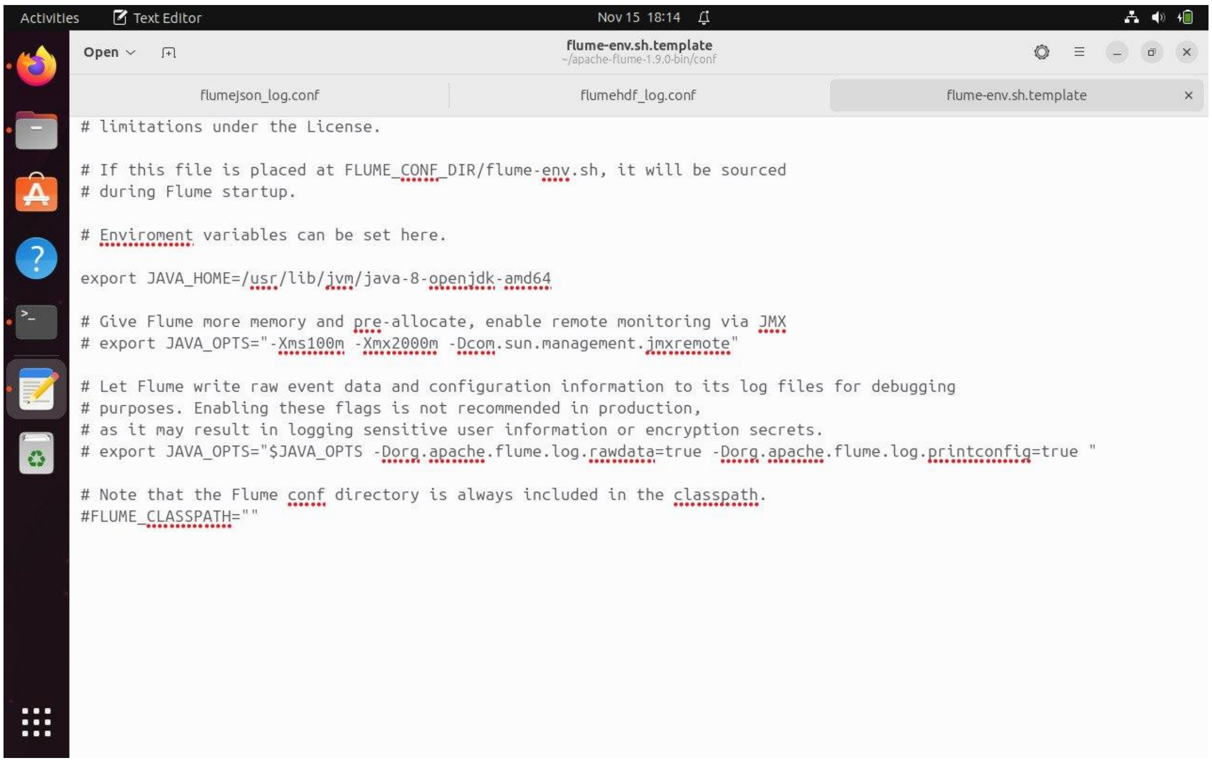
9. Monitor and manage HDFS storage: Regularly monitor the HDFS storage usage and manage it effectively. Implement data retention policies and clean up unnecessary data to optimize storage utilization.

10. Backup and disaster recovery: Implement backup and disaster recovery mechanisms for your Flume agents and HDFS cluster. Regularly back up the Flume agent configurations and HDFS data to ensure data availability and recoverability.

So, to handle fault tolerance and data durability in a Flume-based data ingestion system, we need to use multiple agents, configure channel type, enable replication, implement backoff policies, monitor and alert and test failover scenarios.

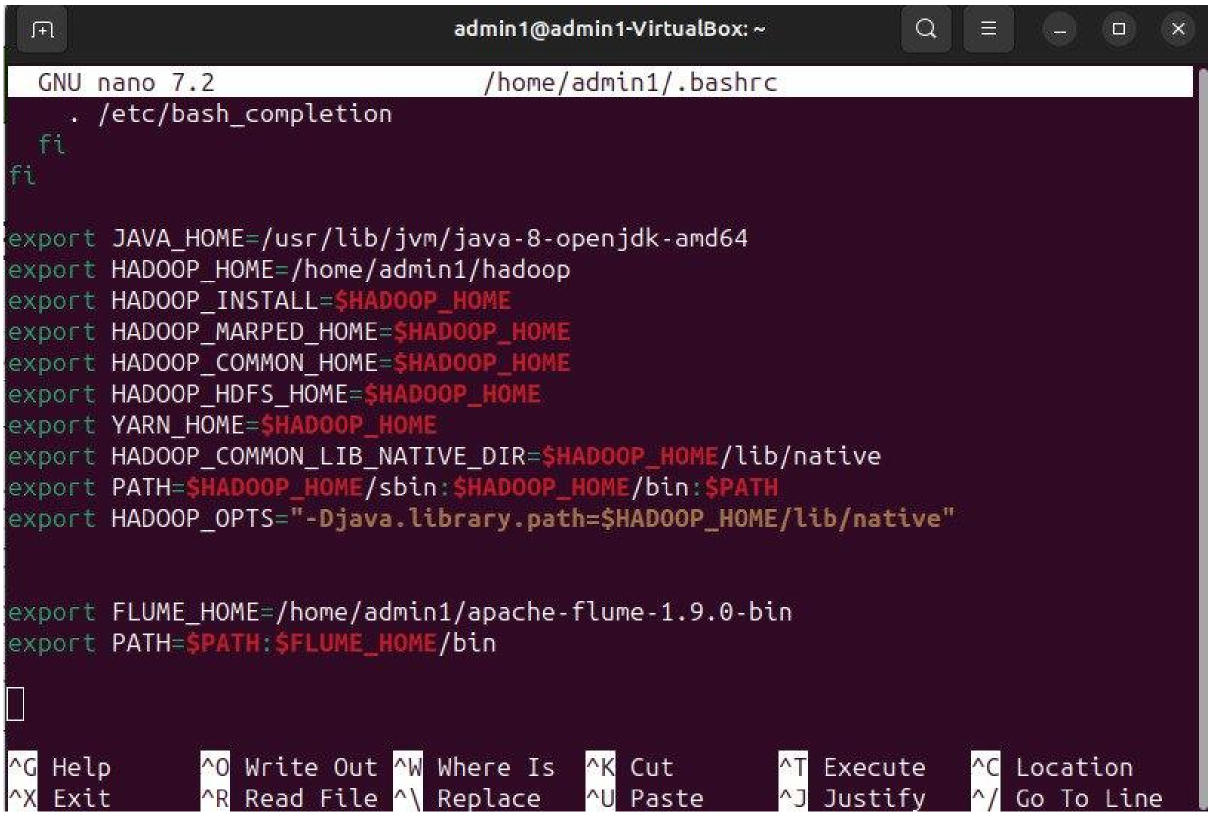
5. Setup

1. I installed the latest version of Apache Flume software from the website "https://flume.apache.org/" to Virtual Box virtual machine running Linux Ubuntu Operating System. And it was saved as apache-flume-1.9.0-bin.tar file in my system.
2. To configure Flume, I modified flume-env.sh, flumeconf.properties, and bash.rc. In the .bashrc file, I set the home folder, the path, and the classpath for Flume. In flume-env.sh file I set the JAVA_Home to the folder where Java was installed in my system.



```
# 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=/usr/lib/jvm/java-8-openjdk-amd64

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

# Let Flume write raw event data and configuration information to its log files for debugging
# purposes. Enabling these flags is not recommended in production,
# as it may result in logging sensitive user information or encryption secrets.
# export JAVA_OPTS="$JAVA_OPTS -Dorg.apache.flume.log.rawdata=true -Dorg.apache.flume.log.printconfig=true "

# Note that the Flume conf directory is always included in the classpath.
#FLUME_CLASSPATH=""
```

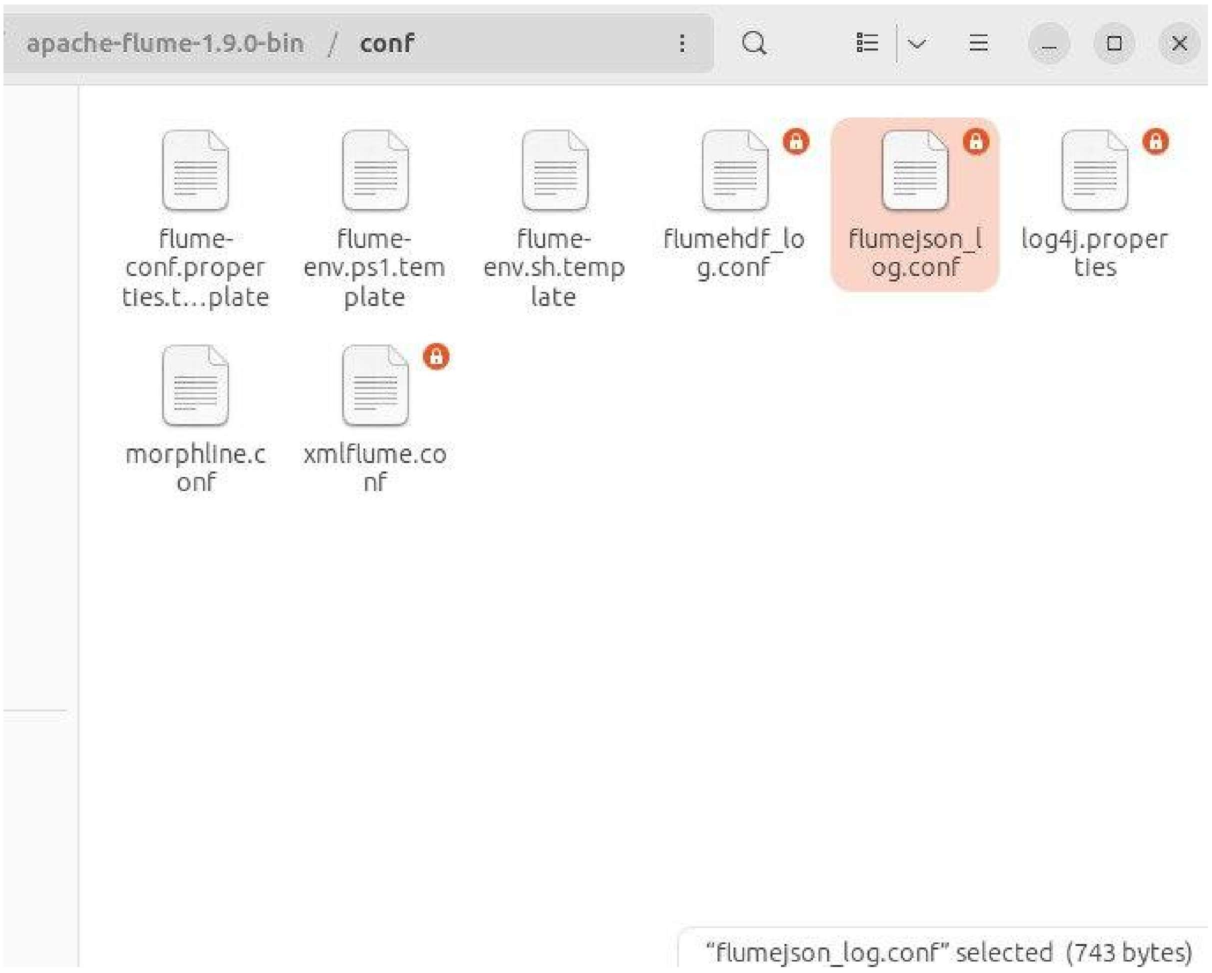
Pic-7. flume-env.sh file after setting JAVA_HOME to the directory of Java installment25



```
admin1@admin1-VirtualBox: ~
  GNU nano 7.2                      /home/admin1/.bashrc
    . /etc/bash_completion
  fi
fi

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/home/admin1/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MARPED_HOME=$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=$HADOOP_HOME/sbin:$HADOOP_HOME/bin:$PATH
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"


export FLUME_HOME=/home/admin1/apache-flume-1.9.0-bin
export PATH=$PATH:$FLUME_HOME/bin


^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```

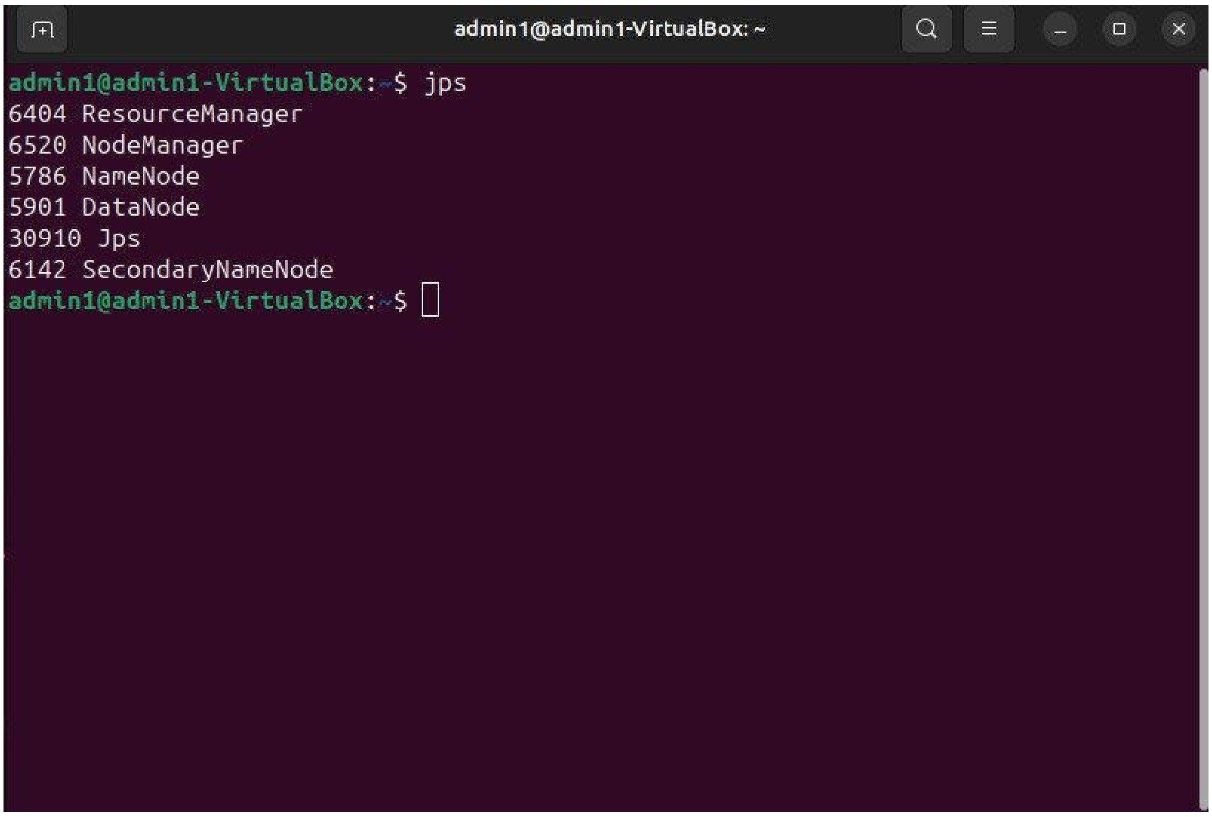
Pic-8. Setting the path in .bashrc file

3. After, I need to list the components such as sources, sinks, and the channels of the agent. To be it more easier I named the source as r1, sink as k1 and channel as c1 in all of my configuration files.
4. I ingested data into HDFS by Apache Flume using three different data resources. The used datasets are json, xml and csv files from "census.gov" open data source.



Pic-9. Conf directory with log files

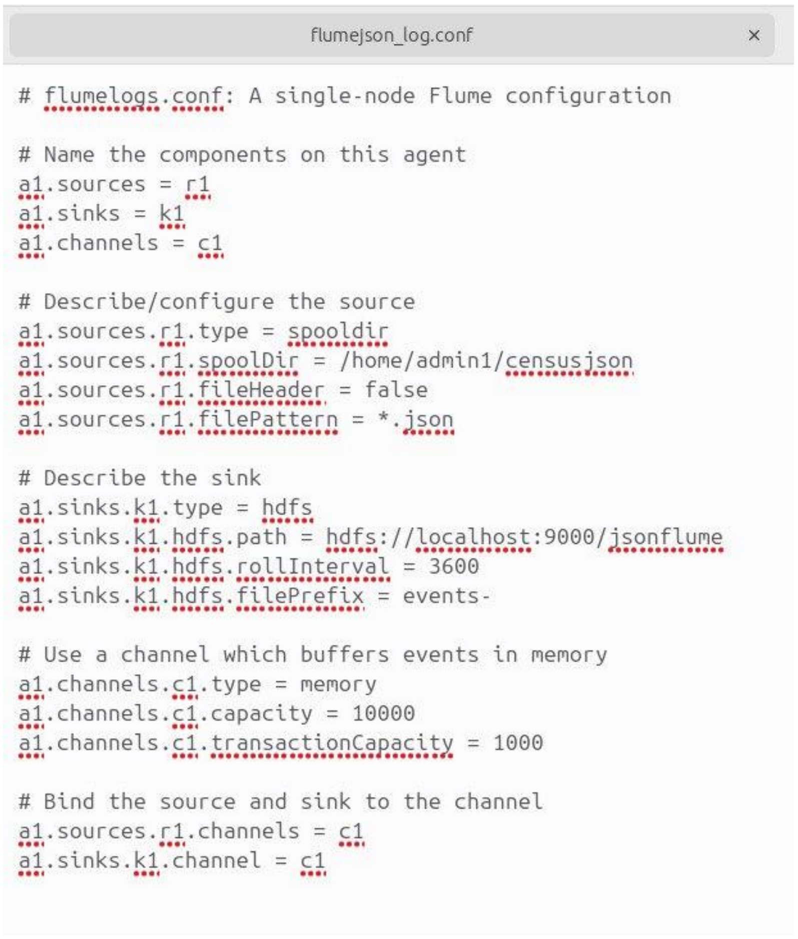
Now I have to specify the properties of the source, channel, and sink. Mostly used property name is "type", and it is used to specify the type of the source we are using.
While specifying the source type, I did it as spooldir, so it shows the directory to the file. Also, I played with fileheader of source, somewhere writing it as true, and in another false. So the original file name can be preserved as a a header if you specify it as true.
Just like the source, each sink will have a separate list of properties. The property named “type” is used to specify the type of the sink we are using. I considered to use HDFS sink.
5. Since we are storing the data in HDFS, we need to install or verify Hadoop. Its already installed, I verified it by $ hadoop version command. Through the sbin directory of Hadoop I started yarn and Hadoop dfs by $ start-dfs.sh and $ start-yarn.sh commands or just using $ start-all.sh command. After I executed jps command to check all Java processes on my machine.



```
admin1@admin1-VirtualBox:~$ jps
6404 ResourceManager
6520 NodeManager
5786 NameNode
5901 DataNode
30910 Jps
6142 SecondaryNameNode
admin1@admin1-VirtualBox:~$
```

Pic-10. Java process status

6. 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. I configured its type as memory. Also, there is capacity and transactionCapacity properties, where capacity is the maximum number of events stored in the channe and transactionCapacity is the maximum number of events the channel accepts or sends.. Their default values are 100. Since the channels connect the sources and sinks, it is required to bind both of them to the channel.
Below is my json configuration file, it is shown that i provided some optional values .

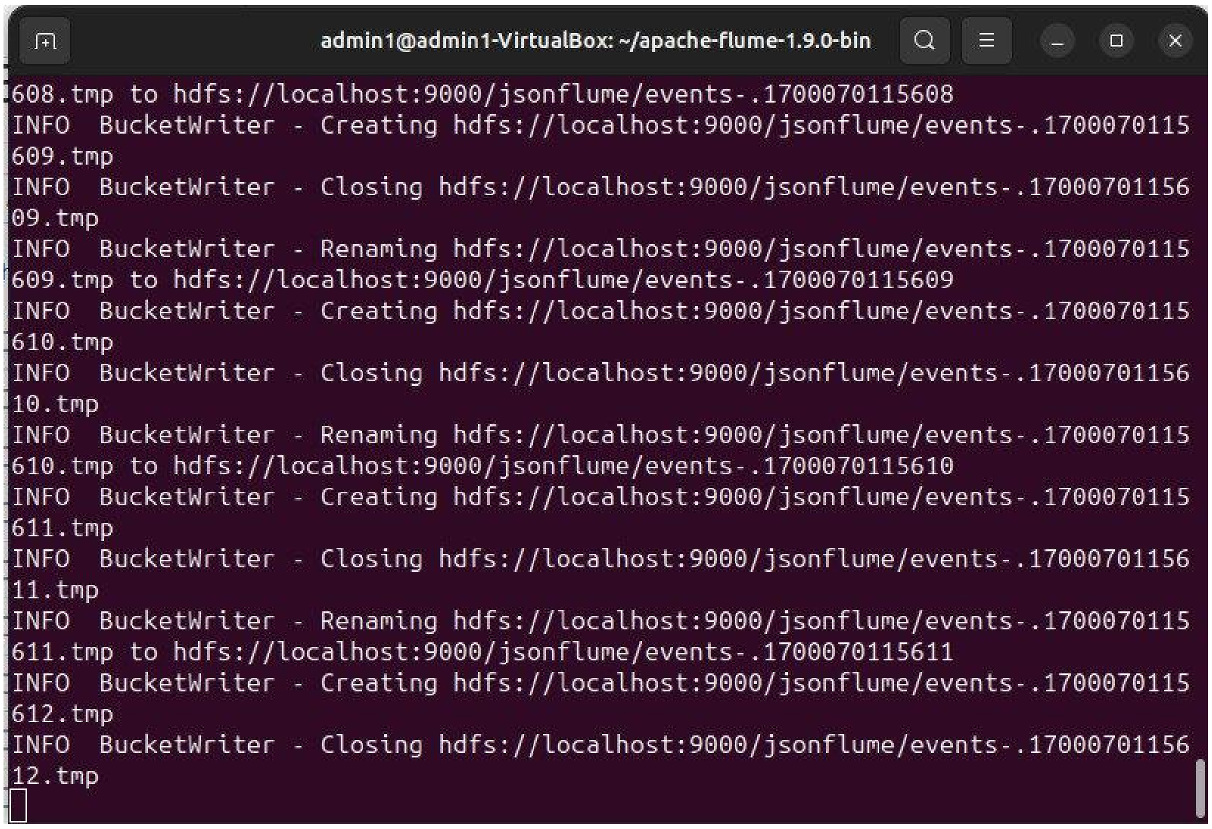

flumejson_log.conf

```
# flumelogs.conf: A single-node Flume configuration

# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = spooldir
a1.sources.r1.spoolDir = /home/admin1/censusjson
a1.sources.r1.fileHeader = false
a1.sources.r1.filePattern = *.json

# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://localhost:9000/jsonflume
a1.sinks.k1.hdfs.rollInterval = 3600
a1.sinks.k1.hdfs.filePrefix = events-

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 10000
a1.channels.c1.transactionCapacity = 1000

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Pic-11. Configuration file

7. After configuration, I have to start the Flume agent. It is done as follows – "bin/flume-ng agent --conf conf --conf-file /home/admin1/apache-flume-1.9.0-bin/conf/flume_logs.conf --name a1 -Dflume.root.logger=INFO, console", 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 agent; -D property =value – Sets a Java system property value.

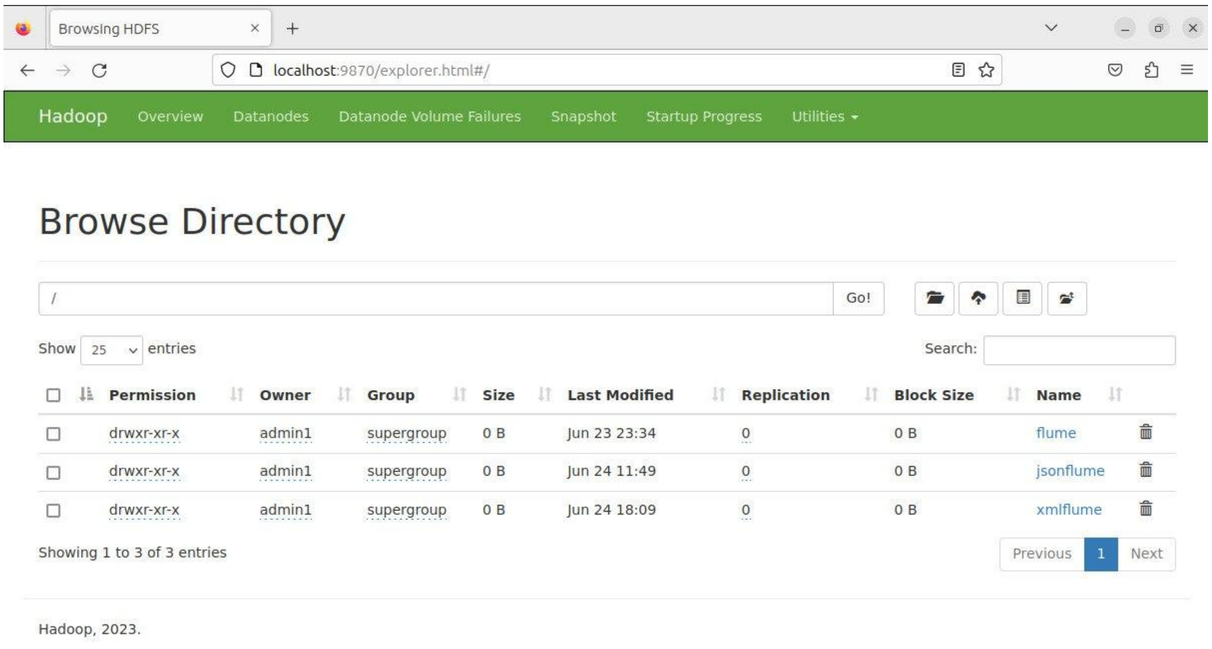
8. After that the source starts generating sequence numbers and sends them to HDFS in a log file.



```
608.tmp to hdfs://localhost:9000/jsonflume/events-.1700070115608
INFO  BucketWriter - Creating hdfs://localhost:9000/jsonflume/events-.1700070115
609.tmp
INFO  BucketWriter - Closing hdfs://localhost:9000/jsonflume/events-.17000701156
09.tmp
INFO  BucketWriter - Renaming hdfs://localhost:9000/jsonflume/events-.1700070115
609.tmp to hdfs://localhost:9000/jsonflume/events-.1700070115609
INFO  BucketWriter - Creating hdfs://localhost:9000/jsonflume/events-.1700070115
610.tmp
INFO  BucketWriter - Closing hdfs://localhost:9000/jsonflume/events-.17000701156
10.tmp
INFO  BucketWriter - Renaming hdfs://localhost:9000/jsonflume/events-.1700070115
610.tmp to hdfs://localhost:9000/jsonflume/events-.1700070115610
INFO  BucketWriter - Creating hdfs://localhost:9000/jsonflume/events-.1700070115
611.tmp
INFO  BucketWriter - Closing hdfs://localhost:9000/jsonflume/events-.17000701156
11.tmp
INFO  BucketWriter - Renaming hdfs://localhost:9000/jsonflume/events-.1700070115
611.tmp to hdfs://localhost:9000/jsonflume/events-.1700070115611
INFO  BucketWriter - Creating hdfs://localhost:9000/jsonflume/events-.1700070115
612.tmp
INFO  BucketWriter - Closing hdfs://localhost:9000/jsonflume/events-.17000701156
12.tmp
```

Pic-12. Command prompt window fetching the data into HDFS

9. I accessed the Hadoop Administration Web UI using the given URL: "http://localhost:9870/", where is my log files stored as given below.



Browse Directory

/

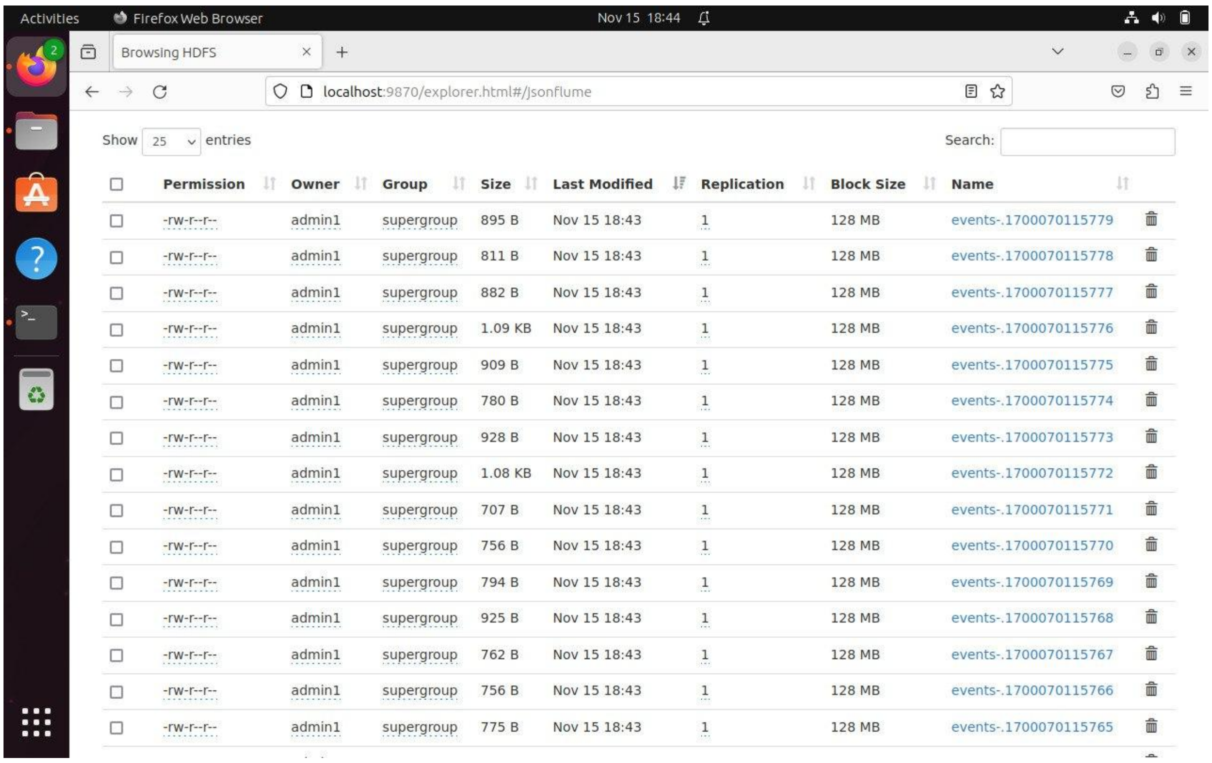
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	admin1	supergroup	0 B	Jun 23 23:34	0	0 B	flume
drwxr-xr-x	admin1	supergroup	0 B	Jun 24 11:49	0	0 B	jsonflume
drwxr-xr-x	admin1	supergroup	0 B	Jun 24 18:09	0	0 B	xmlflume

Showing 1 to 3 of 3 entries

Hadoop, 2023.

Pic-13. HDFS Directory

Then you can see the list of files created by the sequencer and stored in HDFS as shown below.



	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	admin1	supergroup	895 B	Nov 15 18:43	1	128 MB	events-.1700070115779
☐	-rw-r--r--	admin1	supergroup	811 B	Nov 15 18:43	1	128 MB	events-.1700070115778
☐	-rw-r--r--	admin1	supergroup	882 B	Nov 15 18:43	1	128 MB	events-.1700070115777
☐	-rw-r--r--	admin1	supergroup	1.09 KB	Nov 15 18:43	1	128 MB	events-.1700070115776
☐	-rw-r--r--	admin1	supergroup	909 B	Nov 15 18:43	1	128 MB	events-.1700070115775
☐	-rw-r--r--	admin1	supergroup	780 B	Nov 15 18:43	1	128 MB	events-.1700070115774
☐	-rw-r--r--	admin1	supergroup	928 B	Nov 15 18:43	1	128 MB	events-.1700070115773
☐	-rw-r--r--	admin1	supergroup	1.08 KB	Nov 15 18:43	1	128 MB	events-.1700070115772
☐	-rw-r--r--	admin1	supergroup	707 B	Nov 15 18:43	1	128 MB	events-.1700070115771
☐	-rw-r--r--	admin1	supergroup	756 B	Nov 15 18:43	1	128 MB	events-.1700070115770
☐	-rw-r--r--	admin1	supergroup	794 B	Nov 15 18:43	1	128 MB	events-.1700070115769
☐	-rw-r--r--	admin1	supergroup	925 B	Nov 15 18:43	1	128 MB	events-.1700070115768
☐	-rw-r--r--	admin1	supergroup	762 B	Nov 15 18:43	1	128 MB	events-.1700070115767
☐	-rw-r--r--	admin1	supergroup	756 B	Nov 15 18:43	1	128 MB	events-.1700070115766
☐	-rw-r--r--	admin1	supergroup	775 B	Nov 15 18:43	1	128 MB	events-.1700070115765

Pic-14. List of generated log files

As discussed in Flume Architecture, web servers create log files and this data is collected by Flume agents. The channel buffers this information to the receiver, which eventually pushes it to the central storage.

6. Conclusion

By leveraging Flume's flexible architecture and robust source-to-sink data flow model, I was able to seamlessly collect and transport data from diverse sources into HDFS. The integration of Flume agents allowed for easy configuration and management of data ingestion pipelines, enabling the smooth flow of log file data. The scalability of Apache Flume ensured that even with high volumes of data, the ingestion process remained reliable and fault-tolerant. The ability to handle multiple datasets simultaneously further enhanced the efficiency of the ingestion process.

Moreover, the near real-time capabilities of Flume facilitated the timely availability of data for analysis and decision-making purposes. This significantly benefited organizations that rely on up-to-date information to drive their operations. The use of HDFS as the destination for the ingested data provided a robust and scalable storage solution within the Hadoop ecosystem. This allowed for seamless integration with other Hadoop tools and frameworks, enabling advanced analytics and data processing.

In summary, the utilization of Apache Flume for data ingestion with three different datasets into Hadoop, specifically HDFS, has proven to be a successful approach. The combination of Flume's flexibility, scalability, and near real-time capabilities, along with the power of HDFS, sets a solid foundation for organizations to harness the full potential of their data and drive data-driven insights.

7. References

1. https://flume.apache.org/FlumeUserGuide.html
2. https://www.tutorialspoint.com/apache_flume/apache_flume_quick_guide.htm
3. https://www.flume.apache.org
4. https://coderlessons.com/tutorials/bolshie-dannye-i-analitika/izuchite-apache-flume/apache-flume-kratkoe-rukovodstvo
5. https://www.confluent.io/learn/apache-flume/