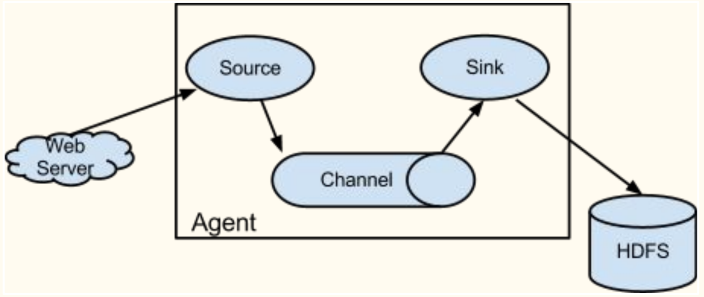
Mastering data ingestion with Apache Flume

Subject: Big Data
Student: Aruzhan Abdirashova
Docente: Antonio Celesti

Contents

Apache Flume Overview

Definition : a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events from various sources to a centralized data store.



Pic-1. The architecture of Flume.

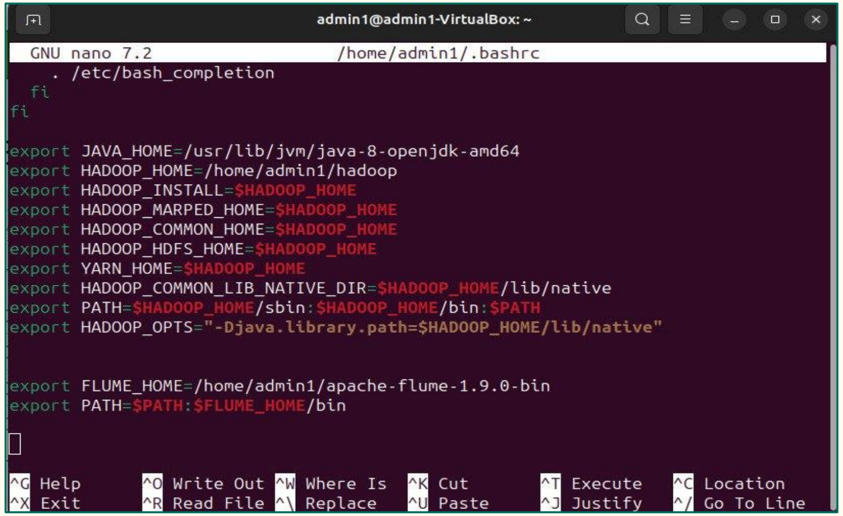
The traditional method of transferring data into the HDFS system is to use the put command. Command: "$ Hadoop fs –put /path of the required file/path in HDFS where to save the file". But the put command of Hadoop suffers from these drawbacks:

1. Using put command, we can transfer only one file at a time while the data generators generate data at a much higher rate.
2. If we use put command, the data is needed to be packaged and should be ready for the upload. Since the web servers generate data continuously, it is a very difficult task.

The 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.

Apache Flume Setup

First of all, 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. 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.



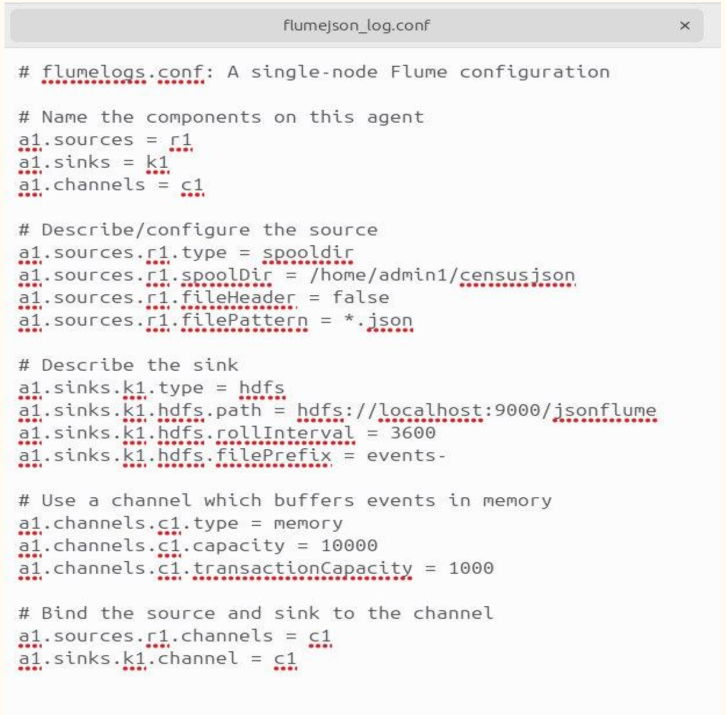
```
 GNU nano 7.2                         /home/admin1/.bashrc
    . /etc/bash_completion
  fi
fi

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-2. Setting the path in .bashrc file.

Flume Agent

In my project, I used only one source, channel and sink. But if you would like to use several agents connected to each other in pipeline we will use one of the formats of Flume called Apache Avro. Avro is a data serialization/deserialization system that uses a compact binary format. The schema is sent as part of the data exchange and is defined using JavaScript Object Notation (JSON).

I listed 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.



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-3. JSON Configuration File.

Agent Source

For every source I used an official website of the United States government named "census.gov". I took three datasets with different formats: csv, json and xml files. In the "type" property i used spooldir and showed the path to the file. I had a chance to use the "exec" type name with xml file, but because I used command "tail" the result came as single event.

Agent Channel

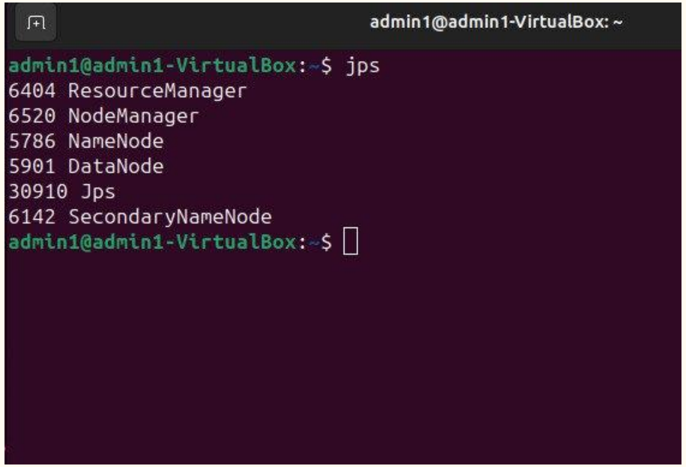
I used the memory channel, stores events in the memory of the Flume agent process. I used capacity property, which determines the maximum number of events that can be stored in the channel. I monitored the memory usage of the Flume agent and adjusted the capacity accordingly. Additionally, there is "transactionCapacity" parameter that specifies the maximum number of events that can be processed in a single transaction. In that case I considered the rate of incoming events and memory capacity of flume agent while the agent was running with command "top" or "free -m".

Agent Sink

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.

In one of the properties "filePrefix", it was set to "-events", which will be prefixed to the files created in HDFS. Also, I used the parameter that determines the time interval after which Flume rolls the current file and creates a new file. It is called "rollInterval".

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

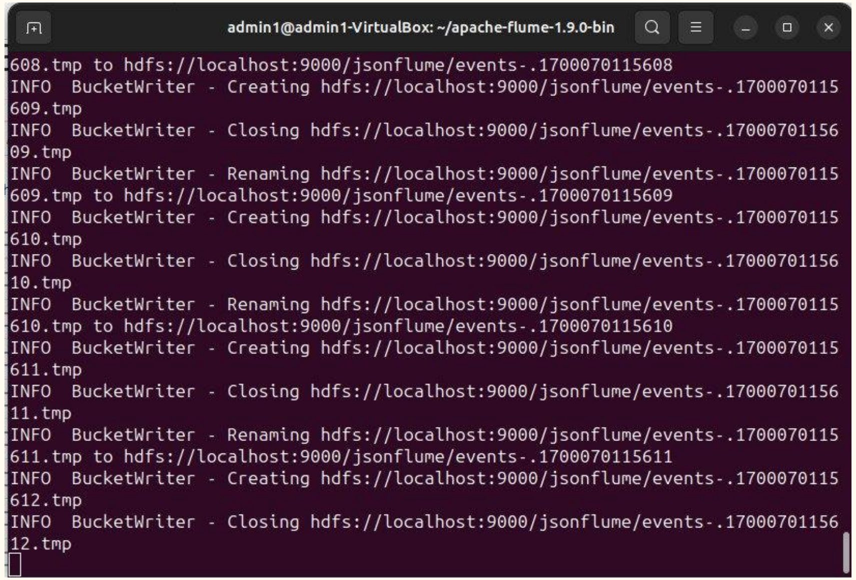
Pic-4. Java process status.

Execution

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.

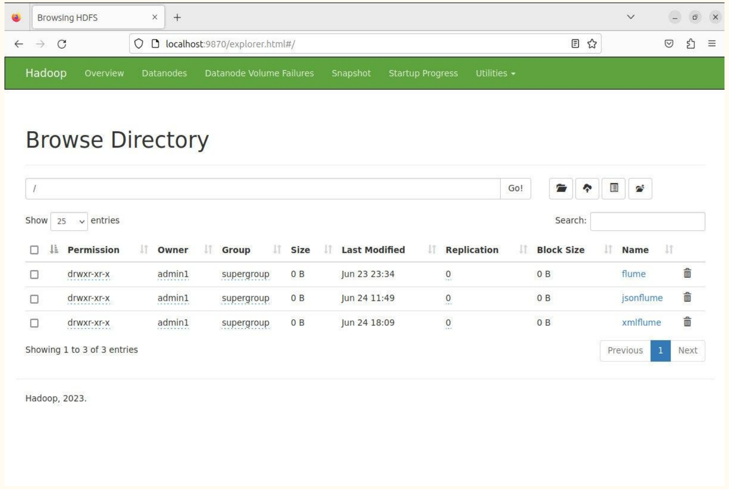


```
608.tmp to hdfs://localhost:9000/jsonflume/events-.1700070115608
INFO  BucketWriter - Creating hdfs://localhost:9000/jsonflume/events-.1700070115
609.tmp
INFO  BucketWriter - Closing hdfs://localhost:9000/jsonflume/events-.17000701156
09.tmp
INFO  BucketWriter - Renaming hdfs://localhost:9000/jsonflume/events-.1700070115
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-5. Command prompt window fetching the data into HDFS.

Verifying HDFS

After that, the source started generating sequence numbers which was pushed into the HDFS in the form of log files. 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

/

Show 25 entries Search:

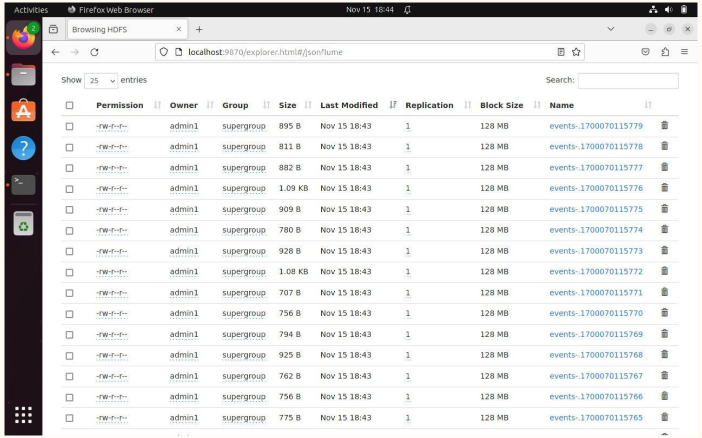
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	admin1	supergroup	0 B	Jun 23 23:34	0	0 B	flume
drwxr-xr-x	admin1	supergroup	0 B	Jun 24 11:49	0	0 B	jsonflume
drwxr-xr-x	admin1	supergroup	0 B	Jun 24 18:09	0	0 B	xmlflume

Showing 1 to 3 of 3 entries

Previous 1 Next

Hadoop, 2023.

Pic-6. HDFS Directory.



Show 25 entries Search:

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-7. List of generated log files.

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.

References

1. https://www.tutorialspoint.com/apache_flume/index.html
2. https://flume.apache.org/FlumeUserGuide.html
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/