Data Science Project

Log Processing from NASA (Space Agency) with Apache Flume and Hbase

Fellipe Augusto Soares Silva

2020

Chapters

Figures

Tables

Summary

This project was developed with the aim of processing, in a Hadoop cluster, log data from requests made to NASA Kennedy Space Center servers located in Florida, United States.

For the project a cluster was built in a Linux environment through a Virtual Machine, that is, through the virtualization of the environment on my physical machine. So, I also performed the entire configuration of the Virtual Machine preparing to receive the Hadoop ecosystem.

To work with the large amount of logs from the NASA server I used Apache Flume. This Hadoop ecosystem framework is a free and reliable log management service for handling, aggregating and collecting large amounts of data by moving this information to Apache Hadoop HDFS according to its configuration specifications.

I developed an Agent that was responsible for collecting, storing and transporting the logs to HDFS, and subsequently improved the way information is delivered to the target platform through the automation of this process.

With Flume working correctly, we entered Apache HBase where we created a table to receive the logs directly from Flume. This part of the project was challenging and at the same time interesting because it showed the power of both tools since from a set of codes it was possible to partition the log file into different columns improving the management and future analysis processes.

With the partition above, through regular expression adjustments in Flume, we were able to insert each request information by IP in different columns in the HBase column family without having to configure the columns in HBase beforehand, making the project more flexible and scalable.

Palavras – Chave: Data Engineering, Data Science, Apache Haddop, Hadoop Ecosystem, NoSQL database, Flume, HBase, NASA, Weblogs, Virtual Machine, Linux, HDFS.

1. Introduction

This project was developed using an Oracle Virtual Machine[1] (VM) where I installed and configured the virtualization of the Linux[2] Operational System from start to finish through the CentOS[3] distribution.

A Virtual Machine is a computational environment that runs on the physical machine but is virtualized and isolated from the real machine. In the VM we can perform tests on different operational systems such as MacOS, Windows, Linux without leaving the base operational system of the machine and can even share items between them.

The choice of CentOS was made because it is a free and robust project regarding the installation of an open - source ecosystem. During the installation a series of configurations were adjusted in order to prepare the virtual environment to receive a Pseudo - Distributed environment from Hadoop Ecosystem[4].

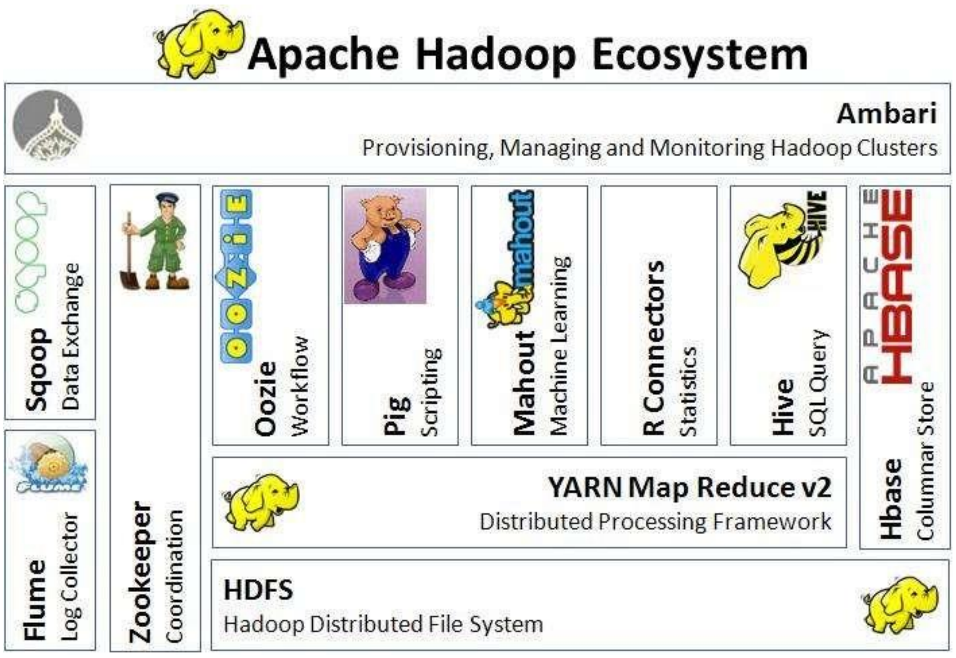


Figure 1 - Hadoop Ecosystem

Hadoop is an open source framework used for distributed storage and parallel processing, it is fault tolerant, scalable, secure and developed for distributed computing in a computer cluster. It is the basis for operations with large data sets, Big Data, as it offers a robust ecosystem

[1] https://www.oracle.com/br/virtualization/virtualbox/
[2] https://en.wikipedia.org/wiki/Linux
[3] https://www.centos.org/
[4] https://hadoop.apache.org/

with different modules that assist in data collection (such as Flume[5] and Sqoop[6]), storage with HDFS (Hadoop Distributed File System), data structuring (with Hive[7] and Hbase[8]), data processing with MapReduce operations using for example Spark[9], and can also perform Machine Learning on large data sets distributed by the cluster with Apache Mahout[10].

Hadoop is an Apache Foundation project, and is mainly composed of three modules:

- Apache HDFS: Responsible for distributed storage in the cluster;
- Hadoop Yarn: Responsible for resource management (memory, CPU ...);
- Hadoop MapReduce: Responsible for parallel processing of large data sets.

An HDFS architecture works with the Master / Worker functions, that is, a Master machine as the master of the entire cluster managing the storage and parallel processing communicating with Workers nodes using a trace of the operations performed.

In a Hadoop environment we have a computer running Master processes (management processes) that are:

- Secondary NameNode: function similar to a backup, although it can assume management functions.
- NameNode: manages HDFS;
- JobTracker: manages MapReduce Jobs.

The other computers in a cluster are the slaves (Slaves / Workers) and it is these processes that do the job itself:

- DataNode: stores and retrieves data from HDFS;
- TaskTracker: performs the Mapping and Reduction work.

DataNode and TaskTracker run on the same machine.

A Client machine, for example my PC running R or Python language, makes a request to the Master computer and this in turn puts the Workers to work either to store or retrieve data from the cluster or perform MapReduce operations.

The Master (or name node) must be the machine within the cluster with the best processing among all. It keeps all your information in memory and to manage all this it has two very important data structures (two files):

[5] https://flume.apache.org/
[6] https://sqoop.apache.org/
[7] https://hive.apache.org/
[8] https://hbase.apache.org/
[9] https://spark.apache.org/
[10] https://mahout.apache.org/

- Fsimage: responsible for storing structural information of the logs such as mapping and namespace of files, in addition to the location of replicas of these files.
- EditLog: responsible for storing all changes to file metadata.

A function as important as distributed storage and parallel processing is replication because in addition to dividing files into blocks, HDFS replicates blocks in an attempt to increase security. By default, HDFS has three (3) replicas allocated on different machines in the cluster (this amount can be configured).

There is still a recommendation for safety and reliability and performance to allocate two (2) replicas in the same rack, but on different machines and the other replica in a different rack.

As physically communication between machines in the same rack is faster than with other racks, for performance reasons when selecting a replica for the process, HDFS gives preference to the replica that belongs to the same rack.

Another benefit with Replication is greater fault tolerance and data reliability, because if a worker machine fails, processing will be done by another machine that contains the replica of that block without the need for data transfer or interrupting application execution.

There are many benefits to working with Apache Hadoop but each of the modules of the Hadoop ecosystem requires a specific configuration that can be found in the official documentation. During the presentation of this project I will highlight the most important configurations I used and for the other configurations I advise you to look for information in the reference material[11].

After installing CentOS with the Linux Operating System, you need to install Hadoop on the Virtual Machine as it forms the basis of the Ecosystem, other products of the Hadoop Ecosystem depend either on HDFS or MapReduce.

When installing CentOS, I created two users:

- root: who is the administrator of the environment
- fellipe: Linux environment user

I could use the fellipe user but for best practices I will build my Hadoop environment on a new user: the hadoop user. In this user I will parameterize my entire parallel and distributed computing environment using HDFS and other ecosystem modules.

[11] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

In practice, Apache Hadoop is installed without an installer, that is, a file is downloaded, unzipped, inserted into a folder on the VM and configured with its environment variables. After that it is ready for use.

The files needed for configuration are:

- .bashrc
- core-site-xml
- hdfs-site.xml

As stated earlier, the items to be configured within these files are in the official product documentation[12].

Another important module for the operation of HDFS is YARN[13]. This module is responsible for managing jobs within HDFS and must be started together with HDFS so that both the storage and processing of data distributed by the cluster can take place. The files needed for configuration are:

- mapred-site.xml
- yarn-site.xml

To initialize both HDFS and YARN after configurations done, we must type the following commands in the terminal:

- start-dfs.sh
- start-yarn.sh

As a result we have to have the following services running on the OS:

- NameNode
- DataNode
- SecondaryNameNode
- NodeManager
- ResourceManager

Continuing to install the Hadoop ecosystem,:

- Zookeeper: It aims to provide a coordination service for high-performance distributed applications that provides the means to facilitate the tasks of configuring a machine in the cluster, synchronizing distributed processes, and groups of services.

[12] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html
[13] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

- HBase: It is a distributed and scalable database, being a NoSQL database, that is, it allows you to store data without worrying about the schema.
- Hive: Its main functionality is to provide an infrastructure that allows the use of SQL language, in fact a modification of SQL that is HiveQL and allows to build a kind of data warehouse under Apache Hadoop. If the goal is to work with structured data and execute almost SQL queries, Apache Hive is an excellent alternative.
- Pig: It is a high level language oriented to data flow and execution for parallel computing. The optimization of Apache Pig does not change the configuration of the Hadoop cluster because it is used in client mode providing a language called Pig Latin and a controller capable of transforming programs of the Pig type into sequences of the MapReduce programming model. What it does is convert the Pig Latin code to JAVA or Scala format (Hadoop programming languages) in order to work on Jobs without much difficulty.
- Spark: Processing of mapreduce jobs within the cluster
- Sqoop: Data transfer tool for Apache Hadoop that can put data from a database into HDFS, Hive or HBase, or do the opposite way of exporting data from this distributed platform back to the relational database.
- Flume: Allows me to bring data from the most varied sources to Apache Hadoop, including log data (when running any application it generates output - information of who accessed the application, when it accessed, what resources it used, if there was memory overflow and so on)
- Mahout: Hadoop Ecosystem product to work with Machine Learning. Simplifies (as Pig does) the Machine Learning process on distributed data.

We have three modes of execution of Haddop where each item of the ecosystem must be configured according to the chosen mode:

- Standalone: In this mode, hadoop is configured to run in local mode. This mode is most recommended for the development phase, which is when most errors normally occur, being necessary to carry out several tests of the execution of the application of my data analysis.

- Pseudo - Distributed: In this mode, all the configurations that are necessary for the execution of a cluster are applied, however, the whole application is processed in Local Mode. Although it is not actually executed in parallel, this mode allows its simulation as it uses all the processes of an effective parallel execution: NameNode, DataNode, JobTracker, TaskTracker and SecondaryNameNode.
- Fully Distributed: Mode used for distributed processing of the Hadoop application on a computer cluster. In this option it is also necessary to edit the XML files previously presented, defining specific parameters and the location of SecondaryNameNode and Nodes Workers. However, as we have several computers in this mode, we must indicate which machines will actually run each component.

Each module was configured following the Hadoop Pseudo - Distributed execution mode so that we would be getting as close as possible to a real environment.

The beginning of the solution to any business problem lies in the understanding of the problem itself, which will be presented in the next chapter.

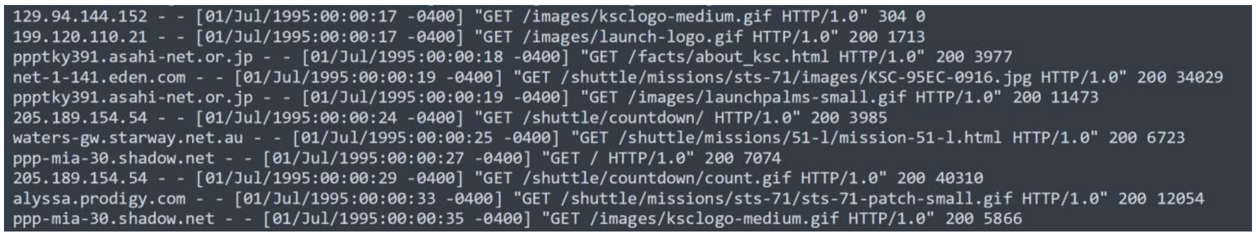
2. Business Problem and Project Development

As described in the title of the project, we will develop the processing of logs from NASA (North American Space Agency) server with Flume and Hbase. To fully understand the problem we will divide the concepts here involved.

Logs, in computing, are relevant records of events that occur in an operating system, for example, accesses on a server, error reporting, file requests, processing, memory usage, record of updates, in short, there is a lot of information that can be stored in a log file. Most of these files are used to let the system administrator know what is happening and how to act.

In this project we will use logs from requests made to NASA Kennedy Space Center[14] servers located in Florida, United States.

Our file corresponds to requests for two months of access and operations performed by users around the world on this NASA server. The information storage structure is as follows:



```
129.94.144.152 - - [01/Jul/1995:00:00:17 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:17 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1713
ppptky391.asahi-net.or.jp - - [01/Jul/1995:00:00:18 -0400] "GET /facts/about_ksc.html HTTP/1.0" 200 3977
net-1-141.eden.com - - [01/Jul/1995:00:00:19 -0400] "GET /shuttle/missions/sts-71/images/KSC-95EC-0916.jpg HTTP/1.0" 200 34029
ppptky391.asahi-net.or.jp - - [01/Jul/1995:00:00:19 -0400] "GET /images/launchpalms-small.gif HTTP/1.0" 200 11473
205.189.154.54 - - [01/Jul/1995:00:00:24 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
waters-gw.starway.net.au - - [01/Jul/1995:00:00:25 -0400] "GET /shuttle/missions/51-l/mission-51-l.html HTTP/1.0" 200 6723
ppp-mia-30.shadow.net - - [01/Jul/1995:00:00:27 -0400] "GET / HTTP/1.0" 200 7074
205.189.154.54 - - [01/Jul/1995:00:00:29 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
alyssa.prodigy.com - - [01/Jul/1995:00:00:33 -0400] "GET /shuttle/missions/sts-71/sts-71-patch-small.gif HTTP/1.0" 200 12054
ppp-mia-30.shadow.net - - [01/Jul/1995:00:00:35 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 200 5866
```

Figure 2 - NASA Server Log

Among the log information above we have the following order:

- Host: where the connection is coming from. Host name or IP address
- Timestamp: access time in the format day/month/year:hour:min:sec -timezone
- Request: request made by the host
- HTTP return code
- Bytes used in the return

These are the five pieces of information that the log file has for each request to the NASA server. In total we have more than 3.5 million rows with log data. Certainly a Big Data problem, so we will use Apache Hadoop for storage and distributed processing.

[14] https://www.kennedyspacecenter.com/

Just in terms of curiosity, I will make a request, that is, I will make an access to the NASA server and request an image of one of the space missions carried out by the Agency. The chosen space mission was the STS - 71, which was the third mission of the Atlantis space shuttle program carried out between June 27th and July 7th in 1995[15].

To perform the requisition, I entered the space mission website STS - 71, searched for the files stored on the server and requested the following file: "/shuttle/missions/sts-71/images/high/KSC-95EC-0916.jpg" which returned the following image of the launch of the space shuttle Atlantis at an incredible angle:

Figure 3 - NASA Server Request (STS Mission - 71 - Atlantis)

At this time NASA's server stored my IP, the location and time of access, the information I requested from the server (which in our case is the figure above) and the number of bytes used in the request.

[15] https://www.nasa.gov/mission_pages/shuttle/shuttlemissions/archives/sts-71.html

2.1Apache Flume

Another concept involved is the use of a framework of the Hadoop ecosystem, Apache Flume[16]. Flume is a log management service that is free and reliable for handling, aggregating and collecting large amounts of data by moving this information to Apache Hadoop HDFS according to its configuration specifications.

In its official documentation we can see how Flume's data flow architecture works, as follows:

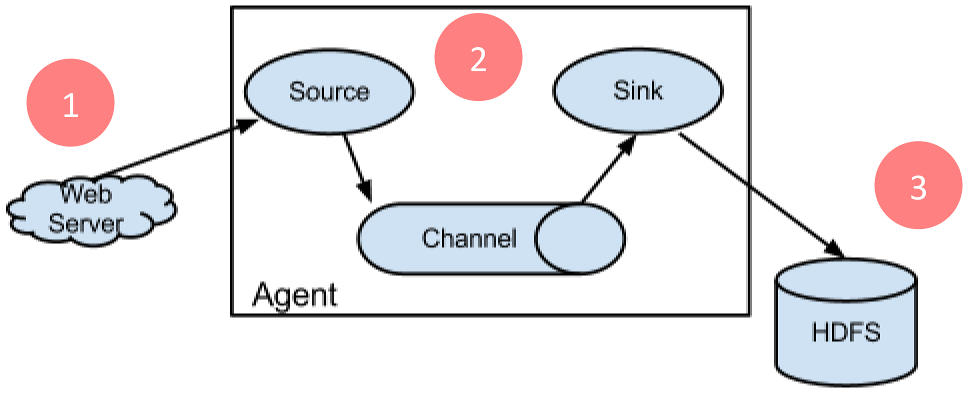


Figure 4 - Flume Architecture

As you can see in the figure above, we have three basic information:

- 1 – Web Server: here is where we will connect to be able to collect the logs
- 2 – Agent: the agent is responsible for all the functionality of Flume.
- 3 – HDFS: our hadoop cluster. Storage location of collected logs.

Following figure 4 we know that item 1 will be the NASA server logs and item 3 will be Hadoop Distributed File System (HDFS) already configured on the Virtual Machine, what we haven't seen yet is item 2, the Agent.

The Agent is a configuration file that tells Flume how to ingest data and transfer it to HDFS. An application with Flume can have one or more Agent and each Agent has the following items that communicate with each other:

- Source: configuration responsible for connecting to the web server and acquiring the logs
- Channel: it acts as a storage queue for the collected logs, releasing it to the sink only when it requests it, otherwise it is stored waiting for the request.

[16] https://flume.apache.org/

- Sink: processes the data released by the channel and delivers it to the destination, HDFS.

An application with Flume can have several Agents connecting to each other until the end of the process delivering to HDFS what was requested, however it is essential to know that regardless of the number of Agents in the application, each Agent must contain a Source, a Channel and a Sink configured.

You may be asking yourself: why have a “channel” in the middle of the path preventing you from taking the file directly to the sink and consequently to HDFS? The answer to this is in the operation because if there is any type of failure in communication with HDFS the channel keeps the log file preserved until connection is reestablished.

We will return to Apache Flume during the configuration of our Agent, but first we will study the last concept: Apache HBase.

2.2Apache HBase

HBase is a non - relational database (NoSQL) distributed and column oriented. As it is a database that is easy to integrate with Hadoop, it is suitable for data processing on a terabyte scale, that is, HBase was not developed for 100 thousand lines of data, but for more than 1 million pieces of information.

With HBase, I do not need to be defining tables within the database since HBase technically acts as a DataStore and not as a Database, thus taking advantage of the storage advantages of HDFS and later of the parallel processing of our Hadoop cluster.

HBase has as main characteristics:

- Horizontal Scalability: it is possible to increase the number of machines that run in parallel.
- Consistent read / write processes (Hbase Read / Hbase Write): It has the advantage of working with HDFS, that is, I will write large volumes of data and make specific readings on the data and not on the entire database (relational database characteristic), here the goal is to save zillions of rows and then run a query that will fetch a few million rows.
- Automatic partitioning: Working with very large tables requires partitioning, because query is performed on parts of the data.
- Automatic disaster recovery: Same as the Hadoop cluster, if one machine fails, the others are still in operation.
- Java API for data access: connections via JDBC are more practical.

If you need to have all the features listed above, HBase can be a great option for your project, but understand that if the goal is to store few lines, or your cluster has few machines then it may not be the ideal database for your work.

Furthermore, the main feature of HBase is that it is a column-oriented database, but what does that mean? Observe the following figure:

Row Key	**Employee**			**Job**	
ID	**FirstName**	**LastName**	**Age**	**e-mail**	**Occupation**
1	Crystal	Smith	27	smith012@employer.com	Sales
2	Jack	Adams	33	jack023@employer.com	Manager
3	Rose	Abel	31	rose033@employer.com	Human Resources

Table 1 - Column Oriented HBase

In the table above I have a COLUMN called “Row Key” that corresponds to the key for each of the lines; next I have a FAMILY OF COLUMNS called "Employee" and another "Job" where the first has three COLUMNS ("FirstName, LastName and Age") and the second has 2 COLUMNS ("e-mail and Ocuppation").

These columns represent the data we want to process, but why divide the columns into families? Here comes the advantage of Hbase: I can place each family on a different server, I can partition a family, I can place partitions of each family within the same server to speed up consultation, recording, searching, etc., that is, I can make arrangements with the tables which would not be possible with relational databases.

We will use HBase in this project because first we will work with few tables and few columns, but with many rows, according to which we will use all the power offered by HDFS since HBase runs on it storing the data in a distributed way on several servers.

2.3 Configuring an Agent

As previously stated, the Agent is responsible for carrying out the entire collection and transport flow. For this flow to happen, it is necessary to configure the source, channel and sink, indicating how each item will interact with the external environment and between themselves.

Initially it is necessary to create a file in the Virtual Machine with the extension “.conf” so that Flume reads this file when it is triggered.

In my case I chose to name it as flume-agent.conf and entered the following configuration:

```
# Identify the components on agent a1:
a1.sources = sr1
a1.channels = ch1
a1.sinks = sk1

# Configure the source:
a1.sources.sr1.type = netcat
a1.sources.sr1.bind = 127.0.0.1
a1.sources.sr1.port = 11111

# Configure a channel that buffers events in memory:
a1.channels.ch1.type = memory
a1.channels.ch1.capacity = 10000
a1.channels.ch1.transactionCapacity = 100

# Describe the sink:
a1.sinks.sk1.type = hdfs
a1.sinks.sk1.hdfs.path = hdfs://localhost:9000/user/flume/a1
a1.sinks.sk1.hdfs.rollInterval = 30
a1.sinks.sk1.hdfs.writeFormat = Text
a1.sinks.sk1.hdfs.fileType = DataStream

# Bind the source and sink to the channel:
a1.sources.sr1.channels = ch1
a1.sinks.sk1.channel = ch1
```

With this file we created an Agent called “a1”. In the first code block we are determining the names of each agent item (sr1 for the source; ch1 for the channel; sk1 for the sink). This information will be used in all other blocks according to the necessary configuration.

In the second block of code we are configuring the source. Notice that we have three items in this block, namely:

- type: the name of the component that will be used to connect to the information source, that is, with the logs. I used netcat but it could be a multitude of other connections.
- bind: host name or IP address to connect
- port: port number to connect

What this source does is connect to a specific port and acquire the data that is inserted there, transforming each line of information into an event to be sent to the channel.

In the third block of code we have the channel configuration. The items in this block are:

- type: I used the memory configuration where events from the source will be stored in a queue in the machine's memory.
- capacity: maximum number of events that can be stored on the channel
- transactionCapacity: maximum number of events per transaction that the channel can receive from the source and also transmit to the sink.

Note that these settings will depend a lot on the business problem and the size of the file to be collected and transmitted to HDFS. For this project the numbers entered were sufficient.

The fourth block of code refers to the sink configuration, namely:

- type: as this business problem aims to insert log data in HDFS, here we have to perform all the configurations based on this framework.
- hdfs.path: directory within HDFS to which events will be inserted.
- hdfs.rollInterval: number of seconds to wait to close the file in use and create a new file, inserting new events in this new file.
- hdfs.writeFormat: file recording format.
- hdfs.fileType: the DataStream format will prevent the output file from being compressed. Depending on the business problem, you could compress the file to formats such as gzip, bzip2, among others.

What this sink is configured to do is insert the events collected by the source and stored by the channel, in HDFS. As configured, when an event is received by the sink, it leaves the file open for 30 seconds and starts receiving and recording events stored in the channel, and as soon as it completes 30 seconds, or reaches the limit of 100 transactions, the sink closes the file, transmits to HDFS and opens a new file receiving new events.

The fifth and final block of codes is telling Flume how to connect all the previous instructions, that is, the source connects to the channel via sr1 - ch1 and the channel connects to the sink via ch1 - sk1.

After inserting the code blocks, save the file and open a new terminal to perform a test of agent a1.

Before performing the test it is necessary to start the cluster:

- start-dfs.sh
- start-yarn.sh

With the services running on the Virtual Machine, type in the terminal:

flume-ng agent -n a1 -f /home/hadoop/flumeag/flume-agent.conf

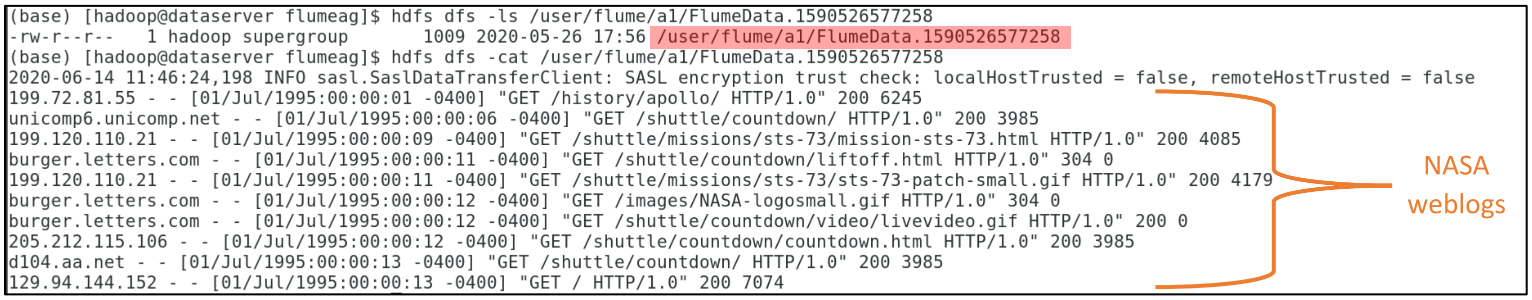
- flume-ng: command to initialize Flume
- agent: command to indicate that we are going to open an agent
- -n a1: command to open the agent we created, agent a1
- -f /home/hadoop/flumeag/flume-agent.conf: location of the agent configuration file that we created.

In this way we start agent a1, which is waiting for logs on the indicated port of localhost 11111. To deliver log files to the agent, use the following command in a new terminal:

head -n 10 /home/hadoop/flumeag/access_log_Jul95 | nc localhost 11111

This command will use nc (which is netcat) to connect to localhost 11111 and send the first 10 lines (head -n 10) from the file access_log_Jul95, the file with NASA server logs.

After this operation we can enter HDFS in the folder specified in the sink (/ user / flume / a1) and watch our file loaded successfully, as shown in the following figure.



```
(base) [hadoop@dataserver flumeag]$ hdfs dfs -ls /user/flume/a1/FlumeData.1590526577258
-rw-r--r--   1 hadoop supergroup       1009 2020-05-26 17:56 /user/flume/a1/FlumeData.1590526577258
(base) [hadoop@dataserver flumeag]$ hdfs dfs -cat /user/flume/a1/FlumeData.1590526577258
2020-06-14 11:46:24,198 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0" 200 3985
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
```

Figure 5 - NASA logs in HDFS I

As you can see in the figure above, we have the first 10 lines of our NASA log file. See the indication in red that the file was saved with the format determined by Flume, that is, as there was no rule determined by me, it was inserted according to default settings.

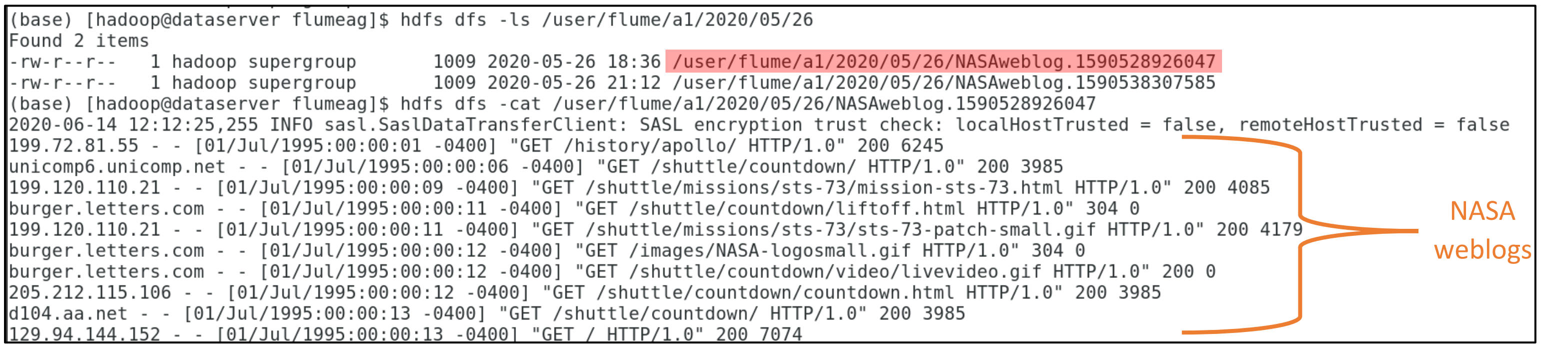
We can improve storage on HDFS right in the sink configuration. I will make an improvement by automatically inserting the date on which we transport the collected events. To do this, insert the following instructions in the fourth block of code:

```
a1.sinks.sk1.hdfs.path = hdfs://localhost:9000/user/flume/a1/%Y/%m/%d
a1.sinks.sk1.hdfs.useLocalTimeStamp = true
a1.sinks.sk1.hdfs.filePrefix = NASAweblog
```

Where:

- hdfs.path: files saved to HDFS will follow the Year/month/date pattern
- hdfs.useLocalTimeStamp: indication to Flume to use the local time in order to fill in the hdfs.path above
- hdfs.filePrefix: in this line I am determining how the prefix of the file saved with NASA's server logs should be. What used to be FlumeData as default will now be NASAweblog.

Re-enter the command at the terminal to load log files and enter HDFS by checking the transport:



```
(base) [hadoop@dataserver flumeag]$ hdfs dfs -ls /user/flume/a1/2020/05/26
Found 2 items
-rw-r--r--   1 hadoop supergroup       1009 2020-05-26 18:36 /user/flume/a1/2020/05/26/NASAweblog.1590528926047
-rw-r--r--   1 hadoop supergroup       1009 2020-05-26 21:12 /user/flume/a1/2020/05/26/NASAweblog.1590538307585
(base) [hadoop@dataserver flumeag]$ hdfs dfs -cat /user/flume/a1/2020/05/26/NASAweblog.1590528926047
2020-06-14 12:12:25,255 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0" 200 3985
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
```

Figure 6 - NASA logs in HDFS II

Now we have a more organized and personalized job according to the desired specifications. See that we transport the same 10 rows of logs from NASA's server, but saving them within the path -> / user / flume / a1 / 2020/05/26 / so we facilitate the use of data by analysts and Data Scientists by listing the transport of logs by date.

With this we performed a complete test of Flume indicating its good functioning allowing us to get into the business problem by moving log data from NASA's server directly to HBase using Flume.

2.4 Apache Flume and HBase

We already have Apache Flume working perfectly and now returning to the business problem we are going to unite its functionality with HBase. For everything to work correctly, Flume's only requirement is that HBase version is lower than HBase2. HBase 1.6 was used in this project.

From the agent a1 previously configured we will adjust the sink so that the flume instead of transferring the log events to HDFS, it transfers to an HBase table that is running on HDFS.

```
# Describe the sink:
a1.sinks.sk1.type = hbase
a1.sinks.sk1.table = NASA
a1.sinks.sk1.columnFamily = logs
a1.sinks.sk1.serializer = org.apache.flume.sink.hbase.RegexHbaseEventSerializer
```

Where:

- type: we switched from hdfs to hbase, so the sink is redirected to this framework.
- table: table name inside HBase to write data collected from NASA server logs
- columnFamily: column family of the table indicated in the previous field where data should be inserted
- serializer: this is the way hbase will handle input information. In this case, data will be recorded with each row being an information in the column family. We will improve this information later.

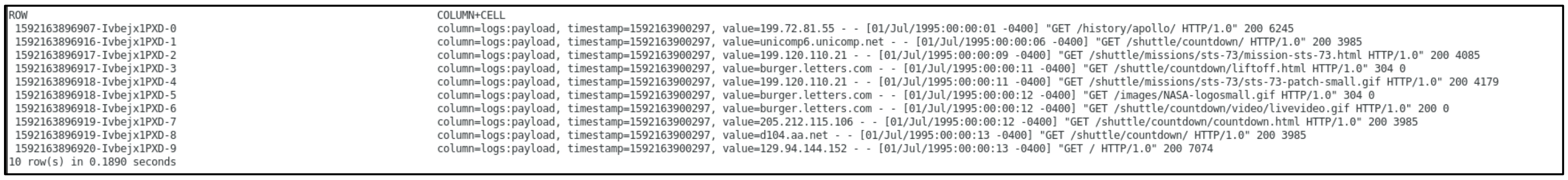
With the adjustments made to flume-agent-hbase.conf, we will enter HBase and create our "NASA" table with the "logs" column family.

Start the HBase service: start-hbase.sh, type hbase shell in a new terminal to open the command line and create the table with the commands: create 'NASA', {NAME => 'logs'}

With that we can activate agent a1 and transport the first 10 lines of NASA's log file again:

- flume-ng agent -n a1 -f /home/hadoop/flumeag/flume-agent-hbase.conf
- head -n 10 /home/hadoop/flumeag/access_log_Jul95 | nc localhost 11111

Getting as a result in our HBase table:

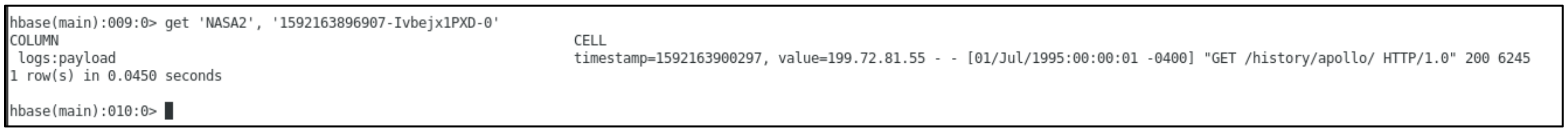


```
ROW                                                 COLUMN+CELL
 1592163896907-Ivbejx1PXD-0                         column=logs:payload, timestamp=1592163900297, value=199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
 1592163896916-Ivbejx1PXD-1                         column=logs:payload, timestamp=1592163900297, value=unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
 1592163896917-Ivbejx1PXD-2                         column=logs:payload, timestamp=1592163900297, value=199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
 1592163896917-Ivbejx1PXD-3                         column=logs:payload, timestamp=1592163900297, value=burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
 1592163896918-Ivbejx1PXD-4                         column=logs:payload, timestamp=1592163900297, value=199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
 1592163896918-Ivbejx1PXD-5                         column=logs:payload, timestamp=1592163900297, value=burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
 1592163896918-Ivbejx1PXD-6                         column=logs:payload, timestamp=1592163900297, value=burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
 1592163896919-Ivbejx1PXD-7                         column=logs:payload, timestamp=1592163900297, value=205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0" 200 3985
 1592163896919-Ivbejx1PXD-8                         column=logs:payload, timestamp=1592163900297, value=d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
 1592163896920-Ivbejx1PXD-9                         column=logs:payload, timestamp=1592163900297, value=129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
10 row(s) in 0.1890 seconds
```

Figure 7 - NASA logs in HBase

As shown above, we upload the first 10 log events collected directly to HBase through Flume, inserting each event in the "logs" column family.

Here would be the end of the project because we have successfully transported the logs, but I will improve the way HBase stores events. Although we have several information for each event, unfortunately they are consolidated in only one column within the column family, notice how the query command returns:



```
hbase(main):009:0> get 'NASA2', '1592163896907-Ivbejx1PXD-0'
COLUMN                                                      CELL
 logs:payload                                               timestamp=1592163900297, value=199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
1 row(s) in 0.0450 seconds

hbase(main):010:0> █
```

Figure 8 - Querying logs in HBase

The “logs” family column was loaded with the entire event, without partition:

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245

What I'm going to do is improve the agent a1 sink configuration file a little more so that when transporting the event to HBase it separates this file into IP columns (being my ROW_KEY), TIME_ACCESS, REQUEST, HTTP_CODE1, HTTP_CODE2, BYTES.

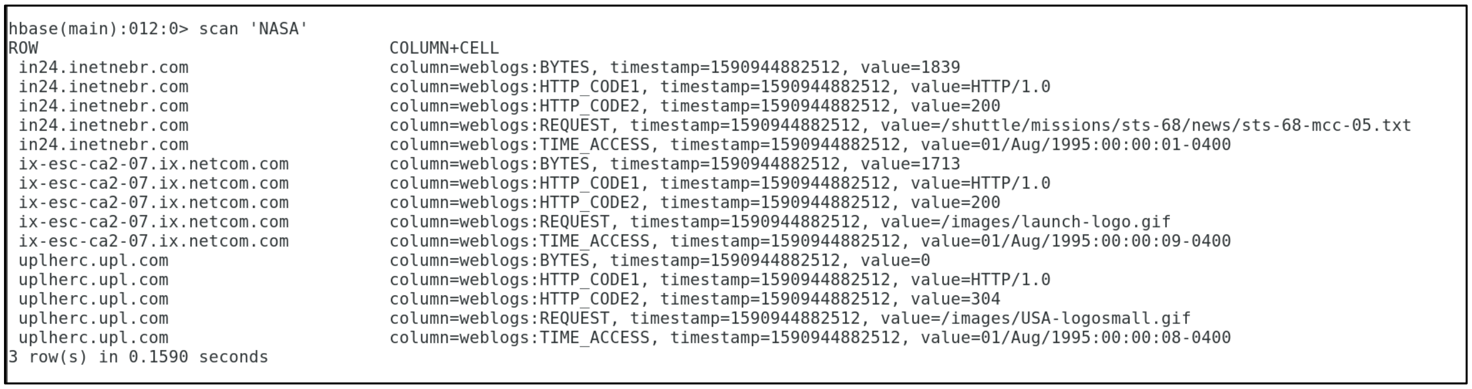
```
#Describe the sink:
a1.sinks.sk1.type = hbase
a1.sinks.sk1.table = NASA
a1.sinks.sk1.columnFamily = weblogs
a1.sinks.sk1.serializer = org.apache.flume.sink.hbase.RegexHbaseEventSerializer
a1.sinks.sk1.serializer.rowKeyIndex = 0
a1.sinks.sk1.serializer.regex = (.+),(.+),(.+),(.+),(.+),(.+)
a1.sinks.sk1.serializer.colNames = ROW_KEY, TIME_ACCESS, REQUEST, HTTP_CODE1,
HTTP_CODE2, BYTES
```

Where:

- serializer.rowKeyIndex: indicative if I should insert event from any column in ROW_KEY
- serializer.regex: regular expression to be searched for in events, it is from there that the file will be inserted in different columns within the column family.
- serializer.colNames: column names within the HBase column family where the data will be inserted.

An interesting fact is in the last line of code “a1.sinks.sk1.serializer.colNames = ROW_KEY, TIME_ACCESS, REQUEST, HTTP_CODE1, HTTP_CODE2, BYTES” as these columns were not specified at the time we created the table within HBase, they are implemented at the moment that the Flume sink will transport the data, giving us more freedom when creating the project with HBase.

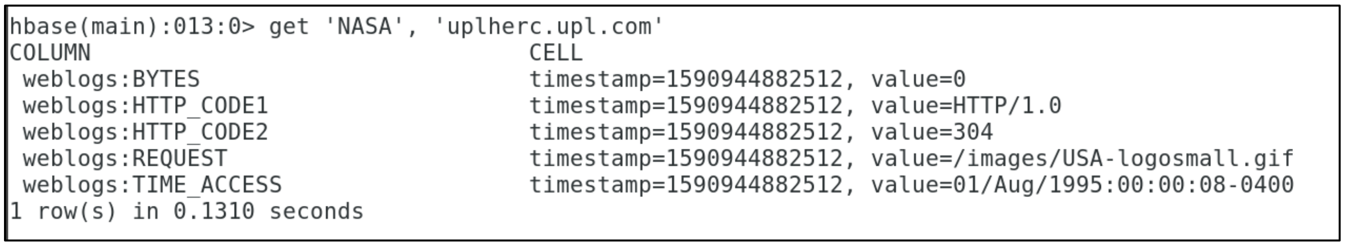
So, I deleted the previous table in HBase and created a new one: create 'NASA', {NAME => 'weblogs'}, initially carrying the first 3 log events from the NASA server with the following result:



```
hbase(main):012:0> scan 'NASA'
ROW                                 COLUMN+CELL
 in24.inetnebr.com                  column=weblogs:BYTES, timestamp=1590944882512, value=1839
 in24.inetnebr.com                  column=weblogs:HTTP_CODE1, timestamp=1590944882512, value=HTTP/1.0
 in24.inetnebr.com                  column=weblogs:HTTP_CODE2, timestamp=1590944882512, value=200
 in24.inetnebr.com                  column=weblogs:REQUEST, timestamp=1590944882512, value=/shuttle/missions/sts-68/news/sts-68-mcc-05.txt
 in24.inetnebr.com                  column=weblogs:TIME_ACCESS, timestamp=1590944882512, value=01/Aug/1995:00:00:01-0400
 ix-esc-ca2-07.ix.netcom.com        column=weblogs:BYTES, timestamp=1590944882512, value=1713
 ix-esc-ca2-07.ix.netcom.com        column=weblogs:HTTP_CODE1, timestamp=1590944882512, value=HTTP/1.0
 ix-esc-ca2-07.ix.netcom.com        column=weblogs:HTTP_CODE2, timestamp=1590944882512, value=200
 ix-esc-ca2-07.ix.netcom.com        column=weblogs:REQUEST, timestamp=1590944882512, value=/images/launch-logo.gif
 ix-esc-ca2-07.ix.netcom.com        column=weblogs:TIME_ACCESS, timestamp=1590944882512, value=01/Aug/1995:00:00:09-0400
 uplherc.upl.com                    column=weblogs:BYTES, timestamp=1590944882512, value=0
 uplherc.upl.com                    column=weblogs:HTTP_CODE1, timestamp=1590944882512, value=HTTP/1.0
 uplherc.upl.com                    column=weblogs:HTTP_CODE2, timestamp=1590944882512, value=304
 uplherc.upl.com                    column=weblogs:REQUEST, timestamp=1590944882512, value=/images/USA-logosmall.gif
 uplherc.upl.com                    column=weblogs:TIME_ACCESS, timestamp=1590944882512, value=01/Aug/1995:00:00:08-0400
3 row(s) in 0.1590 seconds
```

Figure 9 - HBase weblogs NASA

Now note that we can perform queries by host obtaining more structured information and easy to interpret:



```
hbase(main):013:0> get 'NASA', 'uplherc.upl.com'
COLUMN                                CELL
 weblogs:BYTES                        timestamp=1590944882512, value=0
 weblogs:HTTP_CODE1                   timestamp=1590944882512, value=HTTP/1.0
 weblogs:HTTP_CODE2                   timestamp=1590944882512, value=304
 weblogs:REQUEST                      timestamp=1590944882512, value=/images/USA-logosmall.gif
 weblogs:TIME_ACCESS                  timestamp=1590944882512, value=01/Aug/1995:00:00:08-0400
1 row(s) in 0.1310 seconds
```

Figure 10 - HBase query by host

Thus we concluded the project and, with the concepts studied and presented here, we can carry out new collections of NASA server logs with Apache Flume and store each event in a specific column within HBase column family.

3. Final Considerations

The use of Apache Hadoop for a Big Data environment only brought advantages to the project. Using distributed storage and parallel processing helps to make the project more flexible and generate productivity without much bureaucracy with collected data. The most laborious and determining part is the initial cluster configurations, that is, adjusting all parameters takes time and attention.

After configuring the cluster, we installed Hadoop ecosystem, emphasizing two main frameworks: Flume and HBase. Flume was used to move logs into HDFS and later to a table in HBase.

This business problem aimed to transport logs from NASA's server into HBase and for that to happen we used Apache Flume where we created an agent to communicate and transport logs configuring some parameters such as source, channel and sink.

With a configured agent it was possible to see the power of this tool when transporting with precision and agility several logs into HDFS and it can even adjust folder and file nomenclatures during transport. As soon as Flume test with HDFS was successful, it was possible to enter in HBase settings to receive and store the logs in this NoSQL Database.

With HBase we initially created a table and a family of columns to receive the logs, being successful in the transfer process with Flume, but leaving all the information of a log event condensed in one line, thus making the analysis process more difficult.

To improve this process we also took advantage of the integration power between Apache Flume and Apache HBase, where we arrived at an interesting point in the project because it was not necessary to create several columns within a family of columns in HBase, in fact within the agent it was possible to determine which regular expressions should be considered for separating the file into columns and thus insert directly into HBase each log event separated by column in a column family.

The above process, of creating the column rules in the agent, brought more flexibility to the project and showed how it would be possible to automate the transport process for other types of files. We can follow the same concept and extend the features described in this project to other business problems that involve information management and log data management.