ECE Paris

Hadoop Project

Secure Data Flow for Log flow with ArcSight

Adrien CHEVRIER – Pablo TABALES
17/12/2017

Introduction

The objective of this project is to present a solution for the needs of an energy company with their Data Center, the amount of data is growing and adopting new platforms for their system is now a possibility, therefore we will propose an architecture that will allow them to maintain their priorities and store all the information they need.

The company First Energy located in Ohio delivers energy for multiple clients across the US, having their own Data Center for multiple purposes. By legal reasons companies are now asked to retain their logs in case of a network, server or security event, the type of logs they handle are:

- Hardware logs - system messages about server or network hardware incidents and health. For example, a server memory chip logs a high number of read/write errors which indicates a pending failure. Or a power supply fan begins to rotate at a higher or lower speed than it is supposed to, which could indicate a hardware event.

- Operating System logs - these are system messages about the server or switch system software. In our environment, this would include Microsoft Windows, IBM AIX, Red Hat Linux, and Cisco network switch OS. Messages could include events such as network TCPIP process failures, Linux kernel bugs, or Windows reboots.

- Application event logs - these are logs gathered by specific applications running on our servers: SAP, TSM backup, Oracle, PowerPlan, etc. Events could include application faults, restarts, read/write errors, and so on. We don't track every instance of every application (development or testing environments, for example), but anything critical to the daily operation of the business is collected.

- User access and security logs - these track all user access to servers, operating systems, and applications, both successful logins and failed attempts. Also all "probing" events -- attempts to hack in -- are logged.

They have different teams to monitor all these and keep the Data Center working, however handling with all these information they're suspect of different attacks so a Security team was created to protect the flow of this logs and keep them in a safe place, they are currently using ArcSight. ArcSight is a big data solution provided by Hewlett Packard with a Kafka infrastructure to handle event and do data streaming. Letting them do trend analysis to keep track of the behavior of systems and avoid problems, but the HP solution is not enough for the amount of data they're working with.

ArcSight allows the possibility to have Hadoop, this is were our team has a proposal of how to do this the best way possible, since security is a priority and many logs are not being able to be processed or retrieved the need for a reliable data flow is important.

What do we propose?

We set up an architecture that aggregates dataflows of logs coming from the different locations. We will first collect logs at small scale, process them and send them to a distributed system able to handle the millions of logs coming from the different machines. Using a distributed system allows to improve the computer power by adding new machines as the number of entry logs increases.
Our distributed system consists in a streaming engine that aggregates all the data, a processing engine that cleans data, a database that stores the logs.

Technologies chosen

The different technologies must fit our needs as most Big data technologies are optimized for a certain type of process.
We chose to handle dataflow management at small scale with Nifi, this tool initially developed by NSA provides a graphical interface to manage dataflows and automate the flow of data between systems. We will use it to connect the remote systems with our distributed system.
NiFi executes within a JVM on a host operating system. It consists into:

- A **webserver** that hosts NiFi's HTTP-based command and control API.
- A **flow controller** that manages the operations, provides threads for extensions to run on.
- A **flowfile repository** that keeps track of the state of active FlowFiles. A flowFile is a data record, which consists of a pointer to its content.
- A **content repository** that contains the actual content of flowFiles.
- A **provenance repository** where all provenance event data is stored.

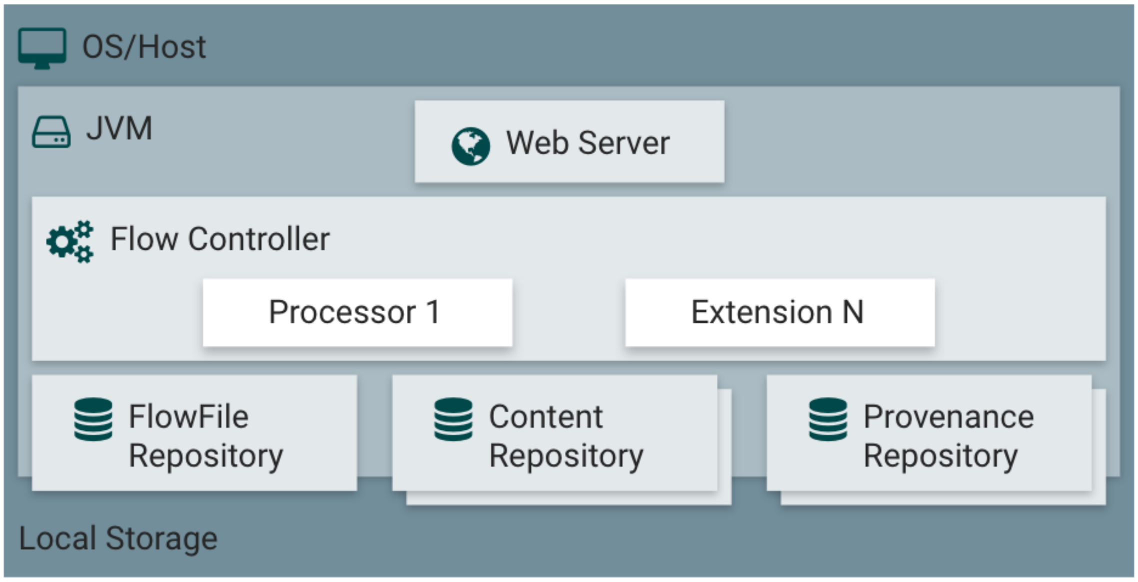


Figure 1: Nifi architecture

The streaming engine that will aggregate all the data and transfer it to the distributed database is Kafka, it was designed for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, really fast, and runs in production for thousands of companies.
Kafka lets applications called producers publish streams of records in categories called topics. Applications, also called consumers subscribe to one or more topics and process the stream of records to produce them. Kafka topics are divided into partitions to allow parallelization by splitting the data into multiple nodes, called brokers in kafka terminology.

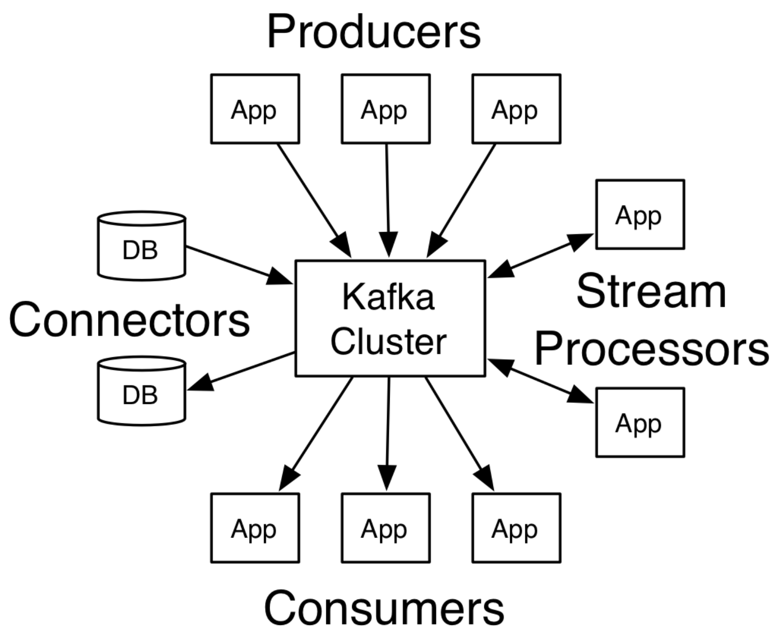


Figure 2: Kafka architecture

This stream will be sent to Spark, Spark Streaming is a n extension of the core Spark API that uses structured data streaming with an ease of use for multiple languages like Java, Scala and Python, Fault Tolerance with the possibility to make computations from the streams and do batch processing, we can join streams and data and finally store them. We can digest data from multiple sources, Kafka included, and permit us to make map reduce jobs over them and make use of all the libraries to the data streams.

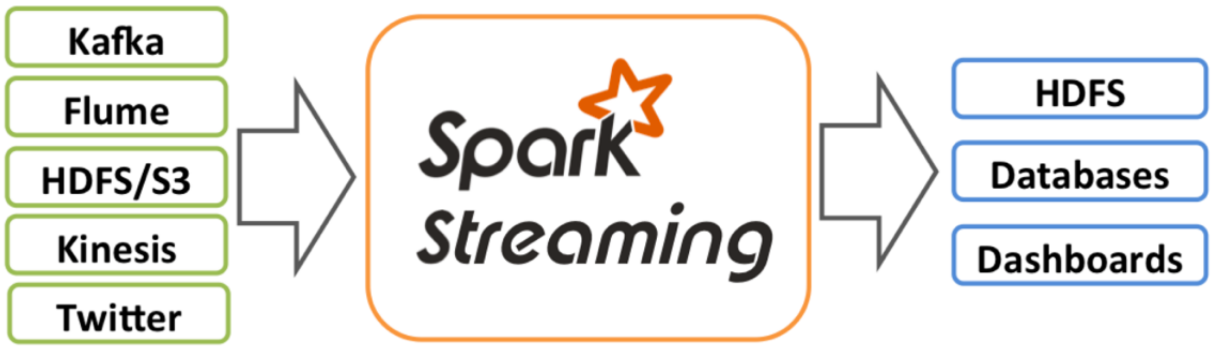
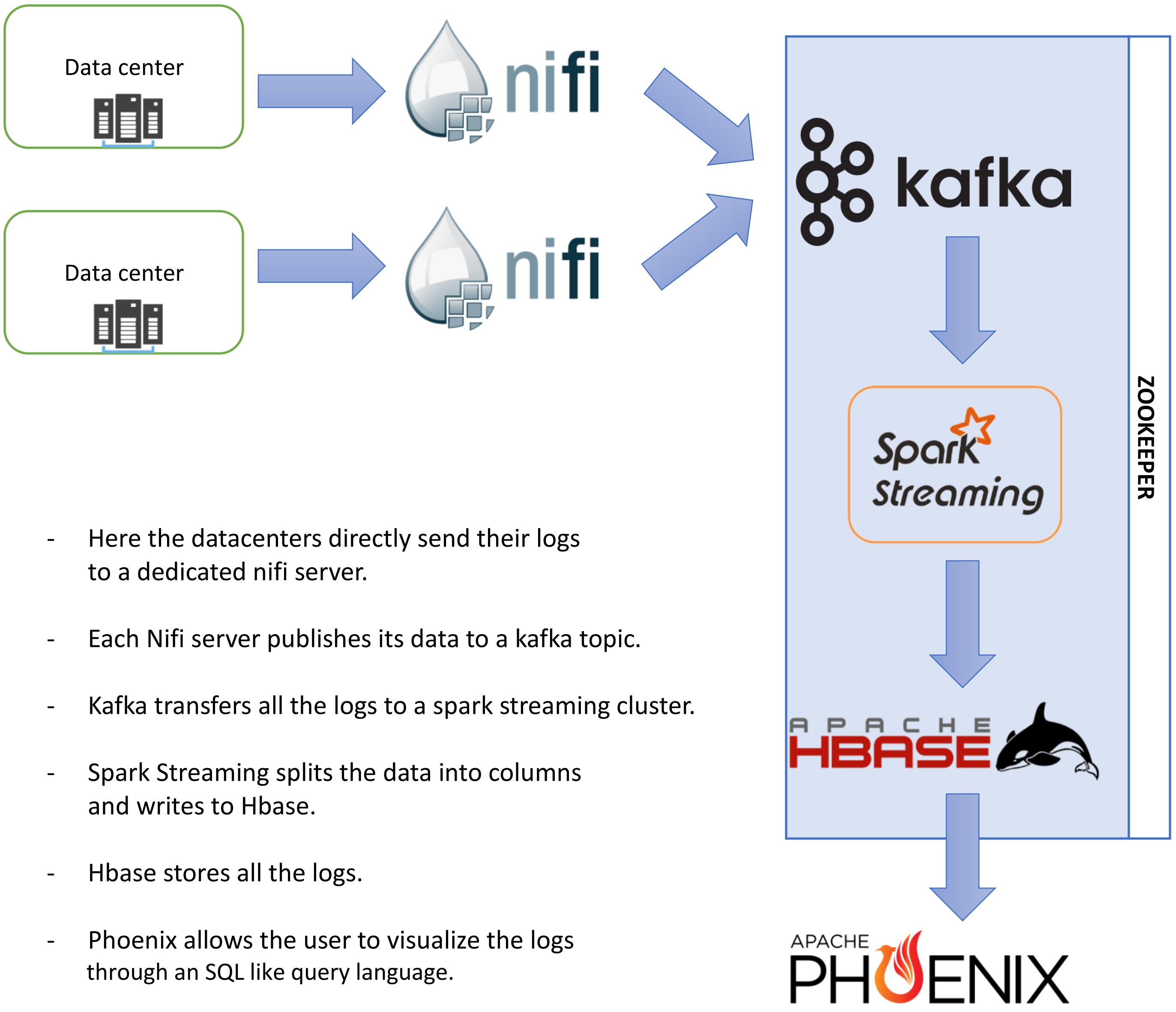


Figure 3: Possible connections with Spark streaming

According to the needs of the client we are looking forward to just store the data, since it's what the law recquires for them and Hbase is the best solution to host very large tables in a very fast way. Hbase is a distributed, scalable, big data store inspired by Google's Bigtable meant to do bulk loads, so we can insert and select data in a very fast. The main objective is storing a big number of logs to later be able to make simple queries, that doesn't need any computations since some of them are already done with Spark.

Architecture

We built a final architecture to stream, process and store all the data into Hbase.



- Here the datacenters directly send their logs to a dedicated nifi server.

- Each Nifi server publishes its data to a kafka topic.

- Kafka transfers all the logs to a spark streaming cluster.

- Spark Streaming splits the data into columns and writes to Hbase.

- Hbase stores all the logs.

- Phoenix allows the user to visualize the logs through an SQL like query language.

- Zookeeper maintains coordination

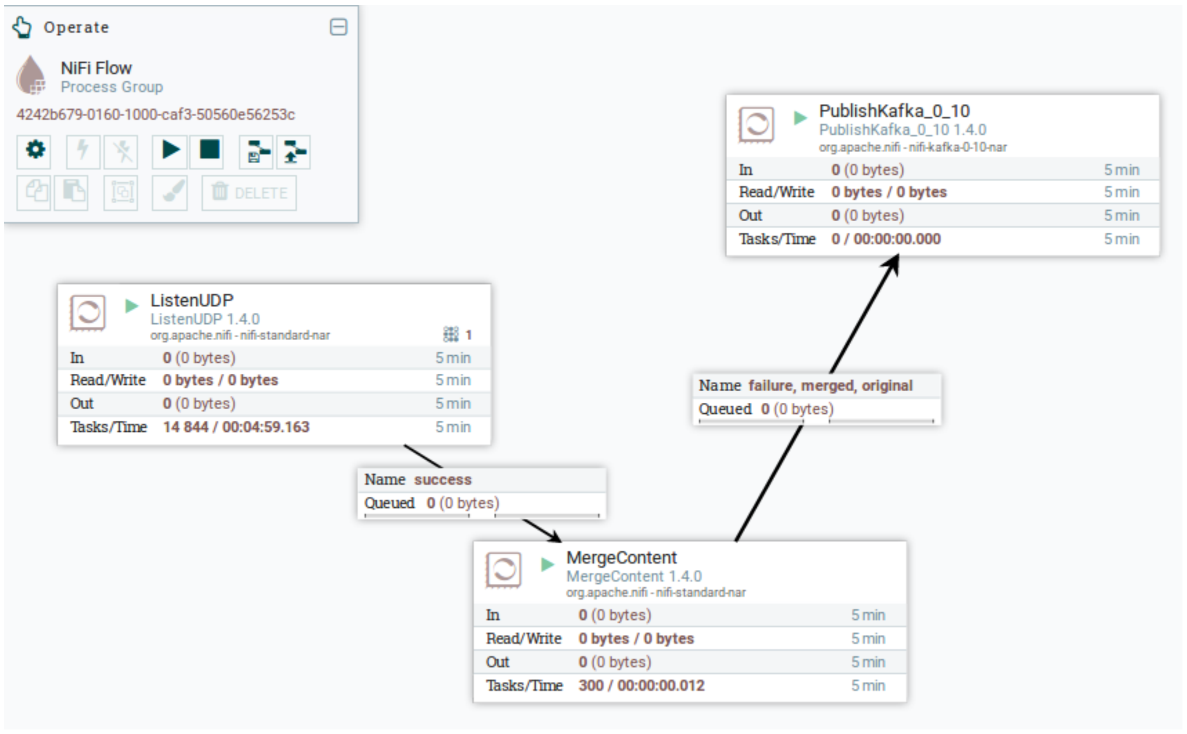
Implementation

Nifi

To simulate the full pipeline, we decided to install

To handle the data coming from several sources with Nifi, we installed Nifi on 2 local machines.

As said previously, the goal of Nifi is to aggregate the logs in a region before sending them to a centralized cluster. In order to do that, we will open a udp port that Nifi will listen on, merge the content to build packets and publish it to kafka. Here Nifi is a producer of kafka.



ListenUDP
ListenUDP 1.4.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	14 844 / 00:04:59.163	5 min

MergeContent
MergeContent 1.4.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	300 / 00:00:00.012	5 min

PublishKafka_0_10
PublishKafka_0_10 1.4.0
org.apache.nifi - nifi-kafka-0-10-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Figure 4: Nifi workflow

Nifi is a producer of Kafka as it publishes the logs to the topic, consumers will consume the messages published by Nifi.

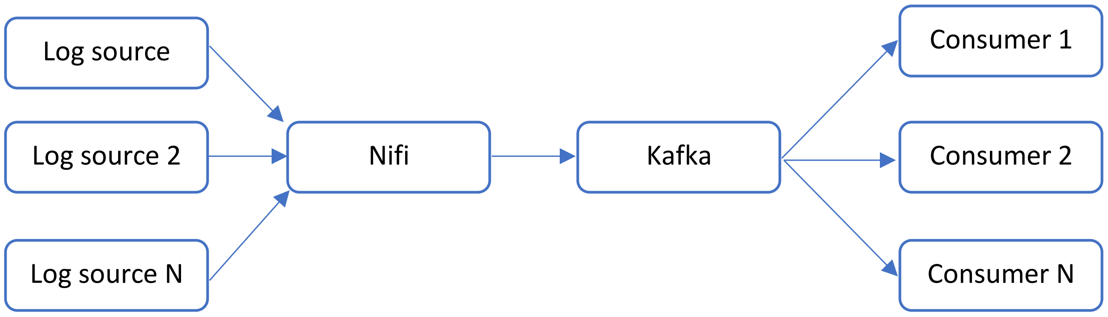
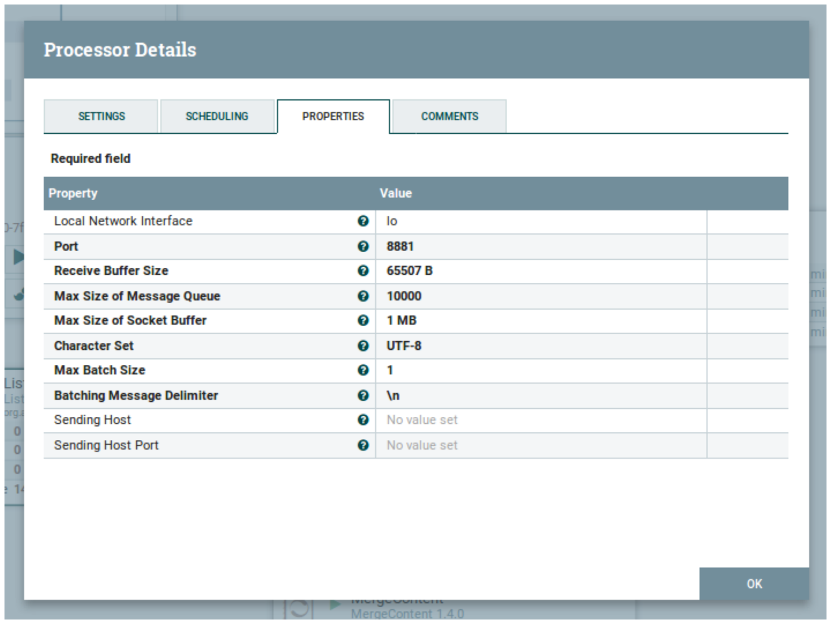


Figure 5: Connection between Nifi and Kafka

We will show the configuration in each of the processor's in Nifi:

ListenUDP

Here we set a listener that will collect all the messages sent to a certain port of its machine.



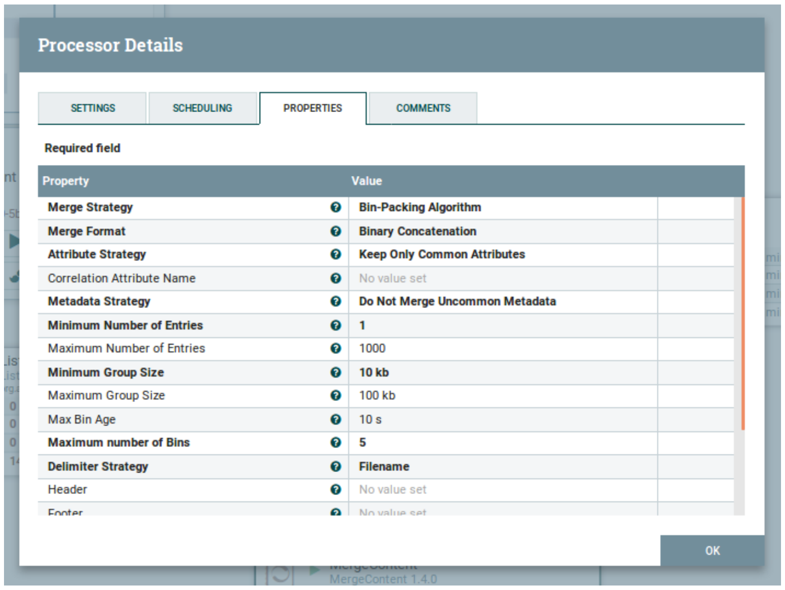
Property	Value
Local Network Interface	lo
Port	8881
Receive Buffer Size	65507 B
Max Size of Message Queue	10000
Max Size of Socket Buffer	1 MB
Character Set	UTF-8
Max Batch Size	1
Batching Message Delimiter	\n
Sending Host	No value set
Sending Host Port	No value set

Figure 6: ListenUDP properties

We can see that we are able to specify different parameters for the processor, from the port to the different limits on the data received.

MergeContent

Merge content has two different purposes ; the first one is to avoid congestion of data flow in NiFi and the second one is to make sure that empty data is not sent to Kafka.


Processor Details

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

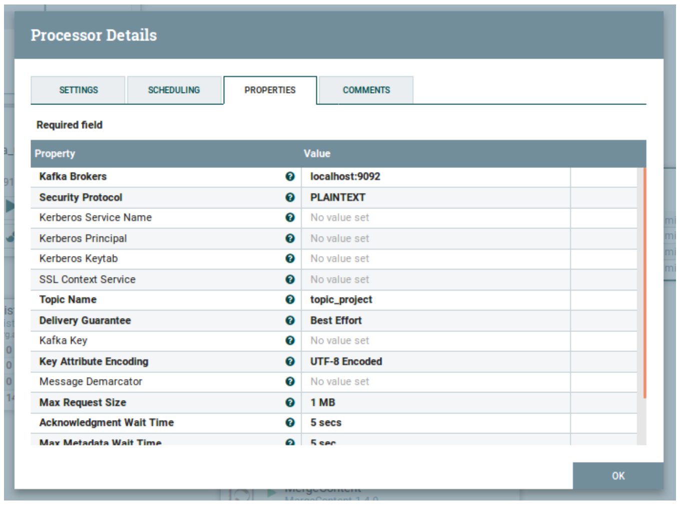
Property	Value
Merge Strategy	**Bin-Packing Algorithm**
Merge Format	**Binary Concatenation**
Attribute Strategy	**Keep Only Common Attributes**
Correlation Attribute Name	No value set
Metadata Strategy	**Do Not Merge Uncommon Metadata**
Minimum Number of Entries	**1**
Maximum Number of Entries	1000
Minimum Group Size	**10 kb**
Maximum Group Size	100 kb
Max Bin Age	10 s
Maximum number of Bins	**5**
Delimiter Strategy	**Filename**
Header	No value set
Footer	No value set

OK

Figure 7: MergeContent properties

PublishKafka

In this last processor all parameter are given where our Kafka consumer is waiting for the our stream, a very important value is the Topic Name cause the way Kafka works with the Topic Name as an ID.


Processor Details

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Kafka Brokers	**localhost:9092**
Security Protocol	**PLAINTEXT**
Kerberos Service Name	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name	**topic_project**
Delivery Guarantee	**Best Effort**
Kafka Key	No value set
Key Attribute Encoding	**UTF-8 Encoded**
Message Demarcator	No value set
Max Request Size	**1 MB**
Acknowledgment Wait Time	**5 secs**
Max Metadata Wait Time	**5 sec**

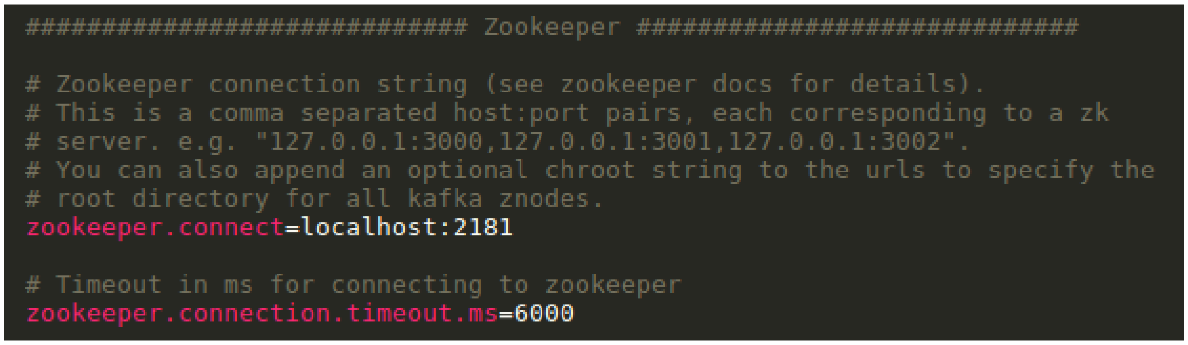
OK

Figure 8: PublishKafka properties

Kafka configuration

Now that we have configured Nifi to receive informations and publish them on a kafka topic, we have to set a kafka server to receive these logs. We installed kafka on a machine and tuned its configuration to fit our needs.

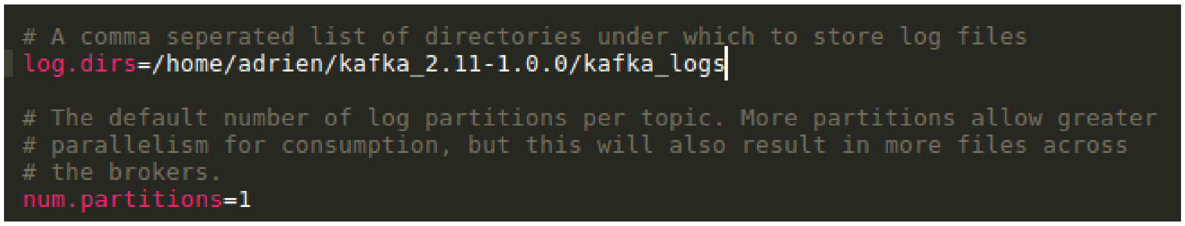
Among Important values to set for Kafka server properties, the first one is the information of the Zookeeper. We set its quorum and port.



```
############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000
```

Figure 9: Zookeeper configuration for kafka in server.properties

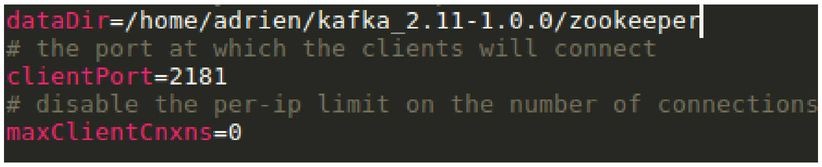
We keep the number of partitions to 1, as we only have one consumer we do not need important parallelization of the topic



```
# A comma seperated list of directories under which to store log files
log.dirs=/home/adrien/kafka_2.11-1.0.0/kafka_logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1
```

Figure 10: kafka configuration

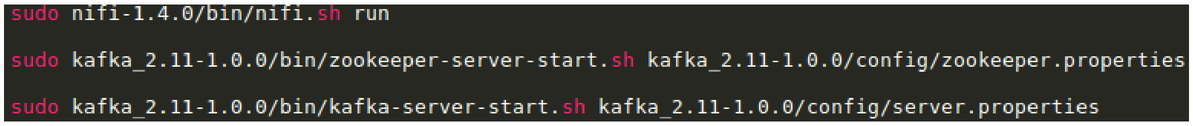
The properties in Zookeeper are also modified.



```
dataDir=/home/adrien/kafka_2.11-1.0.0/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections
maxClientCnxns=0
```

Figure 11: Zookeeper configuration

Once the configuration is finished, we can launch Nifi, zookeeper and Kafka servers.

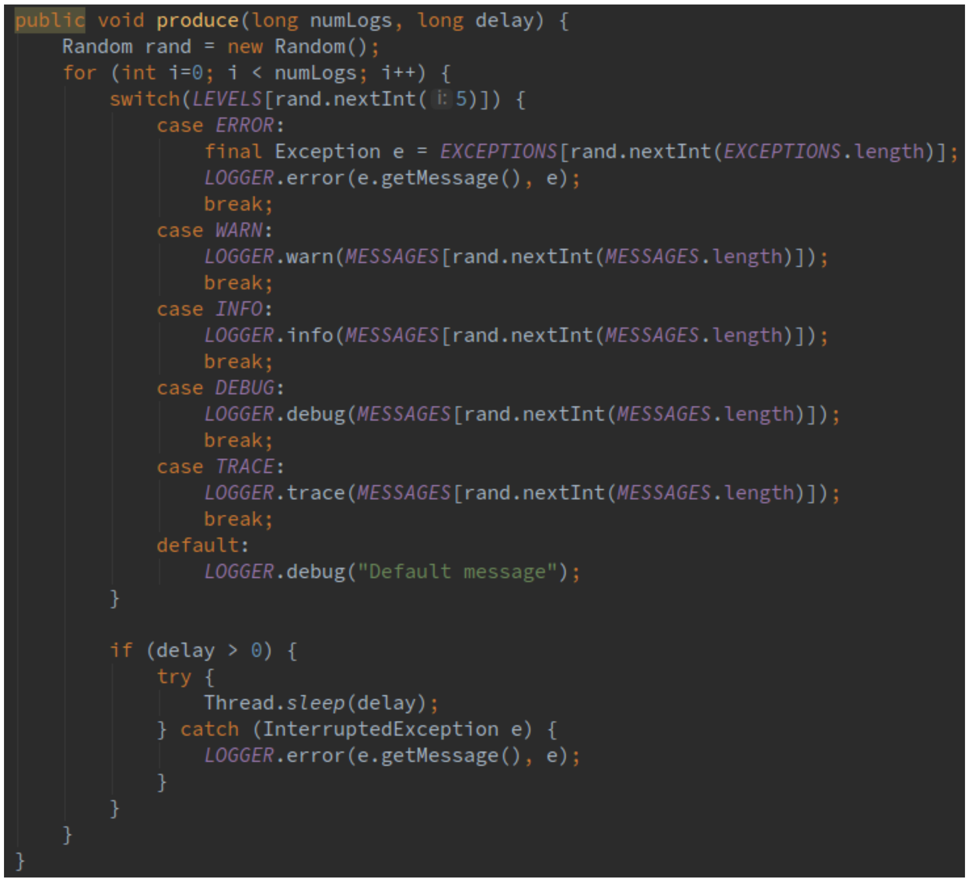


```
sudo nifi-1.4.0/bin/nifi.sh run

sudo kafka_2.11-1.0.0/bin/zookeeper-server-start.sh kafka_2.11-1.0.0/config/zookeeper.properties

sudo kafka_2.11-1.0.0/bin/kafka-server-start.sh kafka_2.11-1.0.0/config/server.properties
```

Figure 12 :Commands used to launch the system

JSON log generator

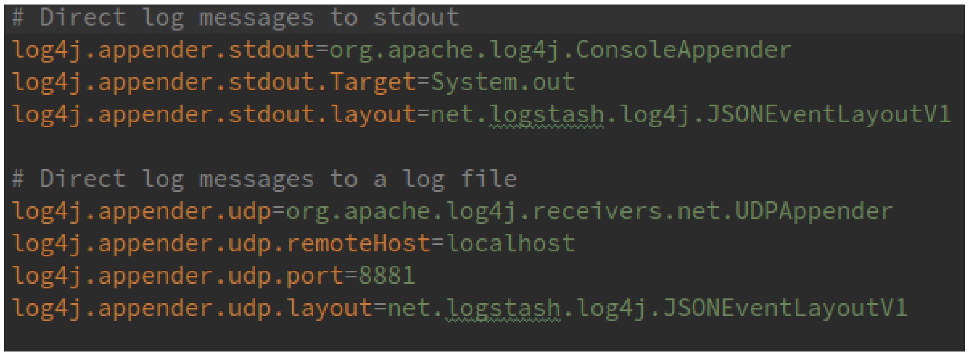
To simulate machine sending logs to Nifi, we use a java application that creates random logs and sends them to a given address. We can see that 6 types of logs are generated with random text, the method creates a certain number of them and the time interval regulates the number of logs sent per second.



```java
public void produce(long numLogs, long delay) {
    Random rand = new Random();
    for (int i=0; i < numLogs; i++) {
        switch(LEVELS[rand.nextInt( i: 5)]) {
            case ERROR:
                final Exception e = EXCEPTIONS[rand.nextInt(EXCEPTIONS.length)];
                LOGGER.error(e.getMessage(), e);
                break;
            case WARN:
                LOGGER.warn(MESSAGES[rand.nextInt(MESSAGES.length)]);
                break;
            case INFO:
                LOGGER.info(MESSAGES[rand.nextInt(MESSAGES.length)]);
                break;
            case DEBUG:
                LOGGER.debug(MESSAGES[rand.nextInt(MESSAGES.length)]);
                break;
            case TRACE:
                LOGGER.trace(MESSAGES[rand.nextInt(MESSAGES.length)]);
                break;
            default:
                LOGGER.debug("Default message");
        }

        if (delay > 0) {
            try {
                Thread.sleep(delay);
            } catch (InterruptedException e) {
                LOGGER.error(e.getMessage(), e);
            }
        }
    }
}
```

Figure 13: Log producer

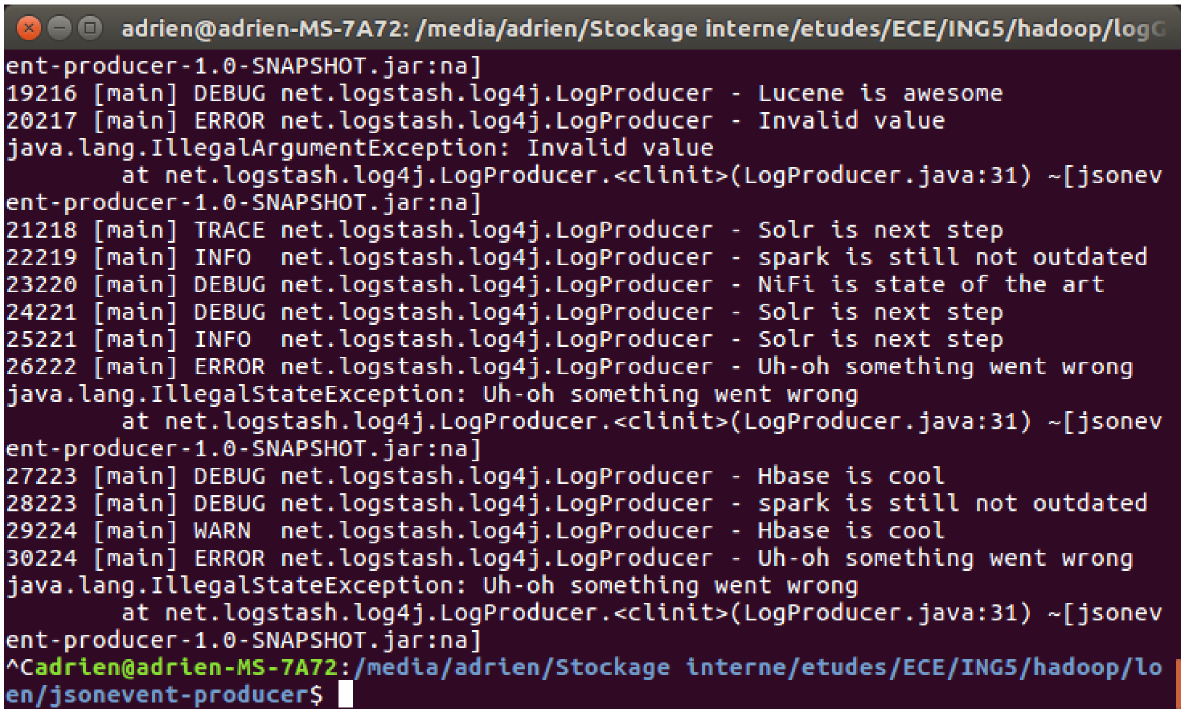
A property file has been created to them the logs both to the system and to a remote host using udp. This udp address will be used by Nifi to collect the messages from all the logs generators. These logs are sent to the remote host using JSON format to ease the processing of the logs later on.



```properties
# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=net.logstash.log4j.JSONEventLayoutV1

# Direct log messages to a log file
log4j.appender.udp=org.apache.log4j.receivers.net.UDPAppender
log4j.appender.udp.remoteHost=localhost
log4j.appender.udp.port=8881
log4j.appender.udp.layout=net.logstash.log4j.JSONEventLayoutV1
```

Figure 14: configuration to send logs to Nifi

We can verify that logs are properly generated in the console once the java program is launched.



```
adrien@adrien-MS-7A72: /media/adrien/Stockage interne/etudes/ECE/ING5/hadoop/logG
ent-producer-1.0-SNAPSHOT.jar:na]
19216 [main] DEBUG net.logstash.log4j.LogProducer - Lucene is awesome
20217 [main] ERROR net.logstash.log4j.LogProducer - Invalid value
java.lang.IllegalArgumentException: Invalid value
        at net.logstash.log4j.LogProducer.<clinit>(LogProducer.java:31) ~[jsonev
ent-producer-1.0-SNAPSHOT.jar:na]
21218 [main] TRACE net.logstash.log4j.LogProducer - Solr is next step
22219 [main] INFO  net.logstash.log4j.LogProducer - spark is still not outdated
23220 [main] DEBUG net.logstash.log4j.LogProducer - NiFi is state of the art
24221 [main] DEBUG net.logstash.log4j.LogProducer - Solr is next step
25221 [main] INFO  net.logstash.log4j.LogProducer - Solr is next step
26222 [main] ERROR net.logstash.log4j.LogProducer - Uh-oh something went wrong
java.lang.IllegalStateException: Uh-oh something went wrong
        at net.logstash.log4j.LogProducer.<clinit>(LogProducer.java:31) ~[jsonev
ent-producer-1.0-SNAPSHOT.jar:na]
27223 [main] DEBUG net.logstash.log4j.LogProducer - Hbase is cool
28223 [main] DEBUG net.logstash.log4j.LogProducer - spark is still not outdated
29224 [main] WARN  net.logstash.log4j.LogProducer - Hbase is cool
30224 [main] ERROR net.logstash.log4j.LogProducer - Uh-oh something went wrong
java.lang.IllegalStateException: Uh-oh something went wrong
        at net.logstash.log4j.LogProducer.<clinit>(LogProducer.java:31) ~[jsonev
ent-producer-1.0-SNAPSHOT.jar:na]
^Cadrien@adrien-MS-7A72:/media/adrien/Stockage interne/etudes/ECE/ING5/hadoop/lo
en/jsonevent-producer$
```

Figure 15 : Logs produced displayed in terminal

In general terms what we are doing is sending all logs coming from Log4j JSON generator program to NiFi to manage all coming to the designated port as shown in the first picture. That port listens to all content then pass them to our Merge processor, as mentioned before is has two purposes; avoid congestion and send real values, then we send all these to the Kafka consumer with the specific Topic Name for ID.

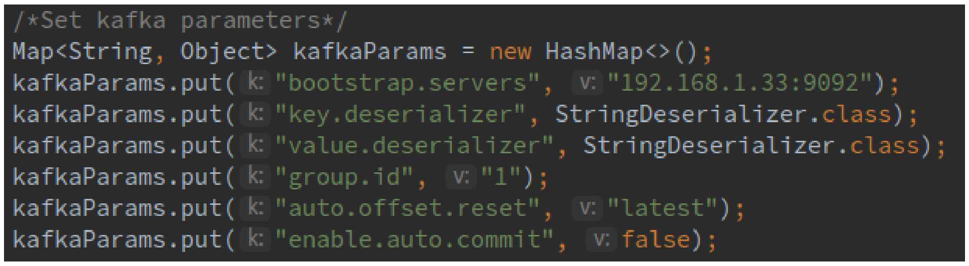
Spark Streaming

The next step is to send these logs to Spark Streaming for possible aggregations if needed, we receive the data and then the objective is to store it in HBase for storage purposes mostly.

we can create the jar for our Java job and launch it.

We programmed a class "VerySimpleStreamingApp" with a main that initializes the connection with the kafka server, starts the stream and receives logs as RDDs, and for each RDD puts the data into a table of an Hbase database. As the logs are sent in a Json format, we can use spark to convert the data and split the values into corresponding columns.

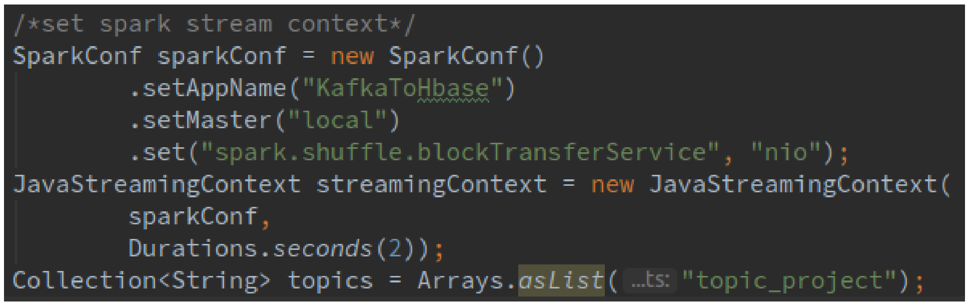
In the kafka parameters, the main parameters are the server address, the key/value classes, the group ID:



```
/*Set kafka parameters*/
Map<String, Object> kafkaParams = new HashMap<>();
kafkaParams.put( k: "bootstrap.servers",  v: "192.168.1.33:9092");
kafkaParams.put( k: "key.deserializer", StringDeserializer.class);
kafkaParams.put( k: "value.deserializer", StringDeserializer.class);
kafkaParams.put( k: "group.id",  v: "1");
kafkaParams.put( k: "auto.offset.reset",  v: "latest");
kafkaParams.put( k: "enable.auto.commit",  v: false);
```

Figure 16: Kafka parameters in Spark job

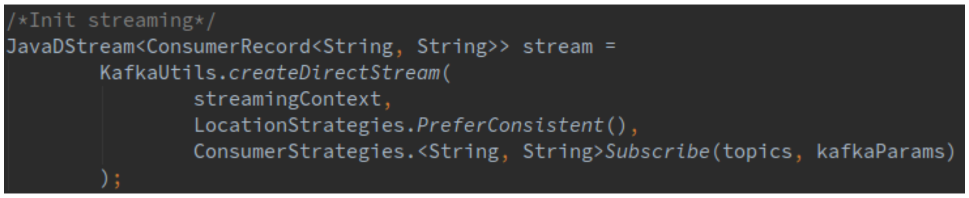
We set the Spark configurations and change the "blockTransferService" to "nio" to avoid an I/O exception apearing in some of our tests. The topic name is the topic that nifi sends the logs to.



```
/*set spark stream context*/
SparkConf sparkConf = new SparkConf()
        .setAppName("KafkaToHbase")
        .setMaster("local")
        .set("spark.shuffle.blockTransferService", "nio");
JavaStreamingContext streamingContext = new JavaStreamingContext(
        sparkConf,
        Durations.seconds(2));
Collection<String> topics = Arrays.asList( ...ts: "topic_project");
```

Figure 17: Spark job configuration

Once the configuration for the Kafka/Spark connection is made, we initialize the stream:



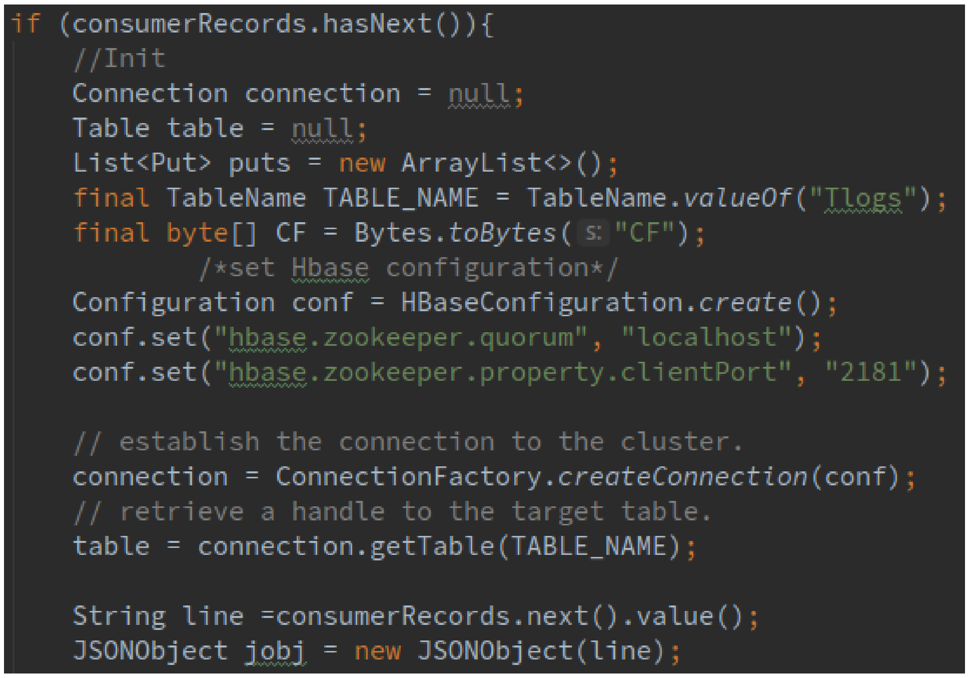
```
/*Init streaming*/
JavaDStream<ConsumerRecord<String, String>> stream =
        KafkaUtils.createDirectStream(
                streamingContext,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.<String, String>Subscribe(topics, kafkaParams)
        );
```

Figure 18: Stream initialisation

Spark will yet receive all the messages sent to the kafka topic "topic_project, the next step is to handle them and send them to Hbase.

As said previously, each message is recieved as an RDD, the function,"stream.foreachRDD" allows us to process them as they arrive. We chose to create a new Hbase connection for each RDD, we connect Spark to the table Tlogs and the column family CF. We also define the zookeeper quorum and port we use to initialize the connection.

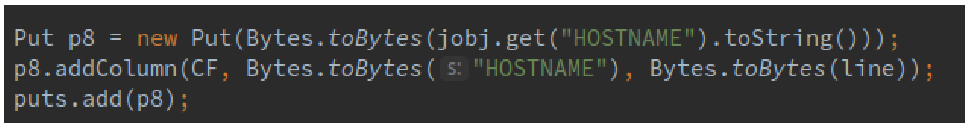
Our log generator sends messages in a JSON formatthat stays unaltred until he arrives to spark, therefore we can store each log in a JSONObject that will be used to send each information to a dedicated column.



```
if (consumerRecords.hasNext()){
    //Init
    Connection connection = null;
    Table table = null;
    List<Put> puts = new ArrayList<>();
    final TableName TABLE_NAME = TableName.valueOf("Tlogs");
    final byte[] CF = Bytes.toBytes( s: "CF");
            /*set Hbase configuration*/
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "localhost");
    conf.set("hbase.zookeeper.property.clientPort", "2181");

    // establish the connection to the cluster.
    connection = ConnectionFactory.createConnection(conf);
    // retrieve a handle to the target table.
    table = connection.getTable(TABLE_NAME);

    String line =consumerRecords.next().value();
    JSONObject jobj = new JSONObject(line);
```

Figure 19: hbase put configuration

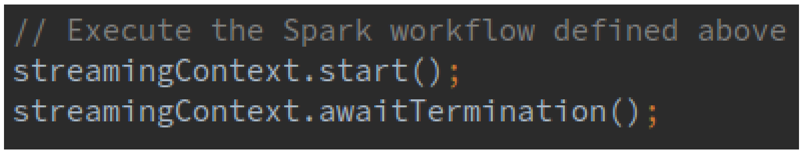
For each JSON key, we read its value, store it in a variable Put used to send data to Hbase, we add a column destination with the name of the json key and send it to Hbase.



```
Put p8 = new Put(Bytes.toBytes(jobj.get("HOSTNAME").toString()));
p8.addColumn(CF, Bytes.toBytes( s: "HOSTNAME"), Bytes.toBytes(line));
puts.add(p8);
```

Figure 20: Code that prepares Hbase Put

Finally we start and stop the stream with the following lines:



```
// Execute the Spark workflow defined above
streamingContext.start();
streamingContext.awaitTermination();
```

Figure 21: Streaming Launching

Once the jar file is compiled and available on the Spark cluster, we can launch it using spark-submit:

spark-submit --class com.stream.KSH.VerySimpleStreamingApp --master local streaming-1.0-SNAPSHOT.jar

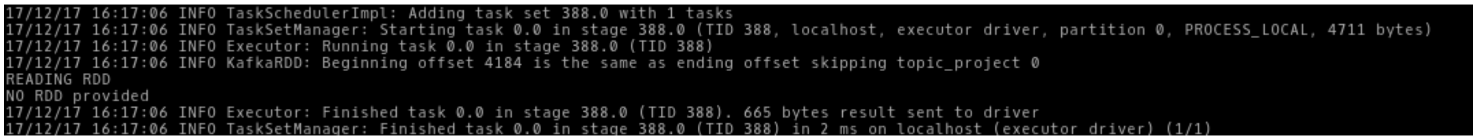
With this job Spark Streaming will be listening to the port where our Kafka Producer is sending the data flow, for later batch the flow and store it in HBase.

Now we can launch Hbase with "bin/start-hbase.sh" and start spark streaming to send the messages to the database

Obtaining results from the complete pipeline

We can verify that spark is properly connected to kafka by watching the data displayed on the terminal when transferred or not.

We first start Nifi, Kafka and Spark without starting the log generator:

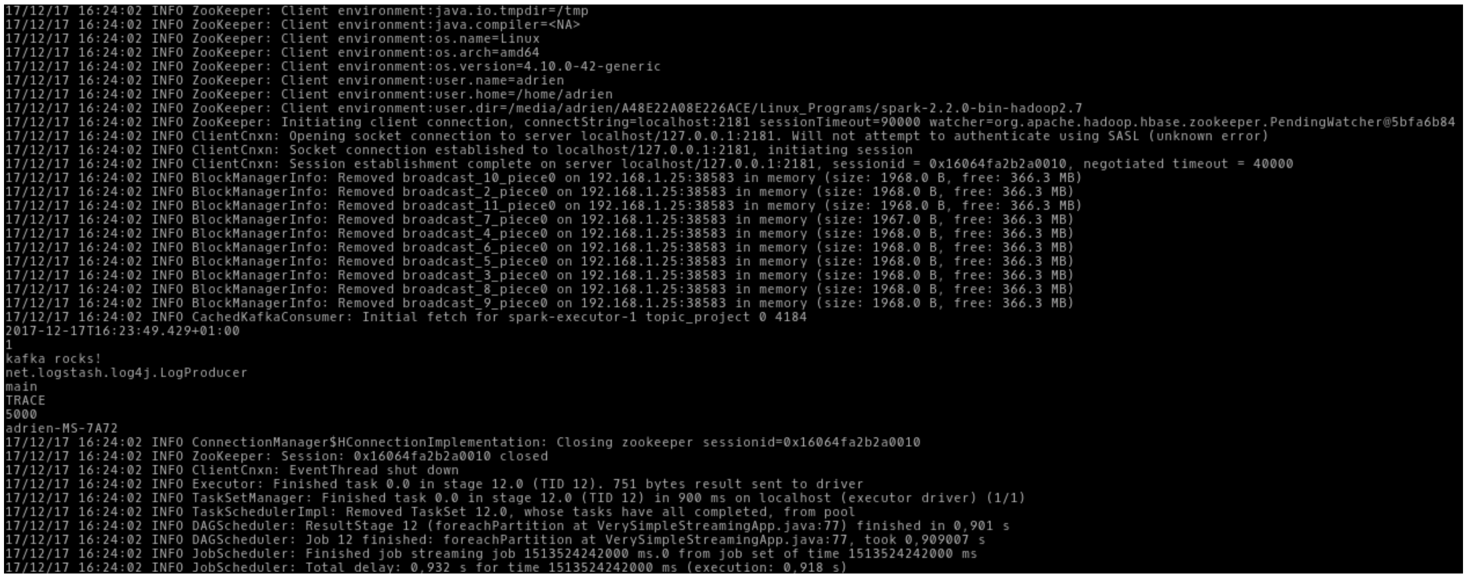


```
17/12/17 16:17:06 INFO TaskSchedulerImpl: Adding task set 388.0 with 1 tasks
17/12/17 16:17:06 INFO TaskSetManager: Starting task 0.0 in stage 388.0 (TID 388, localhost, executor driver, partition 0, PROCESS_LOCAL, 4711 bytes)
17/12/17 16:17:06 INFO Executor: Running task 0.0 in stage 388.0 (TID 388)
17/12/17 16:17:06 INFO KafkaRDD: Beginning offset 4184 is the same as ending offset skipping topic_project 0
READING RDD
NO RDD provided
17/12/17 16:17:06 INFO Executor: Finished task 0.0 in stage 388.0 (TID 388). 665 bytes result sent to driver
17/12/17 16:17:06 INFO TaskSetManager: Finished task 0.0 in stage 388.0 (TID 388) in 2 ms on localhost (executor driver) (1/1)
```

Figure 22: Spark streaming output with no log generated

Above we can see a job launched in Spark streaming, No RDD is provided as no data was taken from the topic. Nothing will be written to Hbase.

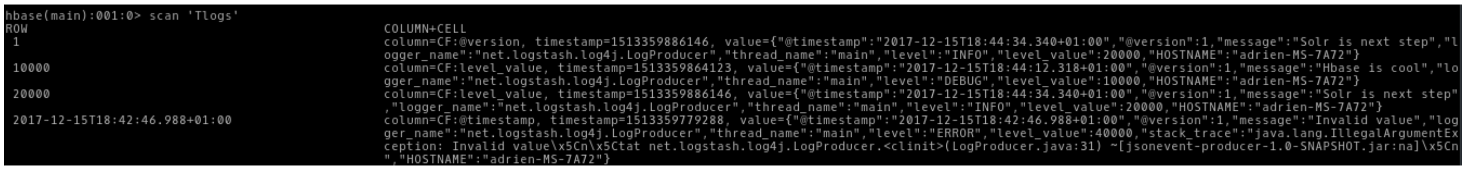
Now we start the Log generator, he sends logs to Nifi through its UDP connection. Messages are available on topic_project, Spark will access them and put them on Hbase.



```
17/12/17 16:24:02 INFO ZooKeeper: Client environment:java.io.tmpdir=/tmp
17/12/17 16:24:02 INFO ZooKeeper: Client environment:java.compiler=<NA>
17/12/17 16:24:02 INFO ZooKeeper: Client environment:os.name=Linux
17/12/17 16:24:02 INFO ZooKeeper: Client environment:os.arch=amd64
17/12/17 16:24:02 INFO ZooKeeper: Client environment:os.version=4.10.0-42-generic
17/12/17 16:24:02 INFO ZooKeeper: Client environment:user.name=adrien
17/12/17 16:24:02 INFO ZooKeeper: Client environment:user.home=/home/adrien
17/12/17 16:24:02 INFO ZooKeeper: Client environment:user.dir=/media/adrien/A48E22A08E226ACE/Linux_Programs/spark-2.2.0-bin-hadoop2.7
17/12/17 16:24:02 INFO ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.PendingWatcher@5bfa6b84
17/12/17 16:24:02 INFO ClientCnxn: Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)
17/12/17 16:24:02 INFO ClientCnxn: Socket connection established to localhost/127.0.0.1:2181, initiating session
17/12/17 16:24:02 INFO ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, sessionid = 0x16064fa2b2a0010, negotiated timeout = 40000
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_10_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_2_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_7_piece0 on 192.168.1.25:38583 in memory (size: 1967.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_4_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_6_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_5_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_3_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_8_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO BlockManagerInfo: Removed broadcast_9_piece0 on 192.168.1.25:38583 in memory (size: 1968.0 B, free: 366.3 MB)
17/12/17 16:24:02 INFO CachedKafkaConsumer: Initial fetch for spark-executor-1 topic_project 0 4184
2017-12-17T16:23:49.429+01:00
1
kafka rocks!
net.logstash.log4j.LogProducer
main
TRACE
5000
adrien-MS-7A72
17/12/17 16:24:02 INFO ConnectionManager$HConnectionImplementation: Closing zookeeper sessionid=0x16064fa2b2a0010
17/12/17 16:24:02 INFO ZooKeeper: Session: 0x16064fa2b2a0010 closed
17/12/17 16:24:02 INFO ClientCnxn: EventThread shut down
17/12/17 16:24:02 INFO Executor: Finished task 0.0 in stage 12.0 (TID 12). 751 bytes result sent to driver
17/12/17 16:24:02 INFO TaskSetManager: Finished task 0.0 in stage 12.0 (TID 12) in 900 ms on localhost (executor driver) (1/1)
17/12/17 16:24:02 INFO TaskSchedulerImpl: Removed TaskSet 12.0, whose tasks have all completed, from pool
17/12/17 16:24:02 INFO DAGScheduler: ResultStage 12 (foreachPartition at VerySimpleStreamingApp.java:77) finished in 0,901 s
17/12/17 16:24:02 INFO DAGScheduler: Job 12 finished: foreachPartition at VerySimpleStreamingApp.java:77, took 0,909007 s
17/12/17 16:24:02 INFO JobScheduler: Finished job streaming job 1513524242000 ms.0 from job set of time 1513524242000 ms
17/12/17 16:24:02 INFO JobScheduler: Total delay: 0,932 s for time 1513524242000 ms (execution: 0,918 s)
```

Figure 23: Spark streaming output when logs are generated

Here spark displays the content of the spitted information that will be put on Hbase on separated columns.We now check if the information has been stored correctly on Hbase by using “hbase shell”:



```
hbase(main):001:0> scan 'Tlogs'
ROW                                          COLUMN+CELL
 1                                           column=CF:@version, timestamp=1513359886146, value={"@timestamp":"2017-12-15T18:44:34.340+01:00","@version":1,"message":"Solr is next step","l
                                             ogger_name":"net.logstash.log4j.LogProducer","thread_name":"main","level":"INFO","level_value":20000,"HOSTNAME":"adrien-MS-7A72"}
 10000                                       column=CF:level_value, timestamp=1513359864123, value={"@timestamp":"2017-12-15T18:44:12.318+01:00","@version":1,"message":"Hbase is cool","lo
                                             gger_name":"net.logstash.log4j.LogProducer","thread_name":"main","level":"DEBUG","level_value":10000,"HOSTNAME":"adrien-MS-7A72"}
 20000                                       column=CF:level_value, timestamp=1513359886146, value={"@timestamp":"2017-12-15T18:44:34.340+01:00","@version":1,"message":"Solr is next step"
                                             ,"logger_name":"net.logstash.log4j.LogProducer","thread_name":"main","level":"INFO","level_value":20000,"HOSTNAME":"adrien-MS-7A72"}
 2017-12-15T18:42:46.988+01:00               column=CF:@timestamp, timestamp=1513359779288, value={"@timestamp":"2017-12-15T18:42:46.988+01:00","@version":1,"message":"Invalid value","log
                                             ger_name":"net.logstash.log4j.LogProducer","thread_name":"main","level":"ERROR","level_value":40000,"stack_trace":"java.lang.IllegalArgumentEx
                                             ception: Invalid value\x5Cn\x5Ctat net.logstash.log4j.LogProducer.<clinit>(LogProducer.java:31) ~[jsonevent-producer-1.0-SNAPSHOT.jar:na]\x5Cn
                                             ","HOSTNAME":"adrien-MS-7A72"}
```

Figure 24: Data stored in Hbase

We perform scan ‘Tlogs’ to see the content of the table we wrote on. We can see that our data has been stored, each row contains a log message with columns corresponding to the different informations.

Conclusion

We found that there's multiple tools available to approach our idea however not all of them are the best choice, we looked for the one's keeping the priorities that First Energy company has systems able to interact with ArcSight(Kafka), security and massive storage performance. We believe that NiFi, Kafka, Spark and HBase are the best solutions for the given dilemma, we are strongly convinced that the use of multiple technologies allows the system to be more reliable and strong.

Coming up with an architecture was our first task, we faced multiple struggles around dependencies, multiple code examples in the subject and it demanded a very sharp eye to see details. It is not an easy task to bring up the system together, but we did our best effort to build it. Sometimes documentation is not very well detailed, and the multiple versions makes the task a bit more tedious, despite the big learning curve ahead of us we learned a lot from this project.

We managed to use Nifi to send logs from all sources to Kafka, ake the data available for any consumer within a topic, use Spark to clean the messages and put them into Hbase. It is now possible to connect to this database to analyze the logs with low latency queries using any appropriate client .

Sources

https://github.com/bbende/jsonevent-producer

https://github.com/dbist/workshops/blob/master/hbase/HBaseJsonLoad/src/main/java/com/hortonworks/hbase/Application.java

https://stackoverflow.com/questions/42558798/spark-to-hbase-writing

https://www.tutorialspoint.com/hbase/hbase_create_data.htm

https://kafka.apache.org/

https://spark.apache.org/docs/latest/streaming-programming-guide.html

https://spark.apache.org/docs/2.2.0/streaming-kafka-0-10-integration.html

https://nifi.apache.org