DataLab - Enterprise IT Digital Twin

Hamza BENGUERRAH, Guillaume PINOT, Matin BAYRAMOV

Table of Contents

1. Presentation of DataLab

The main purpose is to build coherent repositories and map them with logs to simulate IT in a digital way. While these repositories are *theoretical view* of the IT, asset's logs are real *usage view*.

As result, *Referential data* (AIP, MDM-ITC, Storage Master Reports, IDM) and *Usage logs* (Nexthink, Server-sockets, Server usage, Oracle logs) are collected data on which we are mainly focused to do analysis.

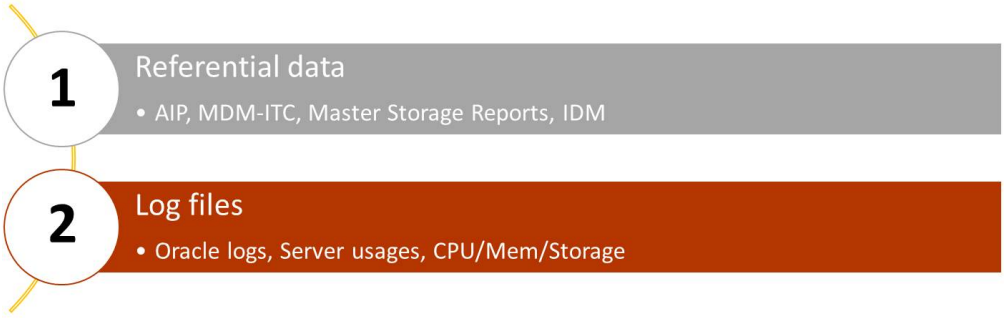


Figure 1. Presentation of two main kinds of data

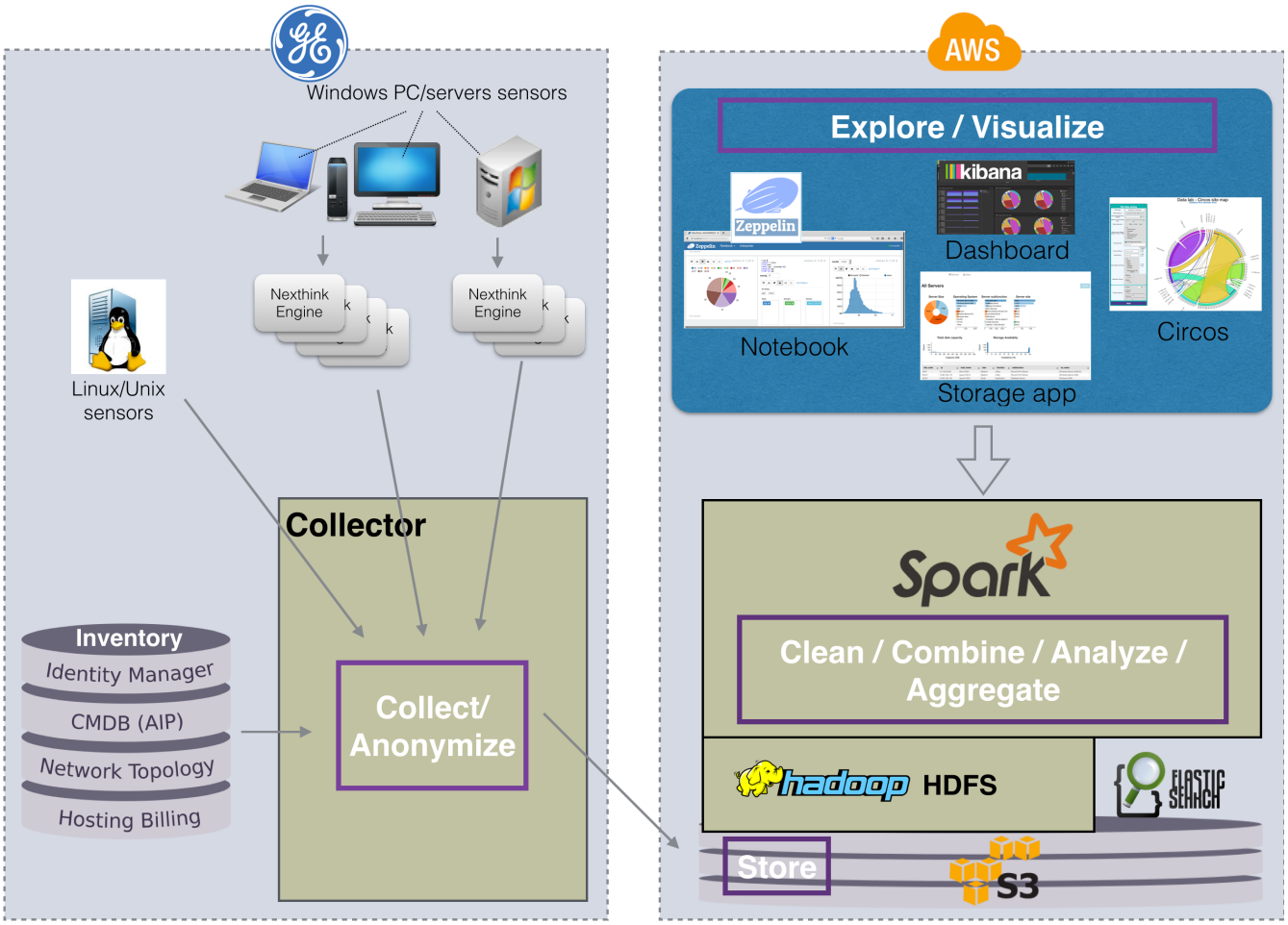


Figure 2. Global view of the project

This figure represents global view of the project. Data collection is shown on the left side. Referential data and logs are collected and anonymized on the Collector server before storing them under s3.

On the right side, we see what are technologies used to store, combine, analyse, aggregate data and how we finally visualize analysis reports via various programs, such az Zeppelin, Circos, etc…

2. Global View

As it is mentioned in the previous chapter, there are different kinds of data which are collected from multiple sources by different ways. We realize some actions to prepare data before being able to do analysis. Therefore, these data manipulations such as extraction from log files, anonymization, etc. are not all applied on all of them. As an example, while sensitive user information data are needed to be anonymized, others are not.

In the next chapters, we will explain one by one how each kind of data are collected, stored and prepared to be analysed.

Before presenting one by one of these collections and their usage analysis, here is an image which shows common processes which are applied during various steps.

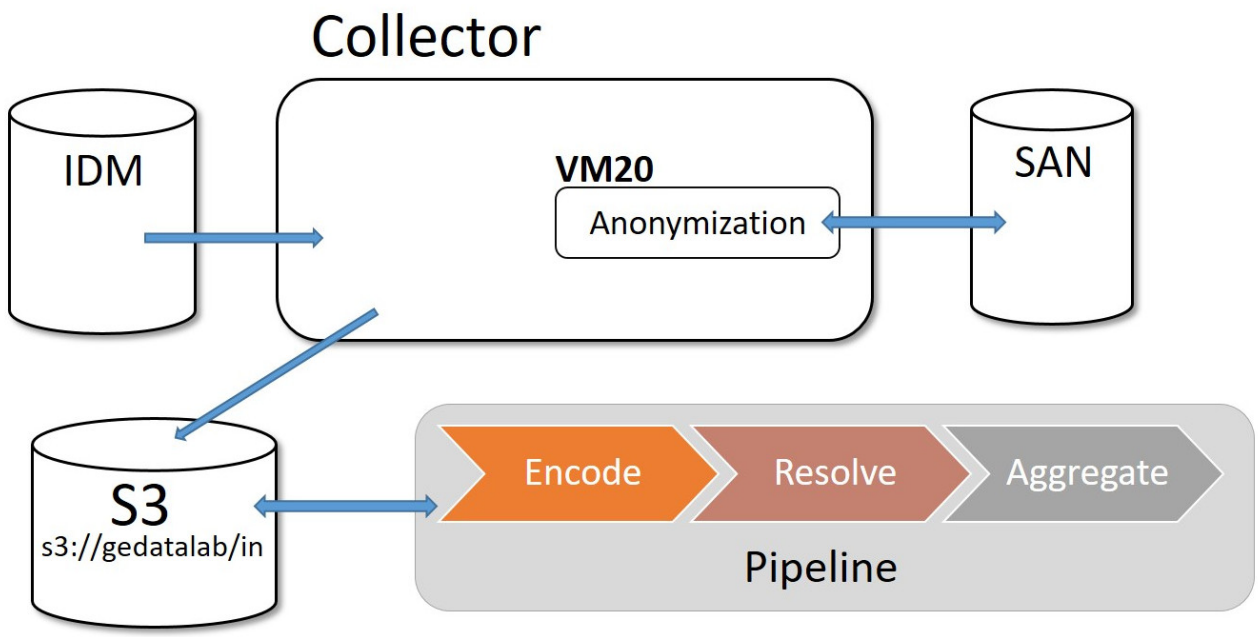


Figure 3. Global view of the common processes

Collected logs are stored in SAN server. They are extracted into CSV format, user information are made anonymous and finally stored on s3.

Generally, the input data arrive to SAN server. They are extracted into correct formats and made anonymous if necessary and stored on s3. When needed they are copied to HDFS and analysed on the pipeline processes. And IDM are used to resolve data if they are made anonymous. In the following chapters we'll give some more detailed information about all of these items and processes.

Collector server (which is named as SARMA10012) collects different kind of data sources which are *nexthink*, *server-usage*, *server-socket*, *oracle-log*. While the way we collect them varies, common actions are performed and same environments are used to process and analyse them.

The main object is to collect all information and put them on the local SAN server. Later, we use VM20 to anonymize these information. And anonymous data are saved under SAN server. Finally it is sent to s3 server.

Collector server have access to IDM data which must be anonymized before sending to s3.

It creates the anonymized IDM which is named as I-ID. This is because we could later be able to know what is the sector, team, site of an anonymized user.

A dictionary file (dictionary.csv) is created under SAN server to keep information about which user data correspond to which I-ID. The collector server is the one which can tell us which anonymized user matches with its real information.

Nexthink, server usage and server socket are then used inside of the process named "Pipeline".

2.1. File/Folder structure organization

Common folder structure strategies are applied on all over the project.

- */in* folder is reserved to keep input data (logs in CSV format, etc.)
- */out* folder is used to store output data after different manipulation such as resolving, aggregating, etc..
- */done* folder is intended to keep data which are already processed. (files already moved/merged, etc.)

As the same way, folder name convention is also common on the source code of the project.

- */bin* folder contains main script files (ex: collect.sh, main.sh, distribute.sh, etc.)
- */conf* folder contains configuration files such as *env.sh* to setup or initialize common variables.

3. References

We are mainly focused on 4 types of referential data collection which are *AIP*, *MDM-ITC*, *Storage Master Report* and *IDM*. They are listed in the following figure with their paths on **s3://gerepo**.

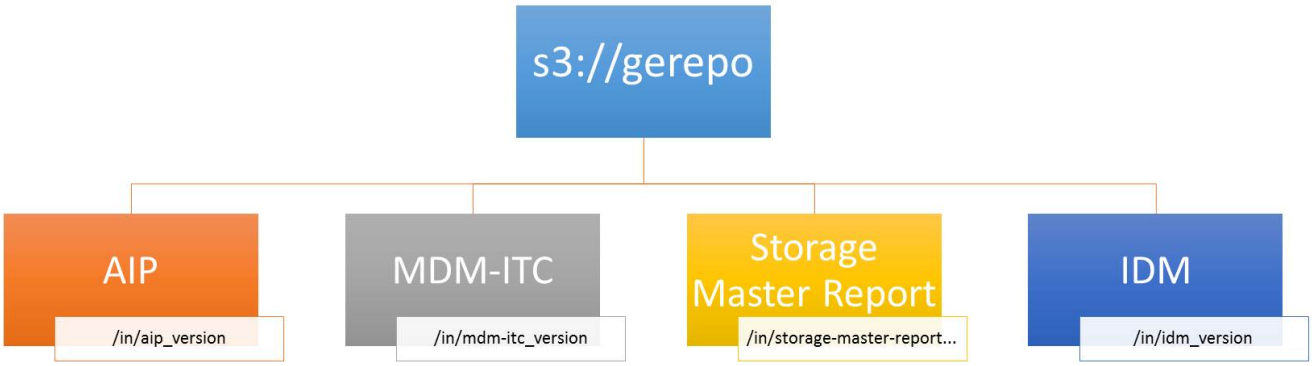


Figure 4. Types of referential data collections

3.1. AIP

AIP (Application Inventory Portal) is data referential for all applications installed on different servers and used by different users. In other words, AIP is a referential of the IT assets of Ge-Alstom and it covers mainly the list of applications, the list of servers on which these applications are deployed, the product software used by these applications, the licenses used by the applications, etc.

There are 3 main tables which are :

- *CartoServer*: list of servers
- *CartoApplication*: list of business applications
- *CartoSoftInstance*: which application is deployed on which server

Daily batches extract AIP table contents in CSV format. They are archived daily and named with datetime and stored under *s3://gerepo/in/aip_version/*.

```
CartoApplication_20161216.csv.gz  CartoServer_20161216.csv.gz  CartoSite_20161216.csv.gz  CartoSoftInstance_20161216.csv.gz
CartoApplication_20161217.csv.gz  CartoServer_20161217.csv.gz  CartoSite_20161217.csv.gz  CartoSoftInstance_20161217.csv.gz
CartoApplication_20161218.csv.gz  CartoServer_20161218.csv.gz  CartoSite_20161218.csv.gz  CartoSoftInstance_20161218.csv.gz
CartoApplication_20161219.csv.gz  CartoServer_20161219.csv.gz  CartoSite_20161219.csv.gz  CartoSoftInstance_20161219.csv.gz
```

Figure 5. List of extracted AIP table contents in s3

Zeppelin notebook URLs are

- https://devzeppelin.gadatalab.com/#/notebook/2BMCK757N
- https://devzeppelin.gadatalab.com/#/notebook/2BXZ39CTF

3.2. MDM-ITC

This is the network topology which gives us any information about network elements. As an

example, we are able to know what are the *IP ranges* for a given *site* thanks to these information. Today, these data are manually transferred into s3://gerepo/in/mdm-itc.

3.3. Storage Master Report

Basically, this report is a single XLS file. For instance, it is manually extracted, updated and stored under s3. It needs to be converted to a CSV file and placed under s3://gerepo/in/storage-master-report_version.

This is a billing file/report for CSC which gives information about server memory usages. It is updated once per month.

It contains also some information about disk storage space used for each server and its instances, kinds of storage, total amount of memory space allocated, used, etc.

Zeppelin notebook URLs are

- https://devzeppelin.gadatalab.com/#/notebook/2BJVRKXMT
- https://devzeppelin.gadatalab.com/#/notebook/2BPKDDK7S

3.4. IDM

IDM is the management of individual identities/users, their authentication, authorization, roles and privileges within different sites, sectors, teams etc.

In short, *IDM* is the list of users with their corresponding sectors, sites, teams etc.

IDM is used to resolve user information which are made anonymous before being stored on s3. However, we don't need IDM to anonymize data.

4. Types of data sources to collect

There are different kind of sources that we collect to analyse. These sources are :

- Oracle logs
- Nexthink
- Server sockets
- Server usage

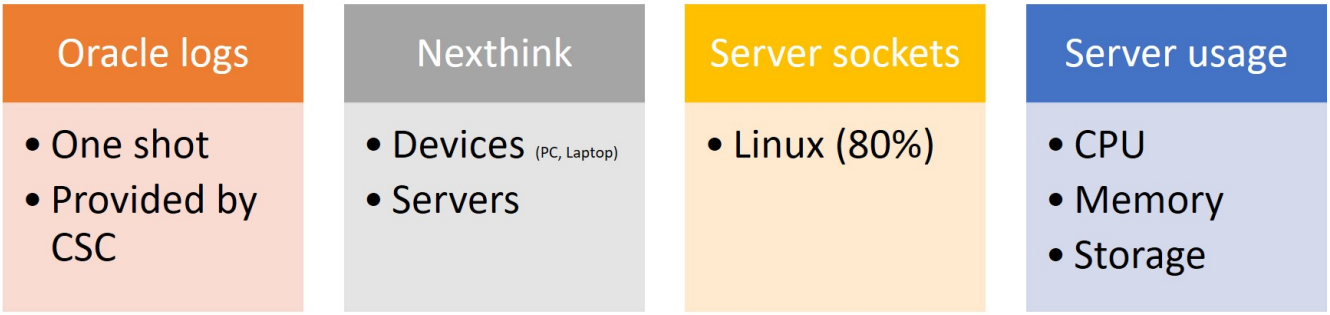


Figure 6. Types of data sources to collect

Oracle logs are database access log files. *Nexthink* is the application which collects data about network connections, program executions, web requests, etc. from PCs, laptops or servers. While Nexthink is mainly applied on Microsoft Windows systems, *Server sockets* is implemented for Linux based machines. Finally, *Server usage* is data collection from servers, including their CPU, memory, storage usages, etc.

5. Nexthink

5.1. What is Nexthink ?

Nexthink is the application which collects information about any actions done on a PC, Laptop or a server. See image below. [1: https://doc.nexthink.com/images/a/a3/Collector.png]

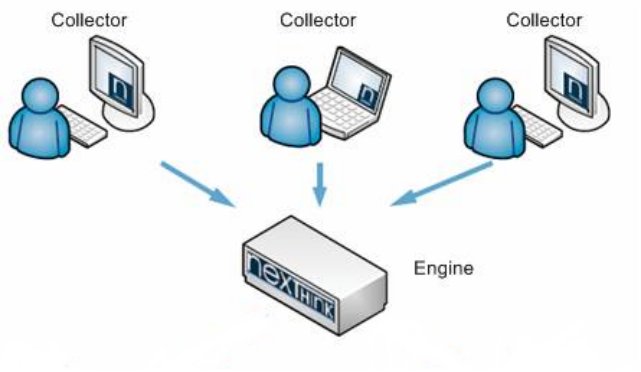


Figure 7. Nexthink Engine which collects data from different devices

A Nexthink *Collector* program installed on different devices captures network connections, program executions, web requests, etc.. and sends data to Nexthink *Engines*. Nexthink *Engines* stock received data and make daily backups.

5.2. Where are data collected on Nexthink side ?

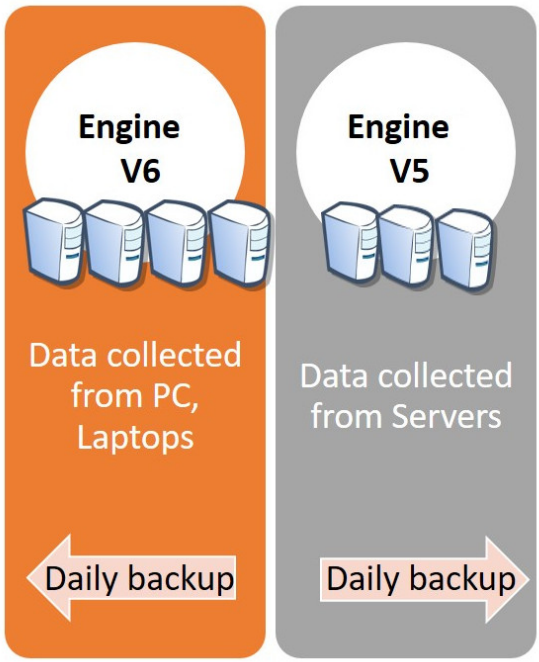


Figure 8. Nexthink Engines with backup servers

There are two groups of Nexthink servers on which collected data are stored.

- *Data collected from PCs, laptops, etc.* There are 21 servers and the version of Nexthink installed is V5.

- *Data collected from Servers*. There are 3 servers in this group. The version of Nexthink is V6.

As a result, data storing and backup processes are different on both kind of engines/servers.

Furthermore, archived data aren't kept for a life on backup servers. As soon as more data are coming, the very old once are erased.

5.3. List of servers

The following CSV files contain the list of hostnames of the nexthink backup servers: *engine_amer.csv*, *engine_asie.csv*, *engine_eur1.csv*, *engine_eur2.csv*, *engine_server.csv*. They are located under */collect/nexthink/0/bin/* folder.

collect/nexthink/0/bin/engine_eur1.csv

```
sabad11478.ad.sys
sabad11479.ad.sys
sabad11480.ad.sys
sabad15419.ad.sys
sabad11481.ad.sys
```

5.4. How collected data are transferred on our side ?

There is a single archive file per nexthink backup server. Each file can contain log history of 5 to 20 days. Here is a picture of list of archived files copied from nexthink backup servers.

```
-rw-r----- 1 nexthink nexthink 2260234495 Dec  1 04:23 sacch15562.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2408645966 Dec  1 04:24 sacch15564.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2297430917 Dec  1 04:24 sabad11480.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2122612713 Dec  1 07:20 susao10145.dom3.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2290572022 Dec  1 07:21 sasao10310.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2253058998 Dec  1 10:20 suwnd10005.dom3.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2261425744 Dec  1 10:20 suwnd10004.dom3.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2159519762 Dec  1 10:22 suwnd15009.dom3.ad.sys_20161201.tgz
-rw-r----- 1 nexthink nexthink 2217723964 Dec  1 21:23 sushah10001.ad.sys_20161202.tgz
[nexthink@sarma10012 0-NXProdBackups]$
```

Figure 9. Archive files on the collecter server copied from nexthink backup servers

We use *scp_engine.sh* script to copy archived data from a nexthink backup server to the collector (SARMA10012) server. This can be done manually per hostname by : *./scp_engine "hostname"*

collect/nexthink/0/bin/scp_engine.sh

```
DumpDir=/datalab3/DATA/0-NXProdBackups
PathFile=/var/nexthink/engine/01/backups/nxengine-backup.tgz
ServerName=$1

DATE=$(date +"%Y%m%d")
HEURE=$(date +"%H%M%S")
cd $DumpDir
echo "$DATE $HEURE scp_engine.sh : Debut de la copie $ServerName..."
scp -pB -o "StrictHostKeyChecking no" datalab@${ServerName}:${PathFile} ./${ServerName}_${DATE}.tgz.tmp
mv ./${ServerName}_${DATE}.tgz.tmp ./${ServerName}_${DATE}.tgz

DATE=$(date +"%Y%m%d")
HEURE=$(date +"%H%M%S")
echo "$DATE $HEURE scp_engine.sh : Fin de la copie $ServerName"
```

However, *dump_all_engine.sh* script is written to execute copy process for a all of servers in a CSV file.

collect/nexthink/0/bin/dump_all_engine.sh

```
    exit 1
fi

for file in $(cat $lst_engine)
do
    sh $Dir/scp_engine.sh $file&
```

Finally, a cron table definition is implemented in order to run data copy in about every 3 days.

collect/nexthink/0/crontab/my_cron

```
30 16 * * 0,3 /datalab2/home/nexthink/cp_engine/dump_all_engine.sh engine_eur1.csv
30 18 * * 0,3 /datalab2/home/nexthink/cp_engine/dump_all_engine.sh engine_eur2.csv
30 14 * * 0,3 /datalab2/home/nexthink/cp_engine/dump_all_engine.sh engine_asie.csv
30 20 * * 0,3 /datalab2/home/nexthink/cp_engine/dump_all_engine.sh engine_amer.csv
30 16 * * * /datalab2/home/nexthink/cp_engine/dump_all_engine.sh engine_server.csv
```

As we try to collect these data every 3 days in order to not loose any part of them, this causes us some overlaps. This is why we should make attention to filter duplicated data during analysis processes.

5.4.1. How input data are used ?

Provided data stored under SAN server in **/0/ folder** are considered as *input data*. As they are collected by Nexthink way, they are not in any columnar format (CSV, parquet, etc.). This is why the very first step is to extract these data into CSV format with the help of a Nexthink environment. We have 2 virtual machines (named as VM5 and VM6) on the Collector server which are dedicated to this purpose. VM6 is used to extract data collected from servers and VM5 for the rest.

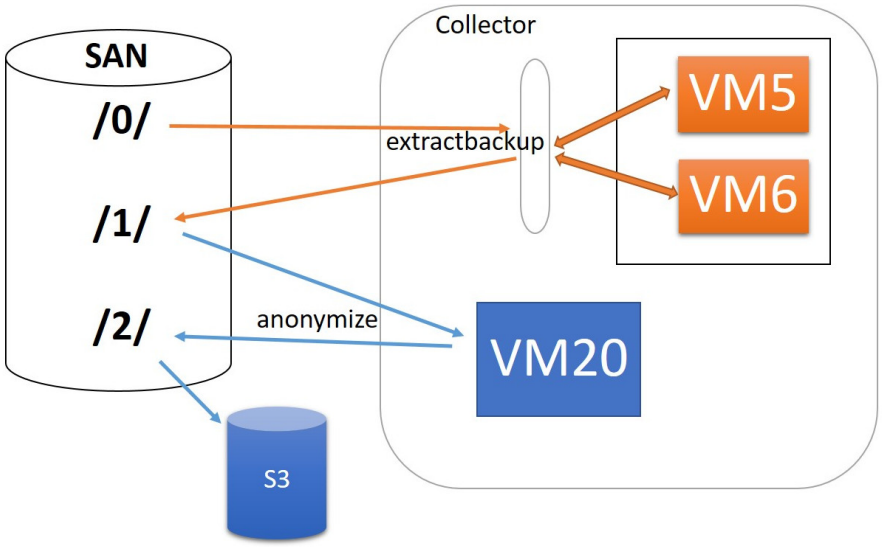


Figure 10. Data extraction and anonymization

5.4.2. Extract-backup

Input data are deployed either on VM5 or VM6 to be extracted. We use NXQL sql requests to extract information and we store them in CSV formats. This operation is executed daily and we call it as *extractbackup*.

Extracted data are copied on **/1/ folder**. At this stage, they are ready to be anonymized.

After extracting, we separate 3 data types under /1/ folder.

- *connection* - anything related to users' "connection" (TCP, UDP, etc..)

> A connection is a link between a device and a destination through the use of network resources. There are two types of connections depending on the transport protocol used for communication: *TCP*: The connection has a status. *UDP*: The connection is stateless.

Information collected about any outgoing (and only outgoing) network requests, such as which user is connected, by which application, IP address requested, HTTP protocol used, server port number, request execution time, request content size, etc. These information are mainly related to the source device of the requests. It can be a simple user machine but also a server.

- *webrequest* - anything related to a "webrequest" (DNS information)

> A web request is a message sent from a client application to a server using the standard web protocol HTTP or its secure version over TLS. Nexthink is able to record not only the web requests of web browsers, but of any application that runs on the device of the end-user.

This kind of data are captured while a web request (HTTP) is detected. Some information about the target device is collected, such as request's DNS address (google, etc.), etc. However, full URL of web requests are not registered at all.

- *execution* - anything related to an application "execution"

An *execution* is an activity that indicates the loading of a binary into the memory of a computer to run it as a separate process.

These are information about the execution of any application used by users. This concern also applications which do not access to internet. (even if a user doesn't login to the application).

These kind of data give us information about which application is executed, by which user, at what time, the version of the application, how much does it take to be started, the path to the application, etc.

As a result, there are 3 main folders (*/connection*, */webrequest*, */execution*) under s3 server for each of these types of data collections.

5.5. Anonymization

The next step in the process is to make anonymous some user information from extracted data. As usual, we use virtual machine VM20 for this purpose. Once done, we store them under /2/ folder on the SAN server in CSV format.

Finally, these data are copied to *s3://gedatalab/in*.

Source code path: ALDatalab/collect/nexthink.

6. Server sockets

This simple project is implemented to collect data from linux-based machines as nexthink works only on windows based machines. We look for information about what is executed or accessed and what are processes invovled by these accesses.

Source code path: ALDatalab/collect/serversocket

6.1. About

A datalab user account is created on about one thousand linux servers. A pair public/private key generated for each of them and public keys are deployed on these servers. Then we connect to the servers with ssh and deploy *monitor.sh* script on them. This script executes linux commands such as "netstat, ps, lsof, etc." every 5 minutes and puts results into server-usage data files. It is activated with a crontab.

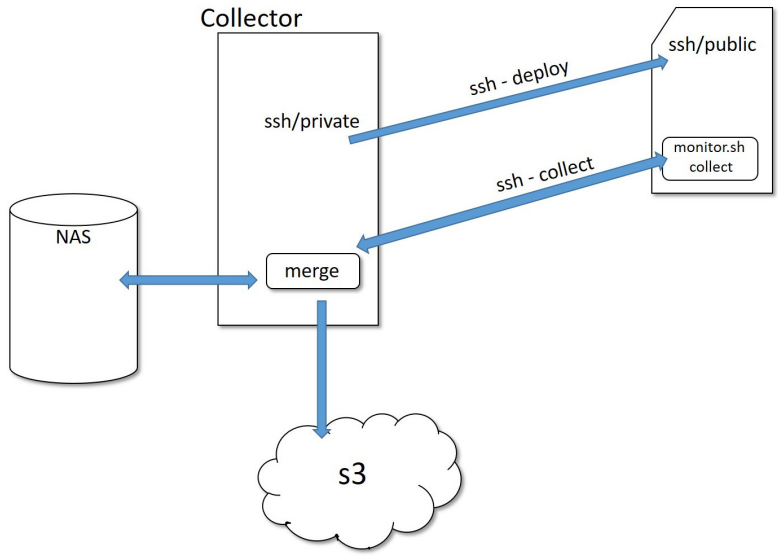


Figure 11. Collecting and deployement of Server sockets data

6.1.1. Config parameters

Configuration parameters, paths to the files, directories are defined in the conf file.

collect/serversocket/conf/conf.sh

```
#!/usr/bin/env bash

ROOTDIR=/datalab2/home/datalab/ALDataLab/collect/serversocket
YES=$ROOTDIR/bin/yes.sh
SCRIPT_SERVER=$ROOTDIR/bin/monitor.sh
```

6.1.2. Script actions

We are able to test server connections, user roles, etc. We can deploy/undeploy the script, execute collect function to collect server usage data.

collect/serversocket/bin/submit.sh

```
case $method in
  test)
    test $serverIP $host $ServerType
    ;;
  deploy)
    deploy $serverIP $host $ServerType
    synchro $serverIP $host $ServerType
    ;;
  update)
    update $serverIP $host $ServerType
    synchro $serverIP $host $ServerType
    ;;
  undeploy)
    undeploy $serverIP $host $ServerType
    ;;
  collect)
    collect $serverIP $host $ServerType
    synchro $serverIP $host $ServerType
    ;;
  synchro)
    synchro $serverIP $host $ServerType
    ;;
esac
exit $?
```

6.1.3. Deploy

collect/serversocket/bin/submit.sh

```
function deploy() {
    #deploy server script on datalab@$SERVER:/var/tmp/script and create corresponding crontab
    SERVERIP=$1
    HOST=$2
    OSTYPE=$3
    [[ "$HOST" == "" ]] && HOST=$SERVERIP
    RET_DEPLOY=1
    test $SERVERIP $HOST $OSTYPE &&\
     TMP_SCRIPT=$(mktemp) &&\
     cat $SCRIPT_SERVER | sed -e "s/^HOSTNAME=\".*\"$/HOSTNAME=\"${HOST}\"/" | sed -e "s/OS_TYPE/${OSTYPE}/" >
$TMP_SCRIPT &&\
     scp $TMP_SCRIPT datalab@$SERVERIP:~/$(basename $SCRIPT_SERVER) &&\
     rm $TMP_SCRIPT &&\
     ssh -o ConnectTimeout=10  -o "BatchMode=yes" -o StrictHostKeyChecking=no datalab@$SERVERIP "chmod +x ~/$(basename
$SCRIPT_SERVER) && echo \"*/5 * * * * ~/$(basename $SCRIPT_SERVER) monitor 2>~/monitor.err 1>~/monitor.out\" >> mycron &&
crontab mycron" &&\
     RET_DEPLOY=0
    return $RET_DEPLOY
}
```

6.1.4. List of servers

collect/serversocket/conf/ServerListToCollect.csv

```
Hostname;ServerFunction;Type;SubFunction;IPServer;InSID;ServerStatus;SiteCode;AdministratedBy;OSName;ToCollectInit;ToColl
ectLast
abp-vpdm;Application;Physical;Not applicable;10.99.142.16;Yes;Active;BJS2;CSC;HP-UX;true;true
null;Utility;null;null;10.178.104.52;null;Active;null;null;null;false;false
null;Application;null;null;10.165.16.17;null;Active;null;null;null;false;false
null;Application;null;null;10.162.135.24;null;Active;null;null;null;false;false
...
```

6.1.5. Monitor

collect/serversocket/bin/monitor.sh

```
function monitor_linux() {
    ret=0
    retcmd=1
    sudo lsof -nPi | sed 1d | \
    awk -vdat=${datMM} -vdeltatime=$TIMEDELTA -vserver=$HOSTNAME -vos=$OSTYPE \
    '$8 ~ /UDP|TCP/ {print
"\""os"\";\""server"\";\""dat"\";\""timedelta"\";\""$1"\";\""$2"\";\""$3"\";\""$8"\";\""$9"\";\""$10"\""}\
     $7 ~ /UDP|TCP/ {print
"\""os"\";\""server"\";\""dat"\";\""timedelta"\";\""$1"\";\""$2"\";\""$3"\";\""$7"\";\""$8"\";\""$9"\""}' | gzip -c >>
$LSOF_MONITOR && retcmd=0
    [[ retcmd -ne 0 ]] && ret=$(($ret+1))

    retcmd=1
    ps -ef | sed 1d | \
    perl -pe "s/([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^
].*)/\"${OSTYPE}\";\"${HOSTNAME}\";\"${datMM}\";\"${TIMEDELTA}\";\"\1\";\"\2\";\"\3\";\"\7\";\"\8\"/" |\
    gzip -c >> $PS_MONITOR && retcmd=0
    [[ retcmd -ne 0 ]] && ret=$(($ret+1))

    retcmd=1
    sudo netstat --ip -anp | sed 1d | \
    awk -vdat=${datMM} -vdeltatime=$TIMEDELTA -vserver=$HOSTNAME -vos=$OSTYPE \
    '$1 ~  /udp|tcp|Udp|Tcp|UDP|TCP/ && $7 != "" {print
"\""os"\";\""server"\";\""dat"\";\""deltatime"\";\""$1"\";\""$4"\";\""$5"\";\""$6"\";\""$7"\""}' | \
    gzip -c >> $NETSTAT_MONITOR && retcmd=0
    [[ retcmd -ne 0 ]] && ret=$(($ret+1))
    return $ret
}
```

6.1.6. Collect

collect/serversocket/bin/submit.sh

```
function collect() {
    SERVERIP=$1
    HOST=$2
    [[ "$HOST" == "" ]] && HOST=$SERVERIP
    DATECUR=$(date --utc --date "now" +"%Y%m%d-%H%M%S")
    RET_COLLECT=1
    #collect data of server
    ssh -o ConnectTimeout=10  -o "BatchMode=yes" -o StrictHostKeyChecking=no datalab@$SERVERIP "~/$(basename
$SCRIPT_SERVER) collect" &&\
     scp datalab@$SERVERIP:~/collect/*.gz $DIR_COLLECT/. &&\
     ssh -o ConnectTimeout=10  -o "BatchMode=yes" -o StrictHostKeyChecking=no datalab@$SERVERIP "rm -f ~/collect/*.gz"
&&\
     RET_COLLECT=0
    echo "$HOST;$SERVERIP;$RET_COLLECT;$DATECUR" >> $SERVERCOLLECT
    echo "$HOST;$SERVERIP;$RET_COLLECT;$DATECUR"
    return $RET_COLLECT
}
```

6.2. Collecting data

Collector server gets files from servers and put them on the NAS server. As there are so many small files, we merge them into larger files before putting on s3://gedatalab/in/serverusage.

6.3. Anonymization

Like *Server usage* data there is no anonymization process for *Server sockets*.

7. Server usage

Another data sources that we collect to analyse is *Server Usage*. It involves the information about server CPU usage, memory and storage usage.

- Source code path : ALDataLab/collect/serverconsumption.

7.1. Method

In order to collect a server's usage information we need to request an HTTP URL by specifying its hostname as a parameter. This URL and other configuration parameters are defined in conf.sh file as below.

collect/serverconsumption/conf/conf.sh

```
SERVERLST=$ROOTDIR/conf/server.lst
URL="http://iww.dcs.itssc.alstom.com/nrtmd/streamsdump/server"
```

7.2. List of servers

List of servers are defined in *server.lst* file. Each line represents a hostname of a server. They are used by collect script.

collect/serverconsumption/conf/server.lst

```
SARMA15601
SARMA15605
SARMA15602
SARMA15604
SARMA15603
...
```

7.3. Collect

As we know how to fetch *Server usage* for a single one, we can handle this for all others too. Request answer for each server is a XLS file. Later, we will convert these files into CSV formats.

collect/serverconsumption/bin/collect.sh

```
for SERVER in $(cat $SERVERLST |awk '{print $1}')
do
    CMD="curl $URL/$SERVER  --compressed"
    echo "$(date +"%Y/%m/%d-%H:%M:%S") : $0 : CMD : $CMD > $dirserverxls/${SERVER}_$suffix.xls"
    $CMD > $dirserverxls/${SERVER}_$suffix.xls
done
```

7.4. Convert XLS to CSV

Once we collected all XLS files we convert them to CSV format. Finally, we compress them into .gz zip format.

collect/serverconsumption/bin/convert-send.sh

```
#compute xls files
function compute() {
    ficout=$dirserverjson/serverusage_$(date +"%Y%m%d-%H%M%S.%3N").json
    $convert_py -o $ficout $* &&\
    for fic in $*
    do
        mv $fic ${dirdone}/.
    done &&\
    gzip $ficout && scp ${ficout}.gz $dirnas &&\
    mv ${ficout}.gz ${dirserverjsonsent}
}
```

There is nothing to make anonymous for these kind of data collection.

Server-usage data are synchronized 2 times in a week.

8. Pipeline

There are 3 main processes that we need to apply. These are *encoding* of collected data from CSV format into "parquet" format, *resolving* them to get complete data and *aggregating* to get some analysis results.

- *Enconding* :

Collected data are in CSV file format and stored in s3://gedatalab/. We need to encode them in "parquet" format in HDFS file system. This will improve data manipulation performance compared to CSV format.

- *Resolving*:

This is the resolution of encoded data by combination with other referential information. As an example, we can re-define a user's sector, site or its team name with the help of resolve methods. This process is intended to find specific information from multiple tables. We can find for example *the number of connections, the number of distinct users, the number of distinct devices, the network traffic volume*, etc..

- *Aggregation*:

This is simply aggregation of data, which may be done daily, monthly, etc.

The input data are stored in s3://gedatalab/. They are then deployed to HDFS file system, encoded, resolved and aggregated. Data output from each of these steps are kept and stored on s3, so we can use any of them at any time.

There are 3 main folders which are /*encoded*, /*resolved*, and /*aggregated*. We store data in these folders after each main step. Each of these folders contains three sub-folders for different types of data (nexthink, server-usage, server-socket). Once we finished data manipulation we store them again on the s3 server for future use.

We execute these processes in a Pipeline. It is execution of various batches on a cluster.

8.1. Pipeline folder structure

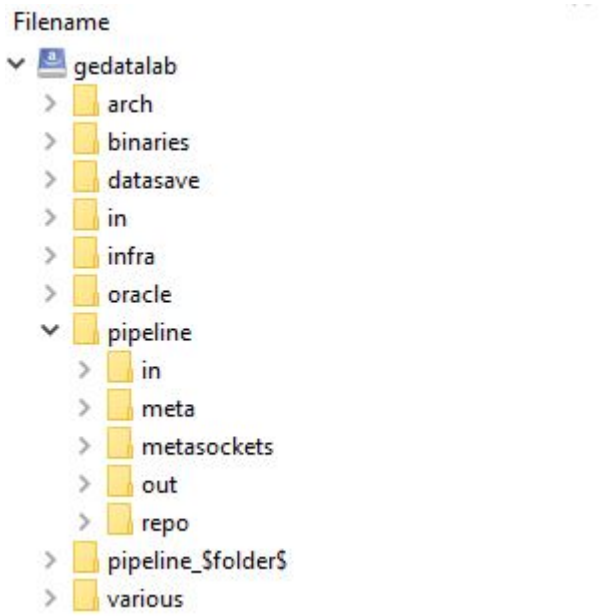


Figure 12. Pipeline folder structure under s3 server.

We organized folder structure of pipeline process as below:

- */gedatalab/in* - contains newly stored data coming from collector server.
- */gedatalab/pipeline* - is the main folder to keep all kind of data used during different steps of pipeline actions.
- */gedatalab/pipeline/in* - will be used to keep default input data for pipeline.
- */gedatalab/pipeline/done* - folder is used to keep output data.
- */gedatalab/pipeline/meta* - is dedicated to store meta-information about *nexthink* data.
- */gedatalab/pipeline/metasocket* - folder is used to store server socket meta-data.
- */gedatalab/pipeline/repo* - folder is used to achieve "resolve" processes. I-ID are stored under this folder.

8.2. Pipeline main actions

collect/oracle/bin/main.sh

```
echo "$(date +"%Y/%m/%d-%H:%M:%S") - $0 : Begin"
$HOME/pipeline/bin/syncHdfsS3.sh fromS3 &&\
$HOME/pipeline/bin/repo.sh &&\
$HOME/pipeline/bin/genAIP.sh &&\
$HOME/pipeline/bin/pipe2to3.sh &&\
$HOME/pipeline/bin/pipe2to3_execution.sh &&\
$HOME/pipeline/bin/pipe3to4.sh -d $begindate $enddate 10 &&\
$HOME/pipeline/bin/pipe3to4Exec.sh -d $begindate $enddate 5 &&\
$HOME/pipeline/bin/pipe4to5.sh -d $begindate $enddate 10 &&\
$HOME/pipeline/bin/webapp.sh &&\
$HOME/pipeline/bin/encodeserversockets.sh &&\
$HOME/pipeline/bin/resolveserversockets.sh -d $begindate $enddate 2 &&\
$HOME/pipeline/bin/aggregateserversockets.sh &&\
$HOME/pipeline/bin/serverusage.sh &&\
$HOME/pipeline/bin/flow.sh &&\
$HOME/pipeline/bin/syncHdfsS3.sh toS3; ret=$?
```

1. Make an archive of the s3://gedatalab/pipeline to s3://gedatalab/arch.
2. Move data from s3://gedatalab/in to s3://gedatalab/pipeline/in
3. Make a copy of data from s3://gedatalab/pipeline to hdfs://data/.
4. Data in HDFS is encoded, resolved and aggregated.
5. Replace (or erase) s3://gedatalab/pipeline with data in HDFS, hdfs://data/.

We need to deploy data on a distributed file system in cluster if we want to analyse them. This is why we use s3 to keep data, but hdfs://data (on Amazon EC2) to manipulate or do analysis on them. While working on Zeppelin notebooks, it is also possible that the data are already deployed on HDFS file system on a previous Zeppelin paragraph. In this case, we can directly use them. Otherwise, we need to deploy data from s3 to HDFS.

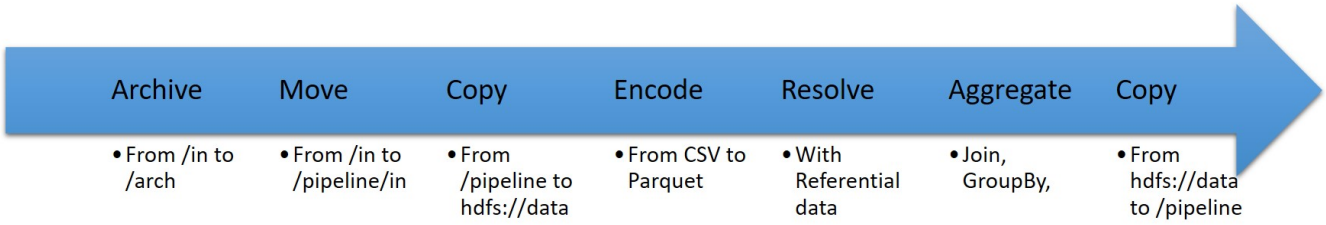


Figure 13. Order of main actions taken place in Pipeline

Note that it is also possible to access on the output data of each state realised in the Pipeline. As an example, resolved data are stored on the *resolved* folder which can be accessed and used at any time.

Pipeline source code is under */src/main/scala/* folder in project.

8.3. Compared to Oracle log file analysis

While we analyse Oracle log files we don’t use Pipeline processes. However, almost all of these processes are also applied to Oracle log file analysis. But all are done on Zeppelin notebooks.

9. Oracle Logs

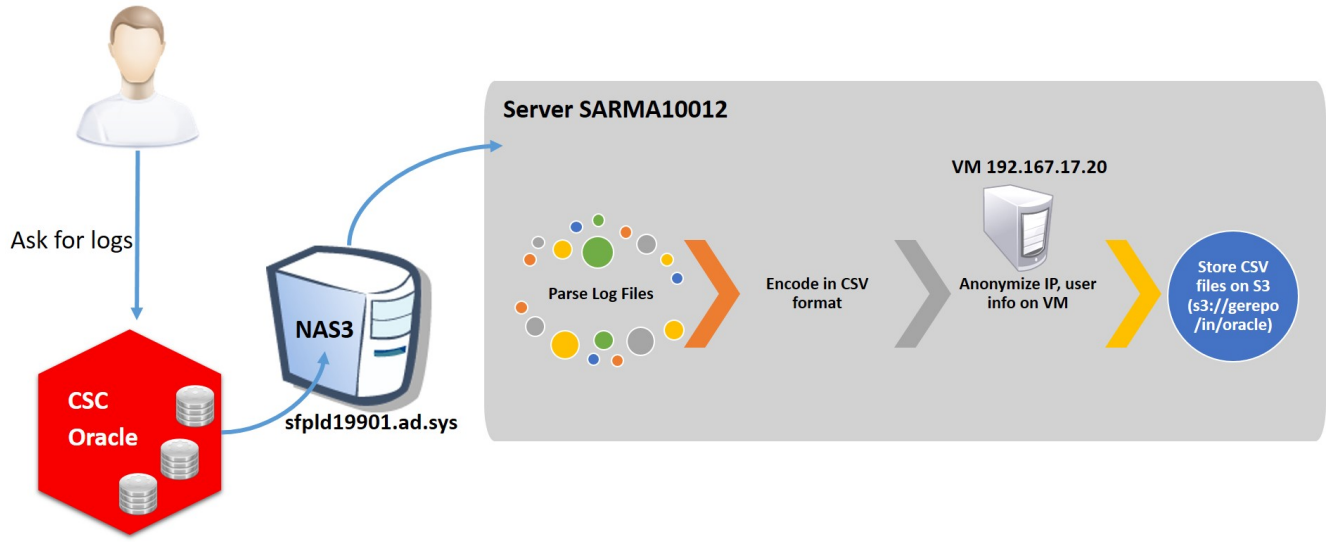


Figure 1: Global view of the processes

9.1. DB access log files

Our goal is to analyse accesses to the specific servers or server instances. This is why we ask to CSC to provide us Oracle DB log files by giving them list of server names, instance names in CSV format. This is a link to one of these files which are available under s3 server.

s3://gecustomers/document/GPI/oracle_log/Datalab_master_file_Oracle_Log_Request_with_SID_20160907_V2.6.csv

The provided log files (which are recorded by different listeners) are stored under NAS3 server. (sfpld19901.ad.sys)

Log file of each server instances named with its server name and instance name. Here are some examples of log files provided. Ex: listener1_sabad19305_IM1.zip, listener_acch15624_MW2.zip.

These log files give us many information about the source of Data Base requests, the kind of source devices such as physical user machine, proxy server, etc.

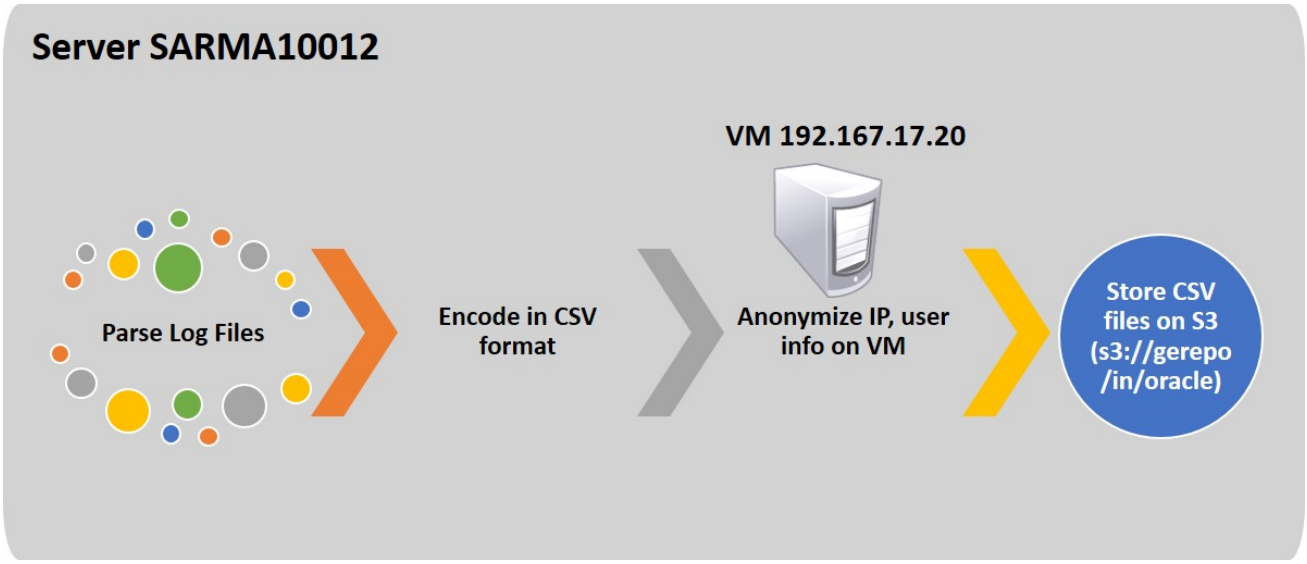


Figure 2: Processus on the SARMA10012 server

9.2. Server SARMA10012

9.2.1. Transferring files from remote server

In order to process log files we need to copy them to SARMA10012 server.

The function *get_from_remote()* in *collect/oracle/bin/main.sh* file is written for this purpose. Archived .zip files are securely transferred from NAS3 server to the local collector server.

collect/oracle/bin/main.sh

```
################################################################
function get_from_remote() {
    #move remote files to local
    for fic in $(ssh $REMOTE_ORACLE_LOG_USER@$REMOTE_ORACLE_LOG_SERVER "cd $REMOTE_ORACLE_LOG_DIR_IN; ls *.zip
2>/dev/null")
    do
        echo "fic : $fic"
        scp -p $REMOTE_ORACLE_LOG_USER@$REMOTE_ORACLE_LOG_SERVER:$REMOTE_ORACLE_LOG_DIR_IN/$fic $LOCAL_ORACLE_LOG_DIR_IN
&&\
        ssh $REMOTE_ORACLE_LOG_USER@$REMOTE_ORACLE_LOG_SERVER "mv $REMOTE_ORACLE_LOG_DIR_IN/$fic
$REMOTE_ORACLE_LOG_DIR_DONE/$fic"
    done
}

################################################################
```

Later, these archived (.zip) log files will be parsed via parse_in() method.

9.2.2. Encode log files in CSV format

As it is known, log files are not in columnar format. Therefore, we need to transform them into columnar format, like CSV.

This process is done inside of the *parse_in()* function in *collect/oracle/bin/main.sh* file. Lines are read one by one and stored inside of a .csv.gz file. And these archive files are sent to S3 server.

9.2.3. Anonymization

Log files contain user access information and host name of the source devices. In order to preserve anonymity we need to anonymize these information before storing them in S3 server. This process is done in VM ubuntu with IP 192.167.17.20. Fields source.user and source.host.name anonymized into I_ID_U and I_ID_D.

Results are exported in .csv.gz format and sent to the S3 server under s3://gerepo/in/oracle repository.

A single Virtual Machine is dedicated to anonymize the received data. Any user information, device IDs, machine IPs are anonymized.

9.2.4. Virtual Machine - 192.167.17.20

Access to the virtual machine which realise this operation is permitted by ssh connection. {ssh datalab@192.167.17.20}

We have implemented *start_vm()* and *poweroff_vm()* functions in *vbox_vm.sh* script to start and stop the VM. Any information about this VM is in *vbox_vm.csv* file.

9.3. Data manipulation - Oracle pipeline

We use a zeppelin notebook to analyse prepared data. URL to the zeppelin notebook **/in/40 - Oracle pipeline** is https://devzeppelin.gadatalab.com/#/notebook/2BWM6SWE5

This notebook contains multiple paragraphs and each of them dedicated to a specific action.

We parse log files which are stored in CSV format in s3://gerepo/in/oracle and encode them in *parquet* columnar format to s3://gerepo/out/oracle. The goal is to run various actions, such as search, filter, join, etc. much more rapidly than it is possible in CSV format.

9.4. Date time format correction

As experience, we noticed that some columns' information in these log files don’t have the same format. As the requests are received from different time zones, log files contain various date time formats. This is an issue because this will not give us correct search or filter results. In order to resolve this problem, we try either to convert or exclude them.

9.5. Resolves (IP, Mdm-Itc)

In this part, we try to resolve source IP address of the registered flux. Because we anonymized some important information before analysis. As an example, it is important now to find out the source mdm-itc "site" of the requests.

However, some site, sector and teranga information is not always correctly reached as the requests are not only done from physical users machines but also from servers.

We also defined a resolve function to find a site name from an IP address.

Finally, results are written to HDFS file system under hdfs://data/temp/oa_oracle_join

9.6. Date time interval precision

Our goal is to analyze line of access logs which are recorded during a specific date interval. This is why we ask to provider units to give us log files for those intervals. However, we notice that archived log files contain logs which are out of the date interval. Corrupted data will not perform a good analysis results. This is why we should ensure the percentage of the lines which contain corrupted date time formats. And then we need to either correct them or exclude from the analysis.

10. Data Consumer Applications

10.1. Applications

In Datalab project we use some web applications in order to consume collected or prepared data. These are *Apache Zeppelin* , *Circos*, *Kibana*, *Application Savings Calculator*, etc...

Figure 14. Datalab - Web applications

Once collected data from different sources are prepared and ready, we use these web applications to consume them. Mainly, they allow us to visualize analysis results for various purposes.

10.2. Zeppelin

As it is said on its official web site, "*Apache Zeppelin is a web-based notebook that enables interactive data analytics*". It is possible to make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more. Please see the figure below for different types of paragraph created under a zeppelin notebook. We are able to explore data analysis results in various ways, graphics, diagrams, etc.

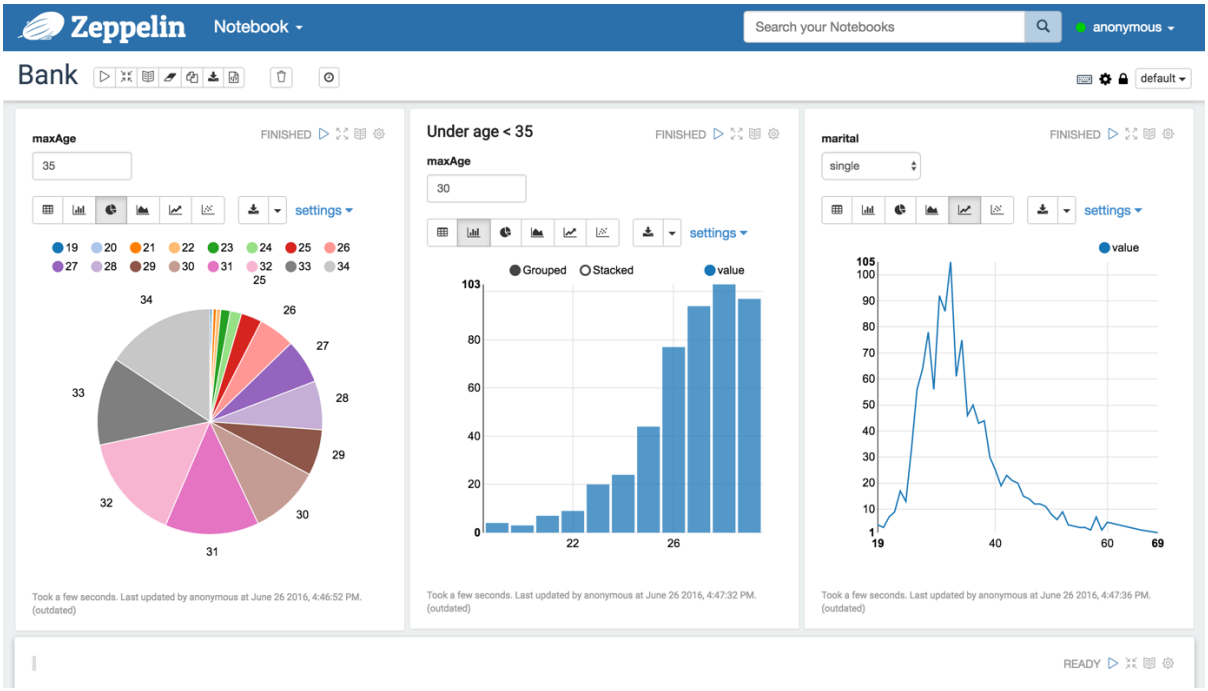


Figure 15. Zeppelin notebook

There are two Zeppelin environments on Datalab project. Main analysis are done on the devZeppelin. Data results are then used on some prodZeppelin notebooks.

Environment	**URL to access**
Development	devzeppelin.gadatalab.com.
Production	zeppelin.gadatalab.com.

10.3. Circos

Circos is one of the Datalab own applications. Its main advantage is to display the data flow between devices or servers. A detailed user interface allow to refine the result by filtering by date, sites, sector, country, etc... The following screenshot from Circos application shows how data can be filtered before visual graphics are generated.

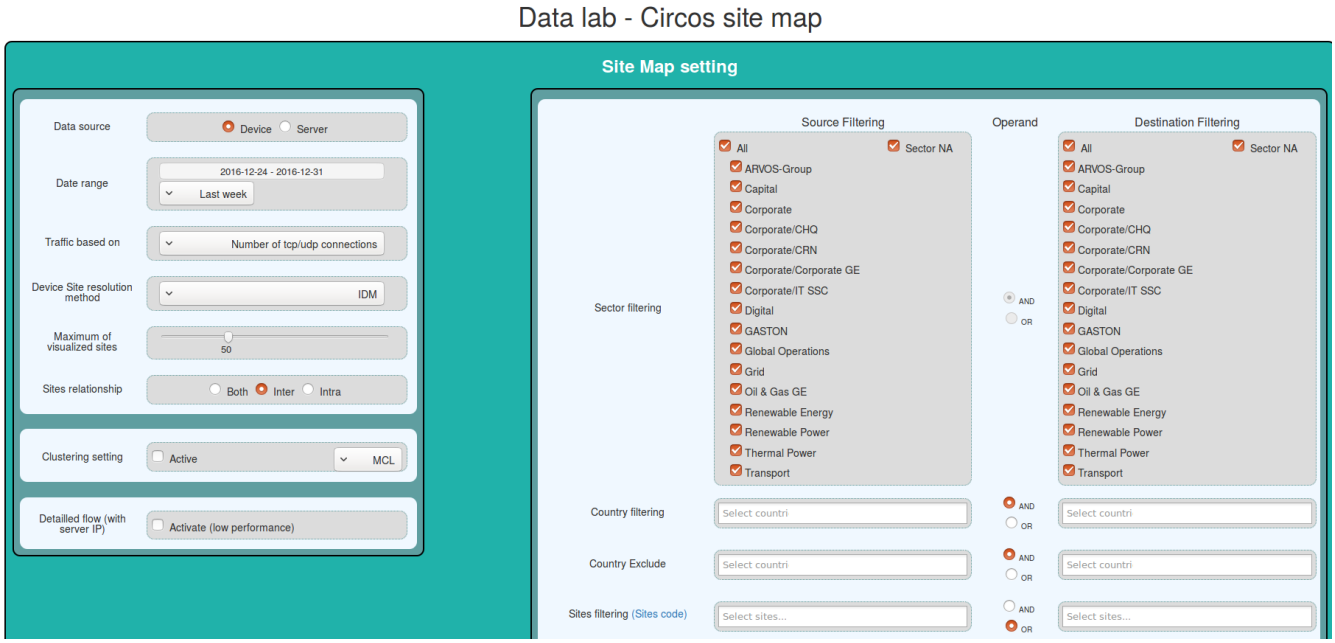


Figure 16. Datalab Circos application

This application is mainly dedicated to carve-out use case. See the following chord diagrams which show data flow between different sectors.

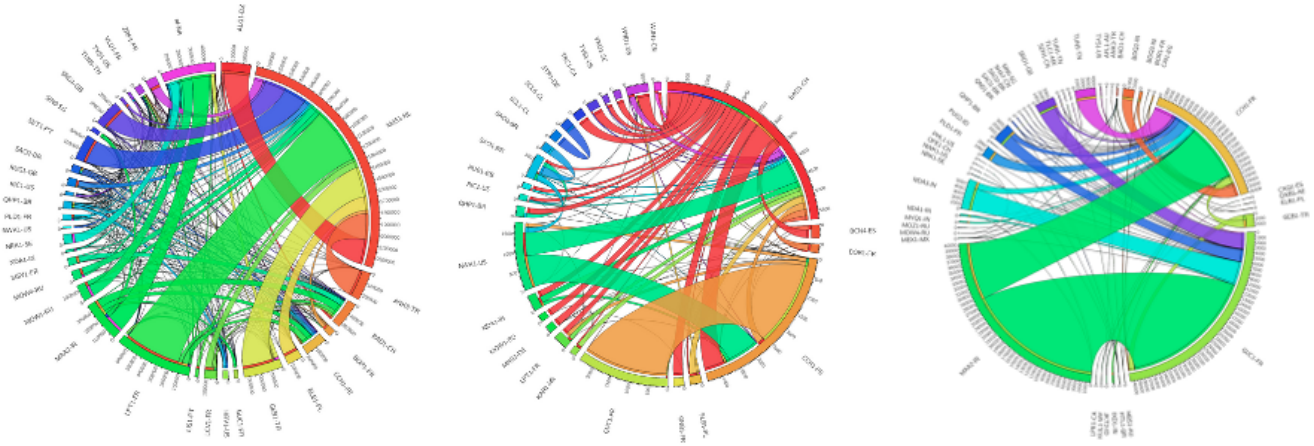


Figure 17. Circos application chord diagrams

It visualizes relationship between sites and extracts flows.

- from device (laptop/desktop) to servers (windows/unix/linux)
- from servers (windows 2003 and later version) to servers (windows/unix/linux)

Access to circos application.

10.4. Kibana

Kibana is another web based application used with Elasticsearch in order to create *dashboard* visuals in real time.

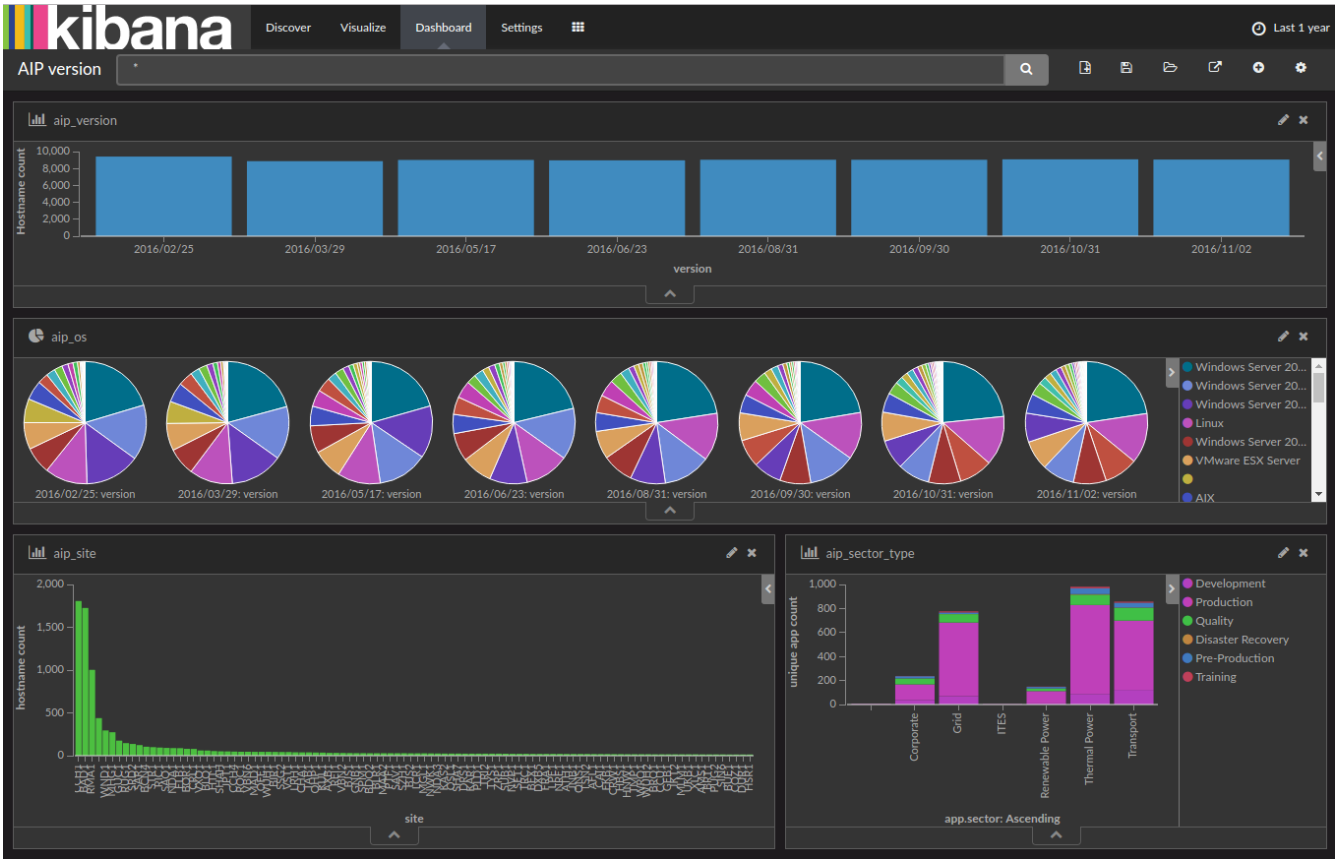


Figure 18. Kibana dashboard example

kibana.gadatalab.com.

10.5. Application Savings Calculator

This simple application lets un know the cost of each resource such as storage, data base server, application server, etc.. on which one or multiple applications are installed. It is very useful to calculate the annual savings in case of a server decommission.

Data consumed by this application come from SEO/AIP or from billing reports. In this application all data are fetched only once from the server side, while it is loaded and all calculations are done on the client side.

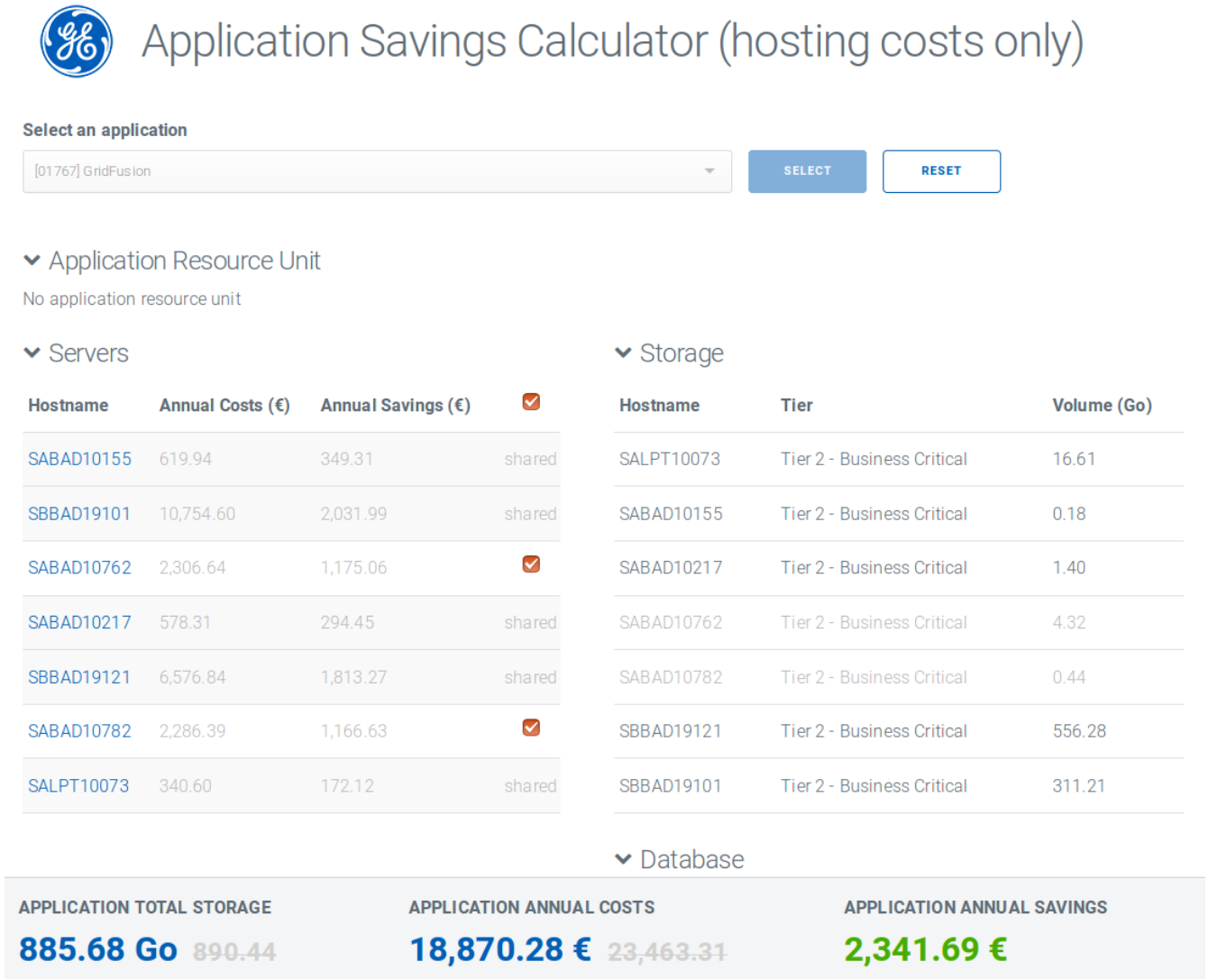


Figure 19. Application savings calculator

Access to circos application.

10.6. Datalab Wiki

Access to the Wiki of Datalab is through : wiki.gadatalab.com.

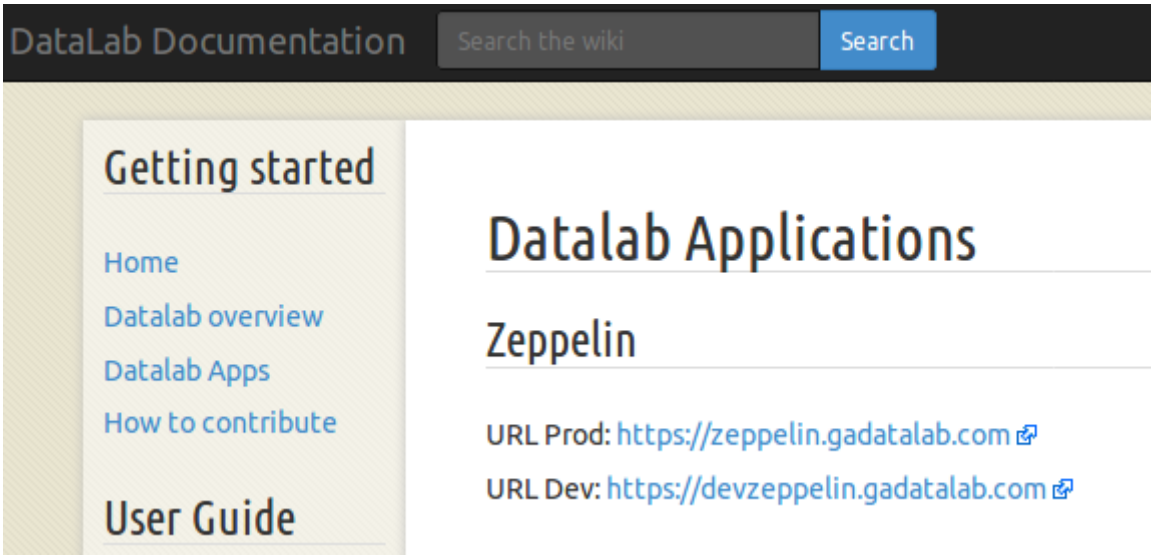


Figure 20. Datalab Project - Wiki

This Wiki gives an overview of the Datalab project, helps to know briefly which are applications used on this project and provides access URLS to them.

You can find some paths to the main repositories on S3 or HDFS in wiki.

11. Datalab Project Applications

11.1. Reverse Proxy Server

In Datalab project we use different web applications in order to consume collected data, do analysis, print visual reports, etc... As you may view on the following illustrated figure, we created a web reverse proxy server which provides secure assess to these applications.

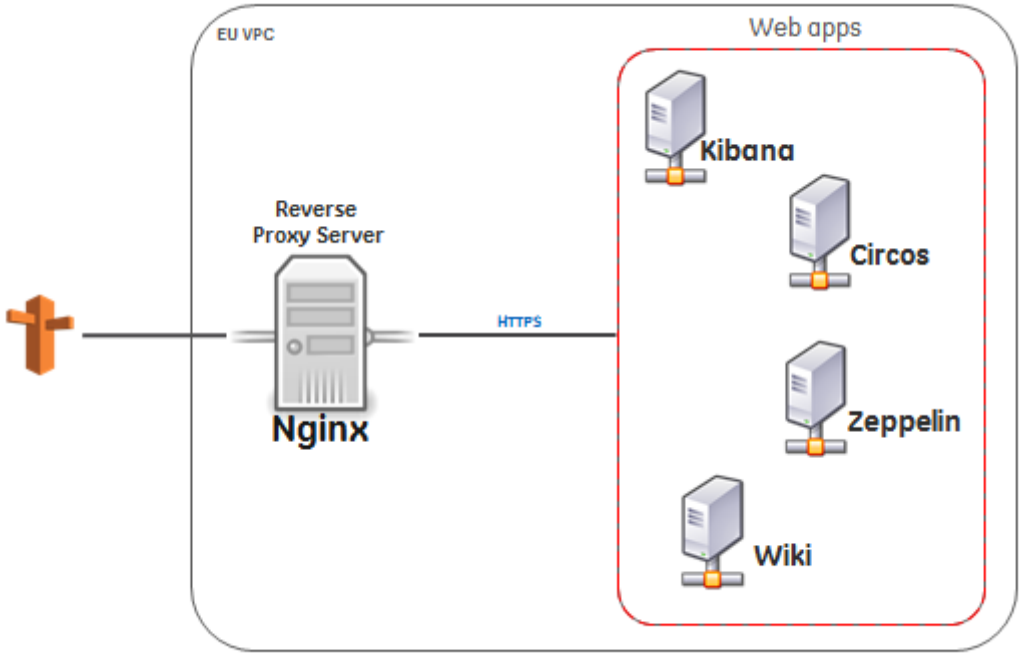


Figure 21. Datalab - Reverse Proxy Server

11.2. Zeppelin

Zeppelin needs to be connected to Apache Spark compute engine which helps to use Hadoop data. Zeppelin is deployed on the same instance of the Master node.

There are 2 Zeppelin environments (dev, prod) in Datalab project.

11.2.1. Backend architecture

As you can see on the following figure, we use about 12 slave node instances in *m1.xlarge* format on AWS with a single *Master node* on which Zeppelin is also installed.

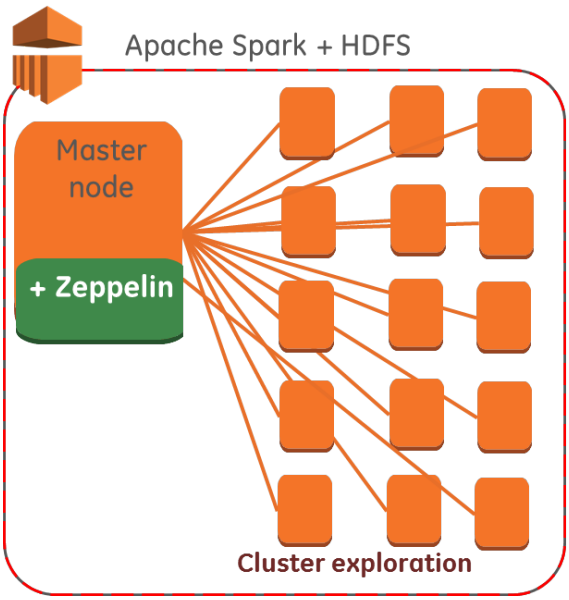


Figure 22. Datalab - Zeppelin on Apache Spark + HDFS cluster

11.2.2. Apache Spark configuration

Spark interpreter must be configured on Zeppelin before being able to use it.

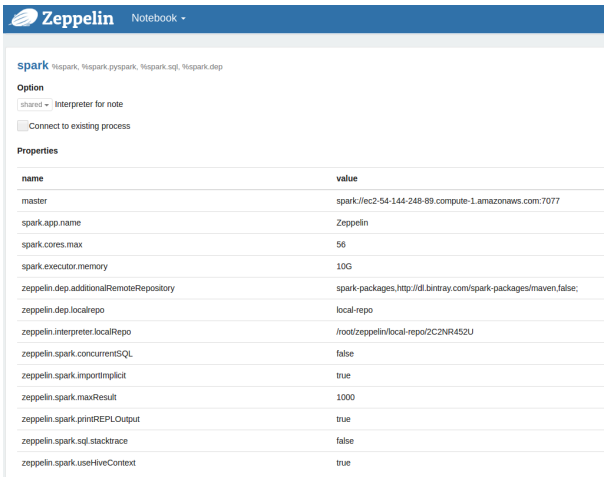
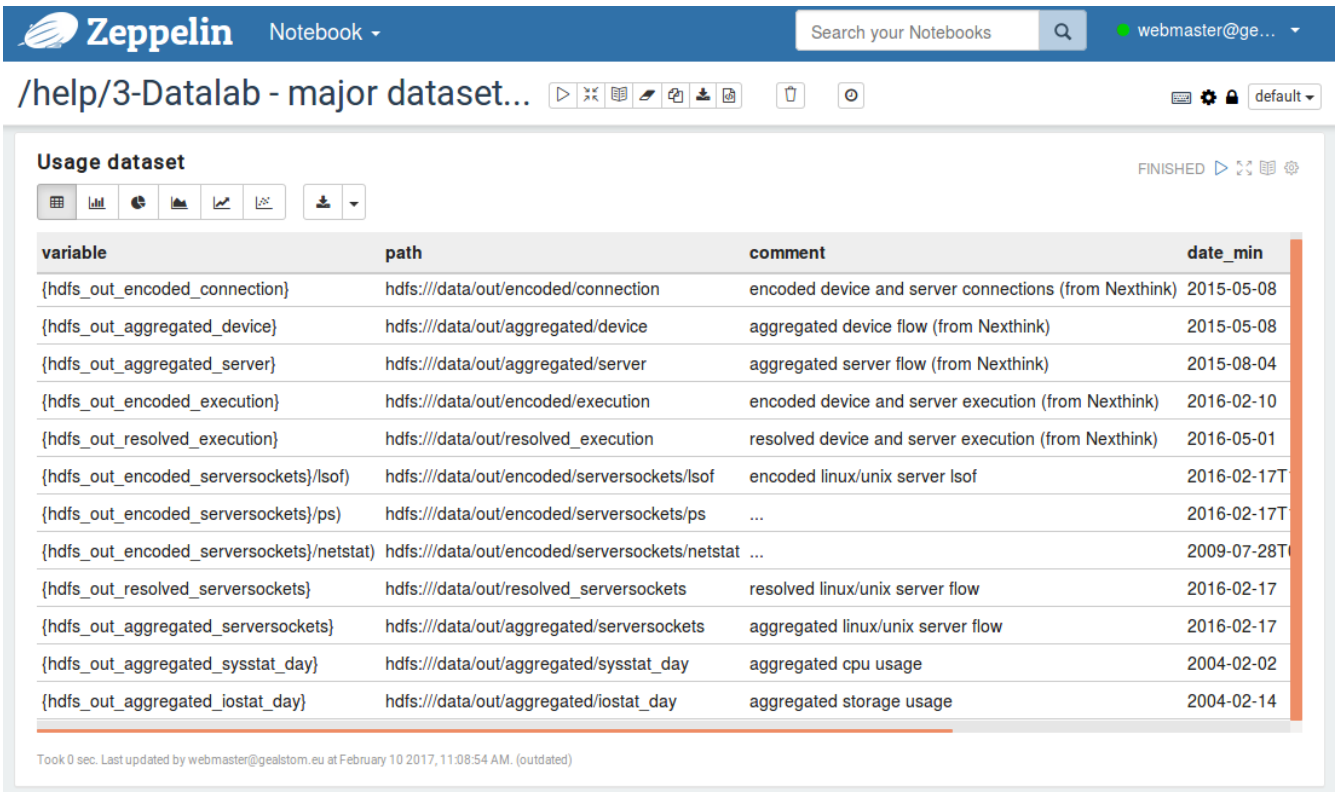


Figure 23. Datalab - Spark interpreter configuration on Zeppelin

11.2.3. Major data sets' path and summary - Help notebooks

In the *help* folder under dev-zeppelin, we collected some important notebooks in which we describe *notebook organisations*, *guidebook*, *tutorials*, *description of major data sets*, etc...



variable	path	comment	date_min
{hdfs_out_encoded_connection}	hdfs:///data/out/encoded/connection	encoded device and server connections (from Nexthink)	2015-05-08
{hdfs_out_aggregated_device}	hdfs:///data/out/aggregated/device	aggregated device flow (from Nexthink)	2015-05-08
{hdfs_out_aggregated_server}	hdfs:///data/out/aggregated/server	aggregated server flow (from Nexthink)	2015-08-04
{hdfs_out_encoded_execution}	hdfs:///data/out/encoded/execution	encoded device and server execution (from Nexthink)	2016-02-10
{hdfs_out_resolved_execution}	hdfs:///data/out/resolved_execution	resolved device and server execution (from Nexthink)	2016-05-01
{hdfs_out_encoded_serversockets}/lsof)	hdfs:///data/out/encoded/serversockets/lsof	encoded linux/unix server lsof	2016-02-17T
{hdfs_out_encoded_serversockets}/ps)	hdfs:///data/out/encoded/serversockets/ps	...	2016-02-17T
{hdfs_out_encoded_serversockets}/netstat)	hdfs:///data/out/encoded/serversockets/netstat	...	2009-07-28T
{hdfs_out_resolved_serversockets}	hdfs:///data/out/resolved_serversockets	resolved linux/unix server flow	2016-02-17
{hdfs_out_aggregated_serversockets}	hdfs:///data/out/aggregated/serversockets	aggregated linux/unix server flow	2016-02-17
{hdfs_out_aggregated_sysstat_day}	hdfs:///data/out/aggregated/sysstat_day	aggregated cpu usage	2004-02-02
{hdfs_out_aggregated_iostat_day}	hdfs:///data/out/aggregated/iostat_day	aggregated storage usage	2004-02-14

Figure 24. Major data sets' path

11.2.4. Notebooks

There are many notebooks created and will be created on both Zeppelin environments. Those notebooks contain different types of code in various languages such as *Bash*, *Spark*, *Scala*, *SQL*, etc.. On the one hand they are part of Datalab project development, on the other hand we don't want to loose them if any accident takes place on AWS servers. This is why we store them in a private repository on Github.

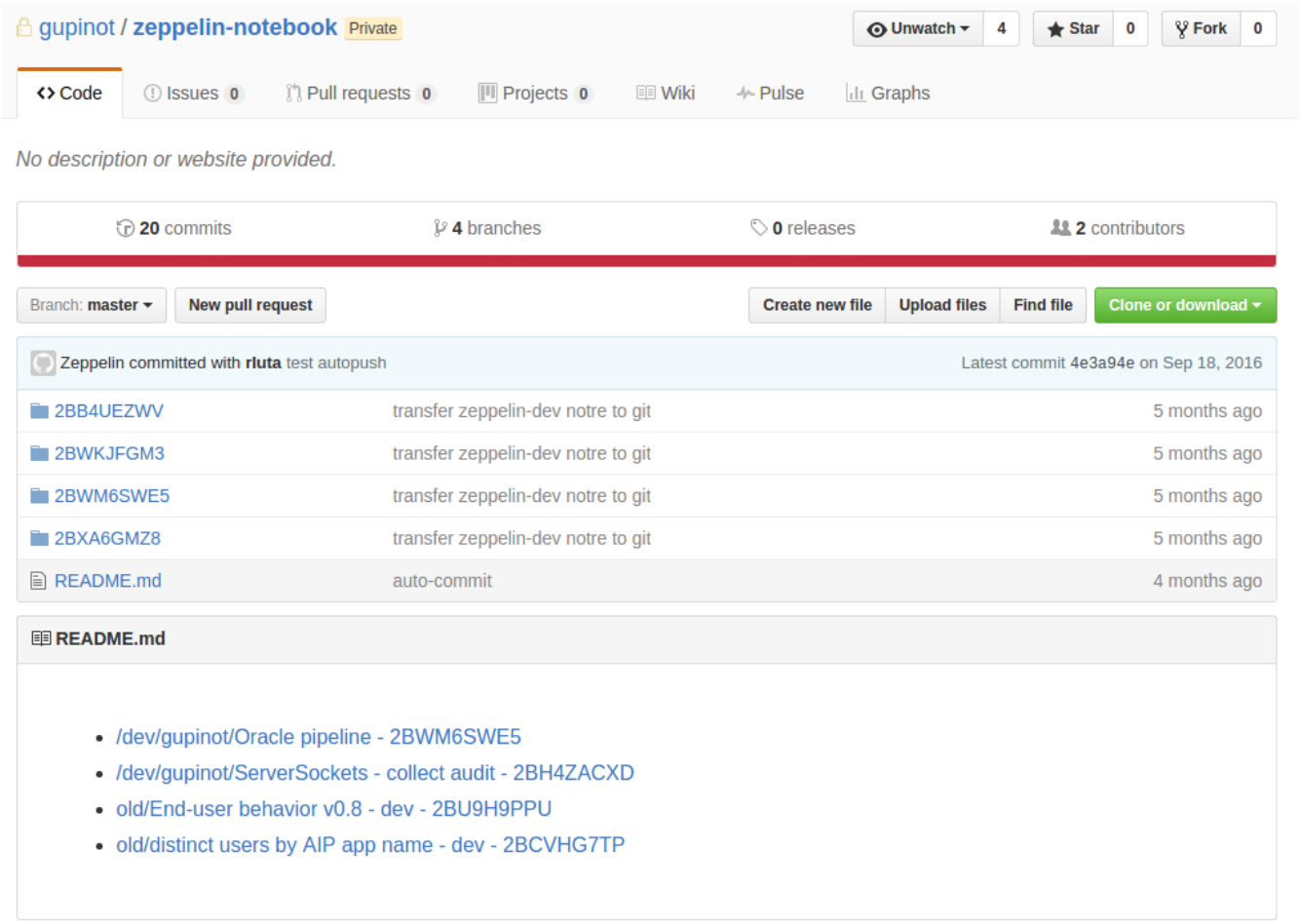


Figure 25. View of a Zeppelin notebook on Github repository

Each Zeppelin notebook content is stored inside of a folder named with its ID (ex: 2BB4UEZWV). As this doesn't allow us to identify the name of the notebooks when we want to restore or import/export them from one environment to the another, we created a link to each notebook folder with its proper name/title. As you can see on the following figure, there is a link for each notebook on the zeppelin-notebook repository's home page on Github.

Notebooks on Zeppelin-dev are stored on a private Github repository. It allows to store each notebook in JSON format and be able to track the changes done on them.

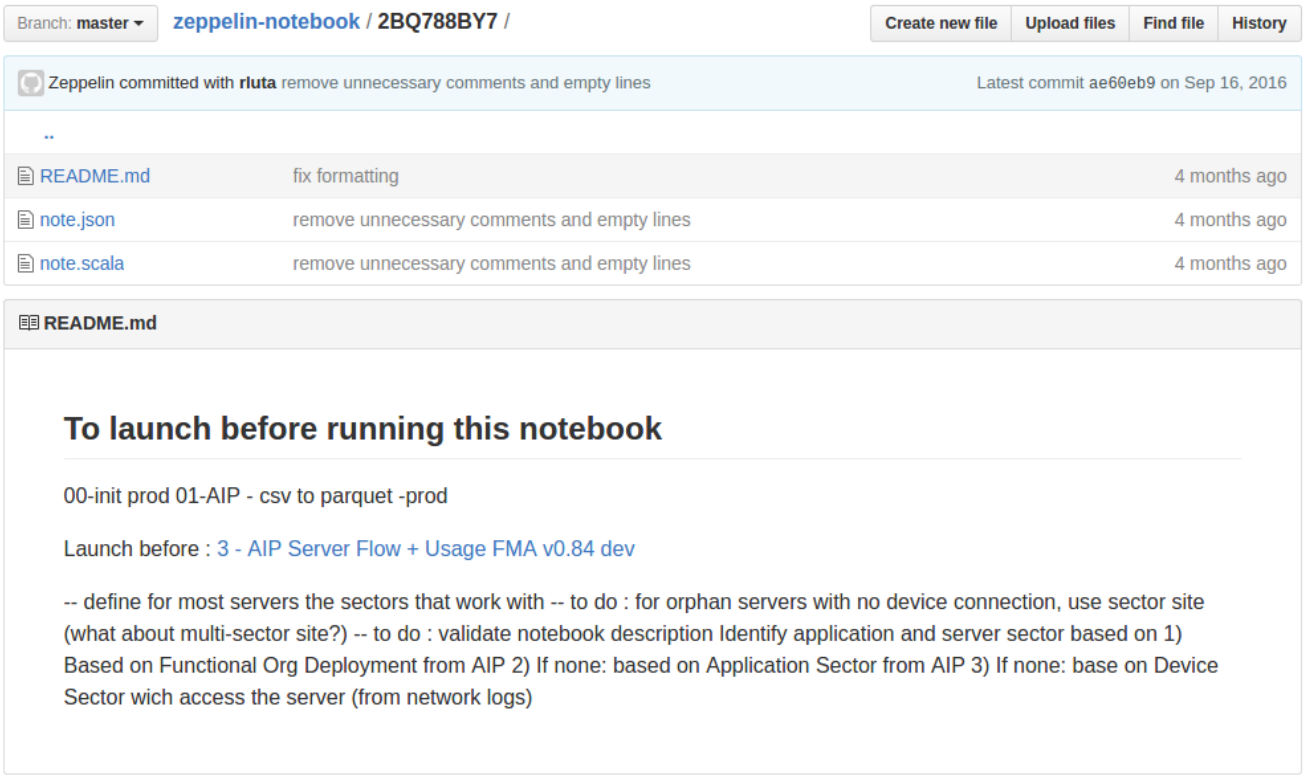


Figure 26. View of a Zeppelin notebook on Github repository

11.3. Zeppelin dev cluster (master + slaves) nodes example

- zeppelin-dev-master-i-981ad300 - m1.xlarge - ec2-54-144-248-89.compute-1.amazonaws.com
- zeppelin-dev-slave-i-0518d19d - m1.xlarge - ec2-54-152-169-50.compute-1.amazonaws.com
- ....
- zeppelin-dev-slave-i-cb18d153 - m1.xlarge - ec2-54-162-212-28.compute-1.amazonaws.com

11.4. Zeppelin prod cluster (master + slaves) nodes example

- zeppelin-prod-master-i-bca7cb22 - m1.xlarge - ec2-54-87-132-205.compute-1.amazonaws.com
- zeppelin-prod-slave-i-1ea6ca80 - m1.xlarge - ec2-54-227-226-162.compute-1.amazonaws.com
- zeppelin-prod-slave-i-65a5c9fb - m1.xlarge - ec2-54-234-214-124.compute-1.amazonaws.com
- zeppelin-prod-slave-i-66a5c9f8 - m1.xlarge - ec2-54-234-193-221.compute-1.amazonaws.com
- zeppelin-prod-slave-i-7fa5c9e1 - m1.xlarge - ec2-54-234-192-94.compute-1.amazonaws.com

11.5. Create cluster on AWS

In order to create HDFS file system with master and slave server instances on AWS, you need to referrer to the ./infra/README.md file. Either for production or development environment, the detailed command line code is given on that file as below:

infra/README.md

```
**Create cluster** :
  - cd spark; ./spark-ec2 -k KeyLezoomerUs -i ~/.ssh/KeyLezoomerUs.pem --pipeline-version=1.3.3 --scala-version 2.11
--region=us-east-1 --zone=us-east-1b --instance-type m1.xlarge --master-instance-type m1.xlarge --spot-price 0.13
--master-spot-price 0.13 --deploy-env dev --zeppelin-bucket gecustomers --es-security-group elasticsearch-discovery
--deploy-profile customers --instance-profile-name customers -s 10 launch zeppelin-dev
  - on web proxy server, associate new master IP to dev.zeppelin upstream in /etc/nginx/conf.d/upstreams.conf file and
restart nginx
  **Import data** : ssh root@zeppelin-master "cd pipeline/bin; ./syncHdfsS3.sh fromS3Simple"
  manual path : chmod 777 /mnt*/spark
</br>
```

Keep in mind that the master IP will be changed if the HDFS architect is re-created. You need then associate this IP with *dev.zeppelin* or *zeppelin* so that these platforms could be reachable.

infra/README.md

```
  - on web proxy server, associate new master IP to dev.zeppelin upstream in /etc/nginx/conf.d/upstreams.conf file and
restart nginx
```