DataLab - Enterprise IT Digital twin

Guillaume PINOT, Matin BAYRAMOV

bayramov.matin-ext@power.alstom.com

Table of Contents

Presentation of DataLab

The main purpose of this project is ….

Global View

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.

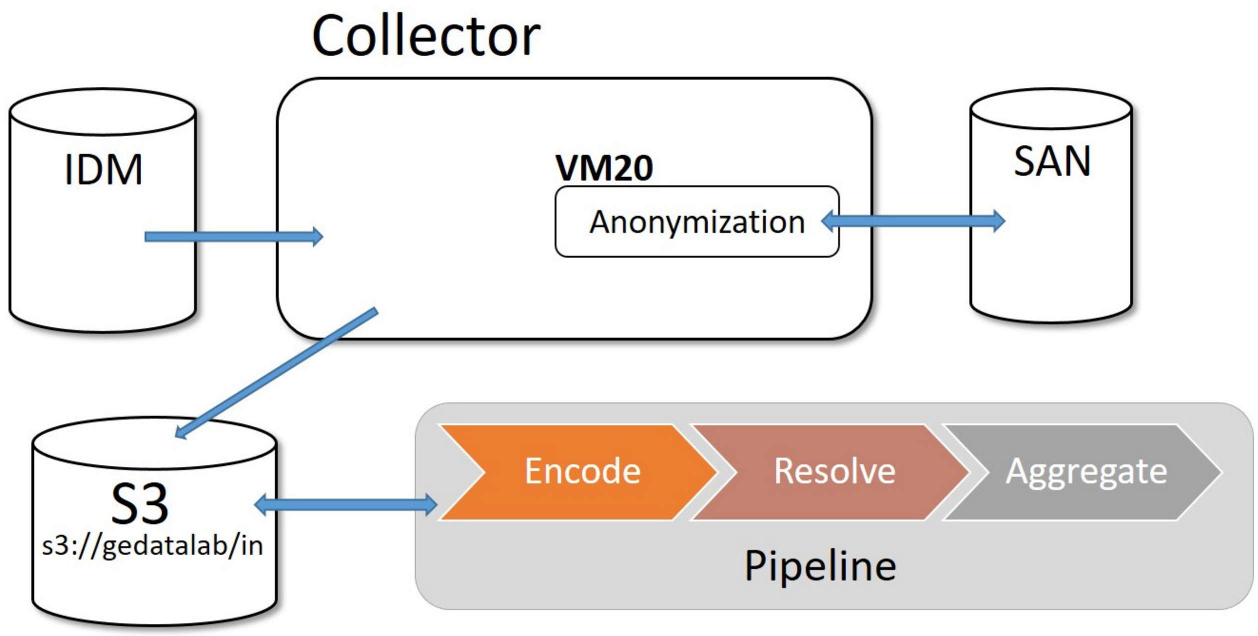

Figure 1. Global view of the common processes

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".

References

In this project we have 3 main types of referenced data collection. These are *AIP*, *MDM-ITC* and *Storage Master Report* which are kept on s3.

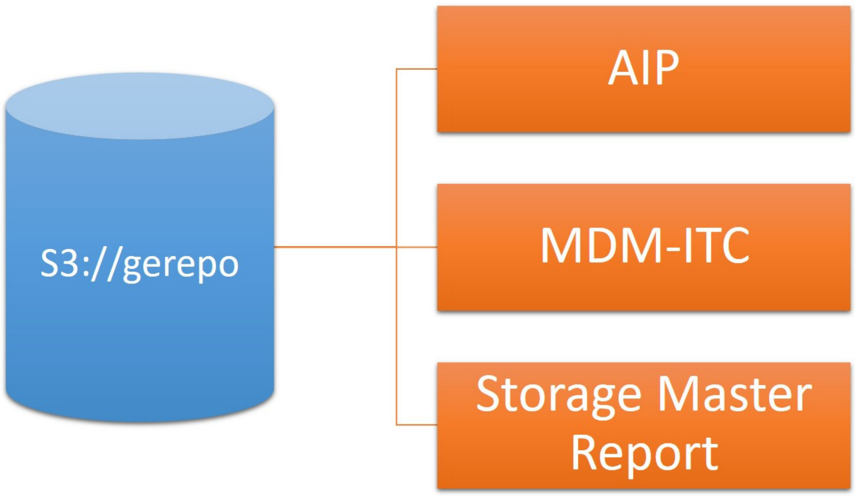

Figure 2. Reference storage on s3

AIP

AIP 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 Alstom and it covers mainly the list of applications, the list of servers where the applications are deployed, the product software used by these applications, the licenses used by the applications, etc.

There are 3 main tables which are :

- *server*: list of servers
- *application*: list of business applications
- *softInstance*: which application is deployed on which server

Data are synchronized daily by batches and they are stored under s3://gerepo/.

Zeppelin notebook URLs are

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

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.

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

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

IDM

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.

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

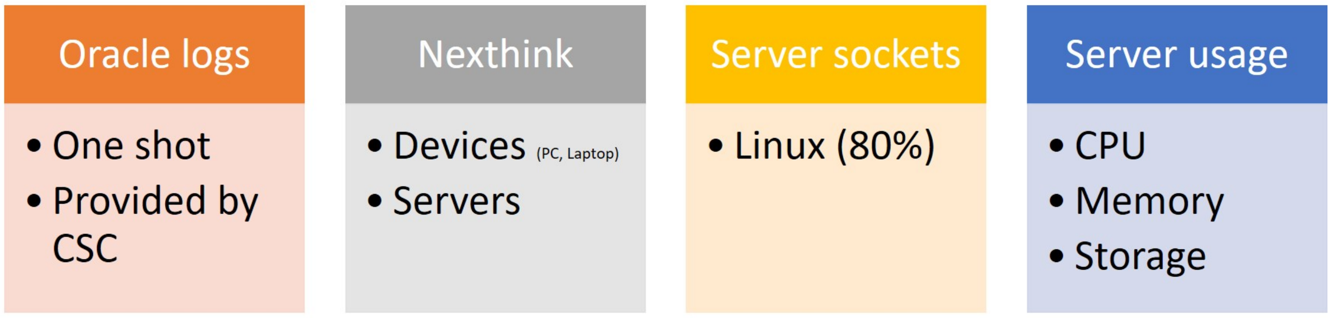

Figure 3. Types of data sources to collect

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.

Nexthink

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]

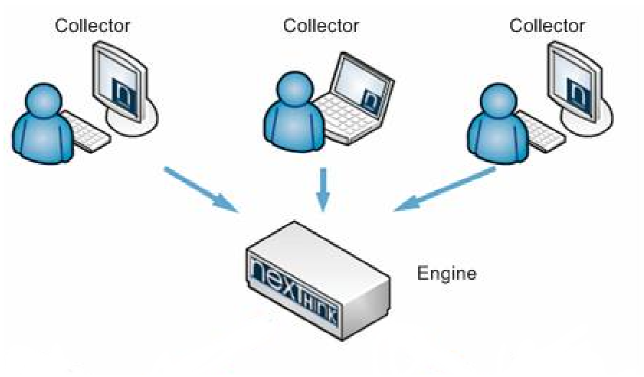

Figure 4. Nexthink Engine which collects data from different devices

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.

Where are data collection on Nexthink side ?

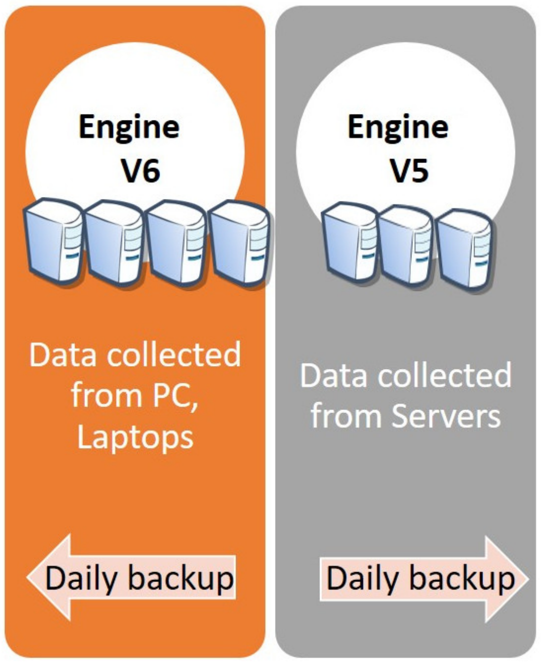

Figure 5. Nexthink Engines with backup servers

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.

How we get and store input data on our side ?

Collector server (SARMA10012) copies these data from Nexthink backup servers to our SAN server every 3 days. Each received file can contain log history of 5 to 20 days.

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.

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.

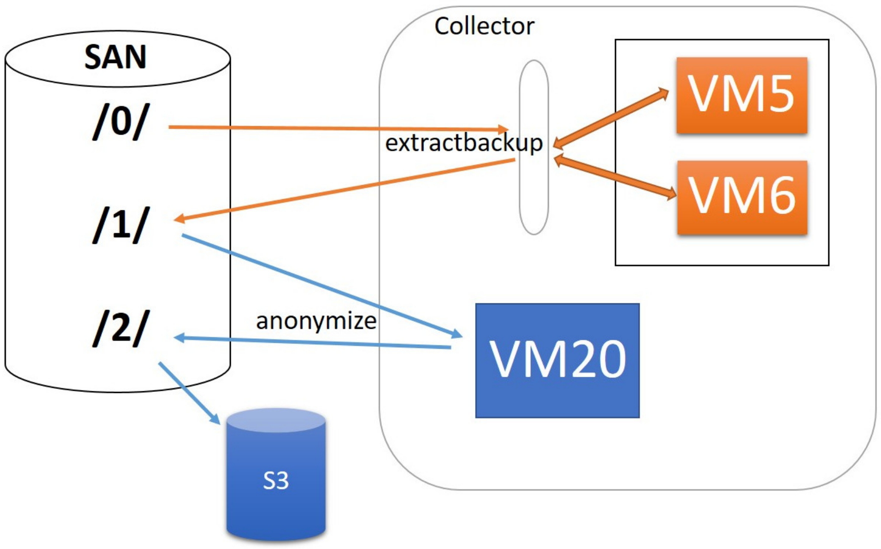

Figure 6. Data extraction and anonymization

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..)

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)

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"

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.

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.

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

Deployment

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.

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.

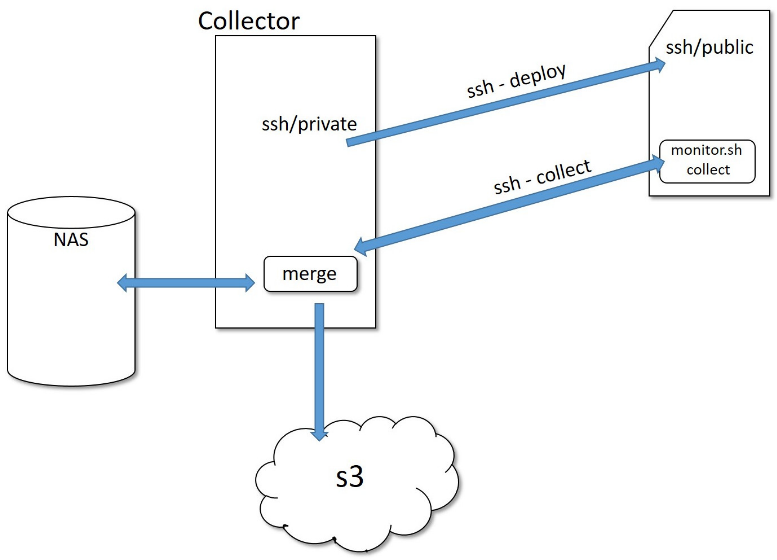

Figure 7. Collecting and deployement of Server sockets data

Anonymization

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

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.

An HTTP URL allows us to call and get these data for a single given server by specifying its hostname as a URL parameter. As response, it is an XLS file returned which will be then converted to CSV format and stored on s3.

To know more about URL and other configurations, look into conf/conf.sh file.

There is a *server.lst* CSV file which contains hostname of all servers for which we want to collect their usage information. Server-usage data are synchronized 2 times in a week.

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

- Source code path : ALDataLab/collect/serverconsumption.
- URL to call : http://iww.dcs.itssc.alstom.com/nrtmd/streamsdump/server

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.

Pipeline folder structure

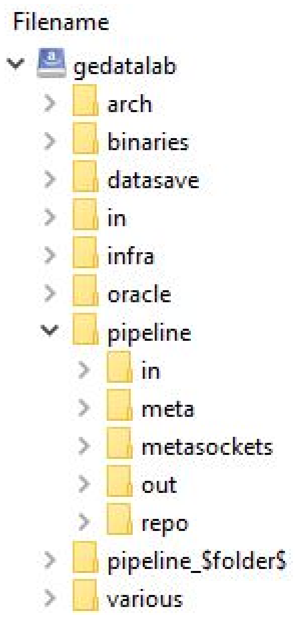

Figure 8. Pipeline folder structure under s3 server.

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.

Pipeline main actions

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.

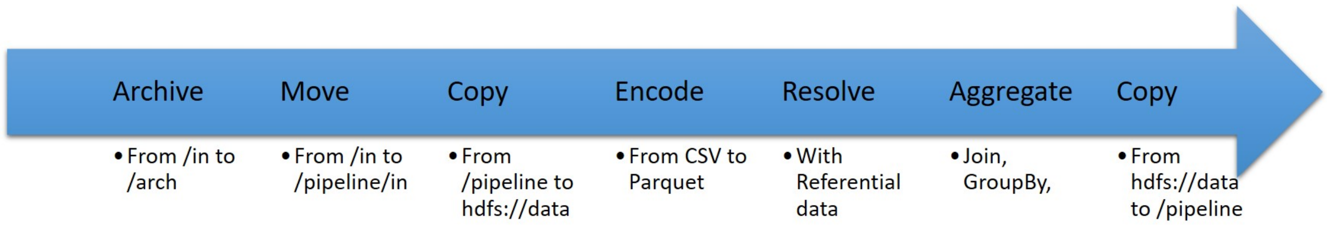

Figure 9. Order of main actions taken place in Pipeline

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.

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.

Oracle Logs

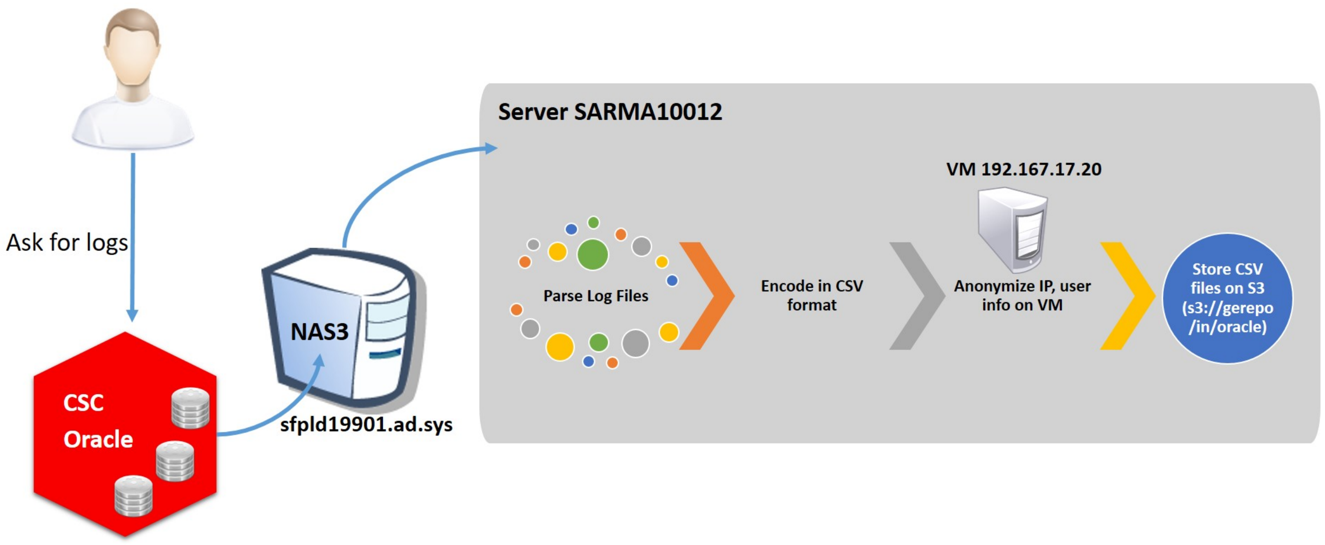

Figure 1: Global view of the processes

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.

Project general file organization

We organised files by respecting to the following folder structure.

- "in" folder is mainly used to keep source files.
- "out" folder is used to save output data
- "done" folder is intended to save data once all processes are finished.

This convention is respected on all over the project.

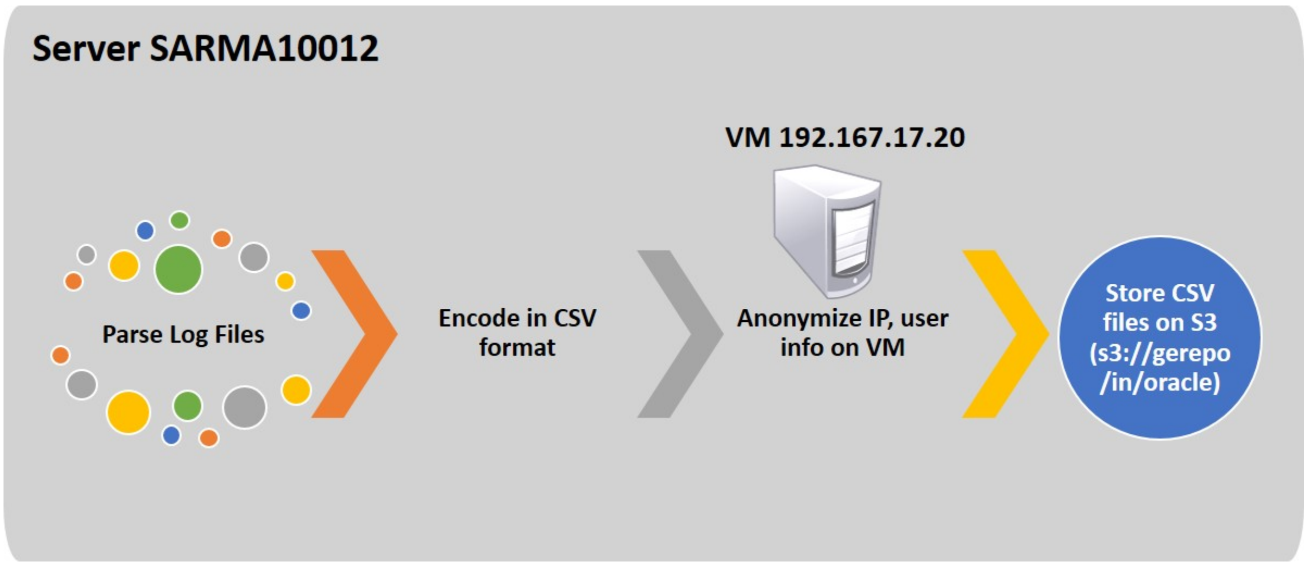
Server SARMA10012
VM 192.167.17.20
Parse Log Files
Encode in CSV format
Anonymize IP, user info on VM
Store CSV files on S3 (s3://gerepo /in/oracle)

Figure 2: Processus on the SARMA10012 server

Server SARMA10012

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.

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.

TODO : user datalab:

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.

TODO : (User nexthink:)

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

Virtual Machine - 192.167.17.20

TODO : VM20

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.

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.

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.

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

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.